Adaptive Comparison-Based Algorithms for Evaluating Set Queries

by

Mehdi Mirzazadeh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2004

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis we study a problem that arises in answering boolean queries submitted to a search engine. Usually a search engine stores the set of IDs of documents containing each word in a pre-computed sorted order and to evaluate a query like "computer AND science" the search engine has to evaluate the union of the sets of documents containing the words "computer" and "science". More complex queries will result in more complex set expressions. In this thesis we consider the problem of evaluation of a set expression with union and intersection as operators and ordered sets as operands.

We explore properties of comparison-based algorithms for the problem. A *proof of a set expression* is the set of comparisons that a comparison-based algorithm performs before it can determine the result of the expression. We discuss the properties of the proofs of set expressions and based on how complex the smallest proofs of a set expression E are, we define a measurement for determining how difficult it is for E to be computed. Then, we design an algorithm that is adaptive to the difficulty of the input expression and we show that the running time of the algorithm is roughly proportional to difficulty of the input expression, where the factor is roughly logarithmic in the number of the operands of the input expression.

Key words: Adaptive algorithm, comparison-based algorithm, search engines, algorithms.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. First and foremost, my thanks are due to my supervisor, Professor Naomi Nishimura, for her guidance and patience. She left me enough freedom to select my own subject and was always available to read my writings and to give useful comments. I owe my deep regards to her for helping me to bring this thesis in the present form. I would like to thank Professor Alex Lopez-Ortiz for introducing the problem to me and for his useful discussions on the issue. I am also so grateful to the readers of my thesis, Professors Alex Lopez-Ortiz and Ian Munro for reading the big volume of the thesis fast enough so that I can be graduated on time.

I appreciate supports and encouragements I received from my friends, both in Waterloo and in other points, during my Master's career. Special thanks to Omran Ahmadi, Amjad Ashoorioon, Ehsan Chiniforooshan, Reza Dorri, Arash Farzan, MohammadTaghi Hajiaghayi, Abbas Heydarnoori, Majid Khabbazian, Vahab Mirrokni, Bashir Sadjad, MohammadAli Safari, Shahram Esmaeilsabzali, and many other good friends. Last not least, I would like to dedicate this thesis to my family, my parents and my two sisters, for their unfailing love and never-ending support.

May God bless you all.

Contents

List of Figures

Chapter 1

Introduction

Traditionally, the time or space complexity of an algorithm is measured by taking the maximum cost of running the algorithm on all instances of a given size, without taking into consideration special properties of instances, other than size, that may affect the cost of running any algorithm. This means that when we are talking about the worst case time complexity, the complexity of the algorithm is determined by just looking at the most difficult instance of every size for that algorithm, ignoring how efficiently the algorithm works for other instances.

For the purpose of illustration, let us use an example to show how we can consider different levels of easiness for instances. Consider the problem of evaluating the intersection of two ordered sets. Given two sorted arrays A and B of size n, the best comparison-based algorithm needs to perform at least $2n - 1$ comparisons in the worst case. That happens when we consider a sorted sequence of $2n$ integers, select integers in odd positions as the sequence stored in the array A, and select integers in even positions as the sequence stored in B. This can be considered as a difficult instance of the problem. But we can construct an easy instance by picking the first n elements of the sequence for the first array and picking the next n elements for the second array. One can show that the intersection of the two sets is empty using just one comparison.

Adaptive algorithms [Meh84] were introduced to solve the easy instances in a short time while at the same time not increasing the worst-case running time. They are adaptive to the hardness of the input and take advantage of easiness of instances. A *difficulty function*

over the set of all possible instances is defined, which is increasing with both size and hardness of the instance. The complexity of the algorithm is measured by looking at the ratio of the running time to the difficulty parameter of all instances of a given size.

We can consider sorting as the seminal problem in the area of adaptive algorithms [GMPR77]. The amount of disorder in an instance is the criterion for difficulty of the problem and many different functions for measuring the amount of disorder have been proposed. The number of exchanges required to sort the input, the maximum difference between indexes of an element in the input and in the sorted order, and the minimum number of monotone subsequences of the input partitioning the input are some examples of measurements of the amount of disorder. For groups of these measurements a number of generic algorithms have been proposed [ECW92].

There are not many results on adaptive algorithms in other areas. The work of Estivill-Castro and Gasca-Soto on adaptive algorithms for the shortest path and minimum spanning tree problems in directed weighted graphs [ECGS97] is one of the few examples. They explored different properties of an input that make the input difficult for Dijkstra's Algorithm, in the case of shortest path problem, and make the input difficult for Prim's and Kruskal's algorithms, in the case of minimum spanning tree problem. For both problems, they showed that the concept of instance easiness is closely related to the concept of instance easiness in sorting problem.

The problem we will discuss in this thesis, the problem of evaluating set queries, has significant applications in the database area, especially for search engines. Search engines like Google store the set of all documents containing each word [BP98]. Then, if the user requires the list of documents containing all or some of a given set of words, the search engine has to evaluate the intersection or the union of the sets corresponding to the given words. More complex queries consisting of logical operators like "and" and "or" yield the problem of evaluating more complex set expressions. Demaine, Lopez-Ortiz, and Munro considered a simple version of the problem in which a collection of sets is given and the intersection or the union of the sets is required to be evaluated. They studied the running time of adaptive algorithms for the problem and designed an adaptive algorithm [DLOM00, DLOM01] for the problem. The running time of their algorithm matches their definition of difficulty of instances. Also, they showed that for any algorithm

$\mathcal{A}$ and any integer n, one can construct an input E of size at least n such that the number of comparisons that $\mathcal{A}$ performs before it solves E is at least the difficulty of E.

A few variants of the simple version of the problem, defined above, have been studied. Barbay [Bar03] considered the problem of evaluating the intersection of the given sets and proved that, compared to deterministic ones, randomized algorithms perform better in terms of the number of comparisons on average. Barbay and Kenyon also studied a more general problem called the *t-threshold problem* in which a collection of sets is given and the problem is to compute the set of all values appearing in at least t sets [BK02, BK03]. They introduced a measurement for determining the difficulty of instances of the t-threshold problem and they presented an algorithm working in $O(tD \log k \log n)$ time where D, k, and n are the difficulty of the input, the number of input sets, and the total number of members of input sets, respectively. They also showed that, given an algorithm $\mathcal{A}$ and integers k, n_1, ..., and n_k, one can construct an input E with k sets of sizes at least n_1, ..., n_k such that $\mathcal{A}$ is required to perform at least $\Omega(D \sum_{i=1}^{k} \log \frac{n_i}{D})$ comparisons to evaluate E where D is the difficulty of E.

In this thesis we will study comparison-based algorithms for the problem of evaluating an expression with intersection and union operators and ordered set operands. We propose a definition for the difficulty of instances and then we present an algorithm whose running time of a given input E is not far from the difficulty of E. We now explain how the thesis is organization, chapter by chapter.

In Chapter 2, we formally define the problem, describe our assumptions about the input, and specify input and output formats. Afterward, we present some basic definitions and lemmas that will be used throughout the thesis. In Section 2.5 we explain that in some special cases we have not solved the problem completely. Finally, at the end of Chapter 2, we define our difficulty function. Similar to the definitions of difficulty functions proposed by Demaine et al. [DLOM00], our definition of the difficulty of instances is based on the set of comparisons that an algorithm should perform on the members of the input ordered sets to obtain the result of the input expression. The term "proof" is used for the set of comparisons after performing which, an algorithm that always output a correct solution may stop running and report its output [DLOM00]. For every possible proof $\mathcal{P}$ for a given input we define the cost of $\mathcal{P}$ as an integer, intuitively showing the time taken by an

algorithm to perform the comparisons in $\mathcal{P}$. Then, the difficulty of the instance will be the minimum cost of all of its proofs.

Chapter 3 introduces a non-deterministic algorithm for the problem that scans members of the input ordered sets from the beginning and while scanning the set members, it generates the result of the input expression. At each point, the algorithm non-deterministically generates a set of evidence called an "eliminating step" either showing some of the smallest non-scanned members are in the result of the expression and so they should be reported to the output or showing some of the smallest non-scanned members are not in the result of the expression and so they can be ignored. The algorithm reports this group of members to the output if necessary and then scans them. Each sequence of eliminating steps generated by the algorithm that leads the algorithm to find the result of the input expression will be called an "eliminating proof". This concept is closed to the concept of "eliminating proof" defined for the simple version of the problem [DLOM00]. We will discuss some properties of eliminating proofs in Chapter 3.

In Chapter 4 we show how a proof can be converted to an eliminating proof. Then, in Chapter 5 we present a deterministic version of the algorithm of Chapter 3 that using dynamic programming processes all possible sequences of eliminating steps. Then, given a proof $\mathcal{P}$ for the input, Chapter 6 compares the running time of the algorithm with the cost of $\mathcal{P}$ using the eliminating proof constructed from $\mathcal{P}$ in Chapter 4. Defining $D(\mathcal{P})$ and $V(\mathcal{P})$ as the cost of $\mathcal{P}$ and the number of members of the input sets participating in comparisons in $\mathcal{P}$, respectively, the running time of the algorithm will be of the form $O(D(\mathcal{P})h+V(\mathcal{P})f+g)$ where h is roughly logarithmic in the number of the operands of the input expression and f and g are two functions of the input expression but independent of the sizes of the input sets. Finally, in Chapter 7 we will summarize the work and include our concluding remarks.

Chapter 2

Basics

2.1 Notation

Let us first introduce the notation used to deal with sequences throughout the thesis. Given a sequence $\Sigma = x_1, \ldots, x_n$, we use length($\Sigma$) to denote the length of the sequence Σ. Also, the next two definitions will be used for appending a sequence or a single item to the end of a sequence.

Definition 2.1 *Given two sequences $\Sigma_1 = x_1, \ldots, x_m$ and $\Sigma_2 = y_1, \ldots, y_n$, we use $\Sigma_1 \circ \Sigma_2$ to denote the sequence $x_1, \ldots, x_m, y_1, \ldots, y_n$.*

Definition 2.2 *Given a sequence $\Sigma = x_1, \ldots, x_m$ and an item y, we use $\Sigma \odot y$ to denote the sequence $x_1, \ldots, x_m, y$.*

The next observation follows immediately from the definition.

Observation 2.1 *Consider a sequence $\Sigma = x_1, \ldots, x_m$ and an item y. If Γ denotes the sequence having just one element y, then $\Sigma \circ \Gamma = \Sigma \odot y$.*

2.2 The Problem

2.2.1 The Instance

We will study the problem of designing an adaptive algorithm for the problem of evaluating a given set expression in which each operator is an intersection or a union operator and each operand is an ordered set. The result of this evaluation is required to be an ordered set also. Speaking informally, we represent the set expression using a tree in which each internal node corresponds to an appearance of a set operator in the expression and each leaf is assigned one of the operands. Every operand is represented by a sequence of objects and each object has a value which is a member of a universal ordered set $\mathbb{V}$. In our representation, the sequences corresponding to every two operands have no object in common. This does not reduce the generality of the problem as different objects are not forced to have distinct values by this assumption. We denote the problem by *SimpleEval*.

Every instance of SimpleEval consists of three components. The first component is a query tree defined as follows:

Definition 2.3 *A* query tree *is a rooted tree Q in which every internal node is of one of the following two types:* union nodes *and* intersection nodes. *Also, no union node is a child of a union node and no intersection node is a child of an intersection node. A query tree Q is* normalized *if every internal node of Q has at least two children.*

Given two nodes u and v of a query tree Q, if v is on the path connecting u to the root of Q, we say v is an *ancestor* of u and u is a *descendant* of v. If $u \neq v$, then v is a *proper ancestor* of u and u is a *proper descendant* of v. A subtree T of Q is a *complete subtree* if for every node w of Q, if w is in T, every descendant of w is also in T. Given a node w of Q, the complete subtree of Q rooted at node w is denoted by $Q[w]$. Given a subtree U we denote the set of node, the set of leaves, the set of union nodes, and the set of intersection nodes of U by $\mathit{nodes}(U)$, $\mathit{leaves}(U)$, $\mathit{unions}(U)$, and $\mathit{intersections}(U)$, respectively.

The second component is the function specifying the sequence of objects corresponding to each operand.

Definition 2.4 *Given a query tree Q, an* object function for Q *is a function ω that associates a non-empty sequence of objects with each leaf of Q such that no object appears in*

more than one such sequence.

The third component is a value function determining the value of each object. We define there to be two values $-\infty$ and ∞ in $\mathbb{V}$ such that $-\infty$ is less than every other value in $\mathbb{V}$ and ∞ is greater than every other value in $\mathbb{V}$.

Definition 2.5 *Given a query tree Q and an object function ω, a* value function for (Q, ω) *is a function μ that assigns a value from $\mathbb{V} - \{-\infty, \infty\}$ to each object appearing in $\omega(l)$, for every leaf l of Q. A value function μ is* ordered *if for every two consecutive objects o_1 and o_2 of $\omega(l)$, for a leaf l of Q, $\mu(o_1) < \mu(o_2)$.*

The value assigned by a value function μ to an object is called the *μ-value* or just the *value* of that object.

Definition 2.6 *A* disordered instance *is a triple $I = (Q, \omega, \mu)$ where Q is a query tree, ω is an object function for Q, and μ is a value function for (Q, ω). A disordered instance I is an* instance *if μ is ordered. A disordered instance (Q, ω, μ) is* normalized *if Q is normalized.*

The *signature of a disordered instance $I = (Q, \omega, \mu)$* is defined as the pair (Q, ω). Given a disordered instance $I = (Q, \omega, \mu)$, for every leaf l of Q, every object appearing in $\omega(l)$ is an *object of I* and also is an *object of the signature* (Q, ω). The set of all objects of I is denoted by *Objects*(I). In addition, when we talk about a specific disordered instance (Q, ω, μ), by an object of a leaf l of Q we mean an object in $\omega(l)$. Also, in this situation, if o is an object of a leaf l of Q, we may say l is the leaf containing o.

Given a disordered instance $I = (Q, \omega, \mu)$ and a leaf l of Q, we now number objects appearing in $\omega(l)$ with numbers between 1 and the length n of $\omega(l)$. Also, to handle boundary cases in proofs, in the next definition, we define two new entities as the 0th and the $(n+1)$st elements of $\omega(l)$.

Definition 2.7 *We define two new objects* BEG *and* END *that are not objects of any disordered instance. Also, given a sequence $\Sigma = o_1, \ldots, o_n$ of objects, for every i, $0 \leq i \leq n+1$, we define $\Sigma[i]$ as follows:*

$$\Sigma[i] = \begin{cases} \text{BEG} & \text{if } i = 0 \\ \text{END} & \text{if } i = n+1 \\ o_i & \text{otherwise} \end{cases}$$

Definition 2.8 *Given a disordered instance $I = (Q, \omega, \mu)$, a leaf l of Q, and a member o of $\{\text{BEG}, \text{END}\} \cup Objects(I)$ such that $\omega(l)[i] = o$, for some i, $0 \leq i \leq \text{length}(\omega(l) + 1)$, we define index$(I, l, o) = i$.*

Now we move towards defining formally what we expect an algorithm designed for SimpleEval to compute.

Definition 2.9 *Given an instance $I = (Q, \omega, \mu)$ and a node v of Q, the value set of v in I is a subset of $\mathbb{V}$ defined as follows. If v is a leaf of Q,* the value set of v in I *is the set of μ-values of objects in $\omega(v)$; otherwise* the value set of v in I *is the union or the intersection of value sets of children of v, depending on the type of v.*

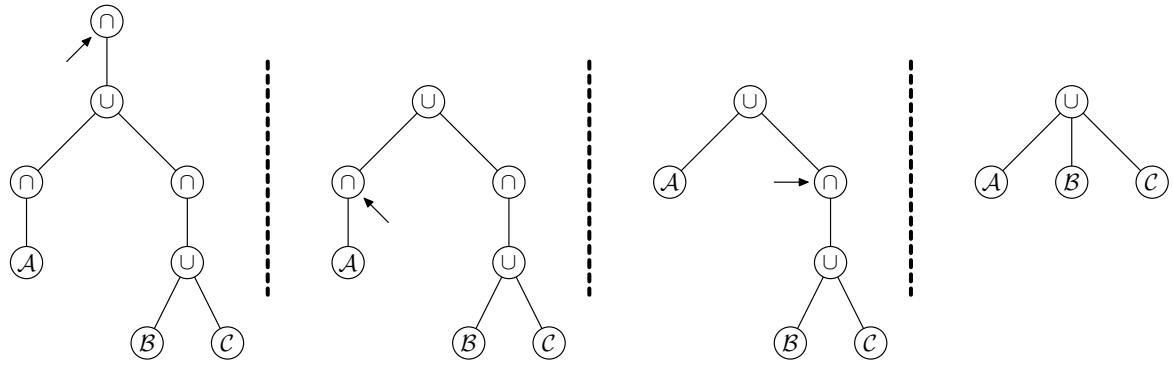
We now prove that the definition of the value set of a node v of a query tree Q depends only on values of objects in sequences associated with leaves in $Q[v]$.

Lemma 2.2 *Consider instances $I = (Q_I, \omega_I, \mu)$ and $J = (Q_J, \omega_J, \nu)$ and a node v appearing in both Q_I and Q_J such that $Q_I[v]$ and $Q_J[v]$ are the same. Also suppose, for every leaf l of $Q_I[v]$, $\omega_I(l) = \omega_J(l)$ and, for every object o of $\omega_I(l)$, $\mu(o) = \nu(o)$. Then, the value set of v in I is the same as the value set of v in J.*

Proof. We use induction on the height of v to prove the lemma. If v is a leaf, it follows from the assumption of the lemma that the set of μ-values of objects of l and the set of ν-values of objects of l are the same and hence by Definition 2.9 the lemma is correct. Now suppose v is an intersection (a union) node with k children $u_1, \ldots, u_k$ and the lemma is true for all children of v. Then, given a child u_i of v, as $Q_I[v] = Q_J[v]$, $Q_I[u_i] = Q_J[u_i]$. Furthermore, since $leaves(Q_I[u_i]) \subseteq leaves(Q_I[v])$, for every leaf l of $Q_I[u_i]$, $\omega_I(l) = \omega_J(l)$ and for every object o of $\omega_I(l)$, $\mu(o) = \nu(o)$. Thus, by induction the value set of every child u_i of v in I is the same as the value set of u_i in J. Hence, due to the definition of the value set of v, the value set of v in I is the same as the value set of v in J. □

Informally speaking, the problem SimpleEval is given an instance I, to evaluate the value set of the root of the query tree of I in I and output it in the format that we will describe. We will use $RootValueSet(I)$ to denote the value set of the root the root of the query tree of I in I.

Figure 2.1 An example of making a query tree normalized



2.2.2 Input and Output

In this section we specify formats of the input and the output of an algorithm for SimpleEval. We first explain about the input to the algorithm. It will be assumed that the given instance is normalized. This does not reduce the generality of the problem as an instance I that is not normalized can be easily made normalized by repeatedly considering a node v with only one child u and executing the following steps until no internal node with only one child remains:

Normalize(v)**:**

if *v is the root* **then**
delete v and make u the new root.
else
if *u is a leaf* **then**
delete v and make u a child of the parent of v.
else
delete both v and u and make every child of u a child of the parent of v.

Figure 2.1 shows how this method works. In this method, each time a node of the query tree is deleted and the method stops only when there is no internal node of degree one.

So, the query tree becomes normalized, ultimately. We now prove that by doing each step of this method, the value set of the root does not change and hence solving the problem for the instance created newly suffices.

We consider the three cases distinguished in the method one by one. When v has only one child u, the value set of v is the same as the value set of u. Now, if v is the root, we delete v and make u the new root. Thus, the value set of the root does not change. Now suppose v is not the root and so v has a parent w which is a union (intersection) node and $v = v_1, v_2, \ldots,$ and v_n are children of w. Hence, the value set of w is the union (intersection) of the value sets of v, v_2, $\ldots$, and v_n, that is, the union (intersection) of the value sets of u and $v_2, \ldots,$ and v_n. Now, first consider the case in which u is a leaf. Then, after the change explained in the method, w has n children u, v_2, v_3, $\ldots$, and v_n. Thus, the value set of w is still the union (intersection) of value sets of u and v_2, $\ldots$, and v_n. Next, consider the other case in which v is an internal node. Then, since w is a union (an intersection) node, v is an intersection (a union) node and thus u is a union (an intersection). Therefore, the value set of u before the change is the union (an intersection) of value sets of children of u and so the value set of w before the change is the union (intersection) of the value sets of the children of u and the value sets of $v_2, \ldots,$ and v_n. In addition, after executing Normalize(v), the children of w are the children of u and $v_2, \ldots,$ and v_n. So, after the change the value set of w is still the union (intersection) of the value sets of the children of u, and the value sets of $v_2, \ldots,$ and v_n. Consequently, in either case the change applied to the subtree rooted at w does not have any influence on the value set of w and thus the value sets of w and ancestors of w, including the root, do not change.

As explained in Chapter 1, in this thesis our focus is on comparison-based algorithms for the problem. In a comparison-based algorithm, all decisions are based on relative values of objects rather than their actual values. To enforce this limitation, we do not let the algorithm read the values of objects from the input. Instead, the algorithm can read the signature of the instance and can also submit queries regarding relative values of objects. Formally, running an algorithm $\mathcal{A}$ on an instance $I = (Q, \omega, \mu)$, at the beginning $\mathcal{A}$ reads the signature (Q, ω). Also, during its execution, $\mathcal{A}$ can submit a query consisting a pair (o_1, o_2) of objects to an oracle $\mathit{Compare}_I$ and then the response of $\mathit{Compare}_I$ is one of the three operators "$<$", "$==$", or "$>$", depending on which of the statements $\mu(o_1) < \mu(o_2)$,

$\mu(o_1) = \mu(o_2)$, or $\mu(o_1) > \mu(o_2)$ is correct, respectively.

Since the algorithm is not aware of the values of objects, the result of the algorithm is an ordered set of objects rather than an ordered set of values. More formally, the algorithm is required to compute a sequence of objects satisfying the conditions defined in the next definition.

Definition 2.10 *Given an instance $I = (Q, \omega, \mu)$, a* solution *of I is a sequence Σ of objects of I such that the following properties hold:*

1. *The set of values of objects of Σ is the same as $RootValueSet(I)$.*
2. *No two objects of Σ have the same value.*
3. *Objects appear in increasing order of value.*

Before describing the format required for the output of an algorithm, using an example we clarify why we do not design algorithms so that they simply evaluate a solution of a given instance and output it. Consider an instance $I = (Q, \omega, \mu)$ in which Q has three nodes: a union node as the root and two leaves l_1 and l_2. Also, $\omega(l_1) = o_1, o_2, \ldots, o_n$, $\omega(l_2) = p_1, p_2, \ldots, p_m$, and μ is defined such that $\mu(o_1) < \mu(o_2) < \ldots < \mu(o_n) < \mu(p_1) < \mu(p_2) < \ldots < \mu(p_m)$. The algorithm knows in advance that I is an instance and so by definition μ is an ordered value function. Hence, by comparing values of o_n and p_1 the algorithm potentially can know that all objects of $\omega(l_1)$ have values less than values of all objects of $\omega(l_2)$. Thus, using just one comparison an algorithm may know $\omega(l_1) \circ \omega(l_2)$ is a solution of the instance, but in order to print this output, the algorithm will need to consume $\Omega(n + m)$ time just to name the objects. To avoid this kind of waste of time, the format of the output is designed such that given an instance (Q, ω, μ), an algorithm can output a subsequence of $\omega(l)$, for a leaf l of Q, in constant time.

Definition 2.11 *Given an instance $I = (Q, \omega, \mu)$, an* output item for I *is a triple (l, j, k) where l is a leaf of Q and $1 \leq j \leq k \leq \text{length}(\omega(l))$. In addition, given a sequence $\Phi = (l_1, j_1, k_1), \ldots, (l_n, j_n, k_n)$ of output items of I, the* expansion of Φ *is defined as the sequence $\Sigma_1 \circ \Sigma_2 \circ \ldots \circ \Sigma_n$, where for every i, $1 \leq i \leq n$, $\Sigma_i = \omega(l_i)[j_i], \omega(l_i)[j_i + 1], \ldots, \omega(l_i)[k_i]$.*

By the previous definition, if we add an output item (l, j, k) to the end of a sequence Φ of output items, the sequence $\omega(l)[j], \omega(l)[j+1], \ldots, \omega(l)[k]$ is appended to the expansion of Φ. We state this fact as an observation for the future reference.

Observation 2.3 *Given a sequence of output items Φ of an instance I and an output item (l, j, k) of I, the expansion of $\Phi \odot (l, j, k) = \Sigma_\Phi \circ \Sigma_1$ where Σ_Φ is the expansion of Φ and $\Sigma_1 = \omega(l)[j], \omega(l)[j+1], \ldots, \omega(l)[k]$.*

Given an instance $I = (Q, \omega, \mu)$, an algorithm for the problem SimpleEval evaluates a sequence Φ of output items of I such that the expansion of Φ is a solution of I. So, the next observation will hold.

Observation 2.4 *Given a comparison-based algorithm $\mathcal{A}$ for SimpleEval and an instance I, the expansion of the output produced by $\mathcal{A}$ when run on I is a solution to I.*

2.3 Comparisons

2.3.1 Comparison-Based Algorithms

In this section we discuss the behavior of comparison-based algorithms for SimpleEval. Informally speaking, a comparison-based algorithm is one which all decisions are based on the previous responses of $Compare_I$ to the queries of of the algorithm. As explained, in each query two objects o_1 and o_2 are specified by the algorithm and the algorithm is informed of the result of the comparison. This result can be stated in the form of a comparison proposition defined as follows.

Definition 2.12 *A* comparison proposition between objects o and p *is a statement of the form o R p where the operator R is one of "$<$", "$>$", or "$==$".*

Whenever an value function μ is fixed, we can talk about the correctness of a comparison proposition as a total order on members of $\mathbb{V}$ is known to us. More formally, given two objects o and p, depending which of the statements $\mu(o) > \mu(p)$, $\mu(o) < \mu(p)$, or $\mu(o) = \mu(p)$ is correct, we say the comparison propositions $o > p$ and $p < o$, $p > o$ and $o < p$,

or $o == p$ and $p == o$ are *satisfied* by the value function μ or by a disordered instance whose value function is μ. Also, given a set $\mathcal{P}$ of comparison propositions, $\mathcal{P}$ *is satisfied by a value function* μ or *by a disordered instance* I if every comparison proposition of $\mathcal{P}$ is satisfied by μ or by I, respectively.

Definition 2.13 *Given a set $\mathcal{P}$ of comparison propositions, we say an object o is* visited by $\mathcal{P}$ *or is* $\mathcal{P}$-visited *if there exists an operator R and an object p such that $o\,R\,p$ or $p\,R\,o$ is in $\mathcal{P}$; otherwise o is* skipped by $\mathcal{P}$ *or is* $\mathcal{P}$-skipped. *The set of objects visited by $\mathcal{P}$ is denoted by Visited($\mathcal{P}$).*

We can present an alternative for the definition of ordered value functions based on the set of comparison propositions that a value function must satisfy in order to be ordered.

Definition 2.14 *Given a disordered instance $I = (Q, \omega, \mu)$, $\mathcal{K}(I)$ is defined as the set of all comparison propositions of the form $o_1 < o_2$ where o_1 and o_2 are two consecutive objects of $\omega(l)$, for a leaf l of Q.*

Observation 2.5 *A disordered instance I is an instance if and only if I satisfies $\mathcal{K}(I)$.*

When an instance I is fixed, given two objects o_1 and o_2 of I such that $o_2 < o_1 \in \mathcal{K}(I)$, we say o_1 is *bigger than* o_2 and o_2 is *smaller than* o_1. In this way, given an instance I and a leaf l of the query tree of I, we may talk about the biggest object or the smallest object of $\omega(l)$.

Given an instance I as the input, a comparison-based algorithm $\mathcal{A}$ each time submits a query (o, p) to $Compare_I$ and then $\mathcal{A}$ receives the response of $Compare_I$ in the form of one of operators "$>$", "$==$", or "$<$", indicating which of the comparison propositions $o < p$, $o == p$, or $o > p$ is satisfied by I. In this situation if c is the comparison proposition between o and p satisfied by I, we say c *is reported to* $\mathcal{A}$. The sequence of all comparison propositions reported to $\mathcal{A}$ when $\mathcal{A}$ is running on I is denoted by $C_{\mathcal{A}}(I)$.

Observation 2.6 *Given a comparison-based algorithm $\mathcal{A}$ and an instance I, I satisfies every comparison proposition in $C_{\mathcal{A}}(I)$.*

By saying an algorithm $\mathcal{A}$ *visits* (*skips*) *an object o of an instance I* we mean o is visited (skipped) by the set of comparison propositions in $C_{\mathcal{A}}(I)$. We use $\mathit{Visited}(I, \mathcal{A})$ to denote the set of all objects visited by the set of comparison propositions in $C_{\mathcal{A}}(I)$.

Given a comparison-based algorithm $\mathcal{A}$, two instances I and J with the same signature, and a set $\mathcal{S}$ of objects of I, consider a special case in which for every comparison proposition c between any pair of objects of $\mathcal{S}$, c is satisfied by I if and only if c is satisfied by J. If when $\mathcal{A}$ is run on I does not visit any object outside $\mathcal{S}$, informally speaking, $\mathcal{A}$ can not distinguish between I and J and so $\mathcal{A}$ will submits the same sequence of queries and will produce the same output when $\mathcal{A}$ is run on I and when $\mathcal{A}$ is run on J. We will discuss such situations more formally after the next definition.

Definition 2.15 *Given a set $\mathcal{V}$ of objects, two instances I_1 and I_2 are* $\mathcal{S}$-identical *if I_1 and I_2 have the same signature and for every comparison proposition c between two objects of $\mathcal{S}$, I_1 satisfies c if and only if I_2 satisfies c.*

Now we investigate the behavior of a comparison-based algorithm $\mathcal{A}$ when running on instances that are similar in terms of relative values of objects visited by $\mathcal{A}$. Suppose a deterministic algorithm $\mathcal{A}$ is given an instance I as the input. By definition, the result of any decision that $\mathcal{A}$ makes at any point during its execution is only determined by what $\mathcal{A}$ has read from the input (that is, the signature of I) and the sequence of operators that $\mathcal{A}$ has received as responses to its queries from $\mathit{Compare}_I$. Using this fact we can prove the next lemma which discusses the set of decisions made by $\mathcal{A}$.

Lemma 2.7 *Consider a comparison-based algorithm $\mathcal{A}$ for SimpleEval and two instances I and J with the same signature and a non-negative integer i such that lengths of both $C_{\mathcal{A}}(I)$ and $C_{\mathcal{A}}(J)$ are at least i. Moreover, suppose the operators of the first i comparison propositions in $C_{\mathcal{A}}(I)$ are the same as the operators of the first i comparison propositions in $C_{\mathcal{A}}(J)$.*

Claim 1. *The length of $C_{\mathcal{A}}(I)$ is at least $i+1$ if and only if the length of $C_{\mathcal{A}}(J)$ is at least $i+1$.*

Claim 2. *If the length of both $C_{\mathcal{A}}(I)$ is less $i+1$ then the outputs generated by $\mathcal{A}$ produces the same outputs for I and for J; otherwise the first (the second) operands of the $(i+1)$th comparison propositions in $C_{\mathcal{A}}(I)$ and in $C_{\mathcal{A}}(J)$ are the same.*

Proof. We first discuss the first claim and then the second claim. After i comparison propositions have been reported to $\mathcal{A}$, $\mathcal{A}$ might either decide to produce a sequence of output items and stop running or decide to submit a further pair of objects as a query to $\mathit{Compare}_I$. As explained before lemma, this decision is based on the signature of the instance and the sequence of the first i operators received form $\mathit{Compare}_I$. Hence, whether or not $C_{\mathcal{A}}(I)$ has at least $i+1$ comparison propositions, depends only on the signature of I and the operators of the first i comparison propositions in $C_{\mathcal{A}}(I)$. So, Claim 1 is true.

Considering again the situation described by lemma, this time we discuss the situation after $\mathcal{A}$ made its decision on whether to submit a new query or to produce an output and finish its execution. Then, depending on whether the algorithm has decided to submit the $(i+1)$st query or to produce the output, the algorithm has to decide on the the new query that it submits or on the output that it produces, respectively. In either case, this new decision is again based only on what the algorithm has read from the input and has received from $\mathit{Compare}_I$ so far, that is, on the signature of I and the sequence of operators of the first i comparison propositions in $C_{\mathcal{A}}(I)$. By definition, deciding on the the $(i+1)$st query to be submitted to $\mathit{Compare}_I$ is equivalent to deciding on the operands of the $(i+1)$th comparison proposition in $C_{\mathcal{A}}(I)$. Hence, Claim 2 is also true. □

In this section we defined comparison propositions and we explored properties of single comparison propositions reported to an algorithm. In the next section we will study the properties of the whole sequence $C_{\mathcal{A}}(I)$, for an instance I and an algorithm $\mathcal{A}$.

2.3.2 Proofs

In this section we study the set of evidence that a comparison-based algorithm $\mathcal{A}$ should collect to make sure that the expansion of the output that $\mathcal{A}$ generates is a solution to the given instance.

Definition 2.16 *Consider a signature $G = (Q, \omega)$, a set $\mathcal{P}$ of comparison propositions between objects of G, and a sequence Σ of objects of G. We say Σ is a* solution certified by $\mathcal{P}$ *if Σ is a solution for every instance I with signature G such that I satisfies $\mathcal{P}$.*

Definition 2.17 *Given a signature $G = (Q, \omega)$, a* proof for G *is a set $\mathcal{P}$ of comparison propositions such that there exists a solution certified by $\mathcal{P}$. In addition, a* proof for an instance I *is a proof $\mathcal{P}$ of the signature of I that is satisfied by I.*

Given a set $\mathcal{P}$ of comparison propositions, if two instances I and J are *Visited*($\mathcal{P}$)-identical, it follows by Definition 2.15 that $\mathcal{P}$ is satisfied by I if and only if $\mathcal{P}$ is satisfied by J. This yields the following observation.

Observation 2.8 *Consider a proof $\mathcal{P}$ of an instance I and suppose J is an instance Visited($\mathcal{P}$)-identical with I. Then $\mathcal{P}$ is a proof for J.*

The next lemma will be used to construct a proof for a given instance.

Lemma 2.9 *Consider an instance $I = (Q, \omega, \mu)$, a sequence Σ of objects of I, and a set $\mathcal{V}$ of objects of I such that Σ is a solution for every instance that is $\mathcal{V}$-identical with I. The set $\mathcal{P}$ of all comparison propositions between objects of $\mathcal{V}$ satisfied by I is a proof for I.*

Proof. We prove that Σ is a solution certified by $\mathcal{P}$ and this yields the fact that $\mathcal{P}$ is a proof for I, by Definition 2.17. To show that Σ is certified by $\mathcal{P}$, by Definition 2.16 it suffices to prove that, given an instance $J = (Q, \omega, \nu)$ satisfying $\mathcal{P}$, Σ is a solution to J. To prove this fact, we prove that J is $\mathcal{V}$-identical with I and then it follows from the assumption of the lemma that Σ is a solution to J. So, we consider a comparison proposition $c = o\,R\,p$ between objects in $\mathcal{V}$ and we prove that c is satisfied by I if and only if c is satisfied by J.

We now assume to the contrary that c is satisfied by only one of I and J and we obtain a contradiction. Depending on whether $\mu(o) > \mu(p)$, $\mu(o) = \mu(p)$, or $\mu(o) < \mu(p)$, by definition one of the comparison propositions $o > p$, $o == p$, or $o < p$ is satisfied by I. So, there is a comparison proposition d between $\mu(o)$ and $\mu(p)$ satisfied by I. Hence, by the definition of $\mathcal{P}$, d is in $\mathcal{P}$ and so J satisfies d. Therefore, d is satisfied by both I and J while c is satisfied by exactly one of them and thus $c \neq d$. So, one of the instances I and J, say I, satisfies the two distinct comparison propositions c and d between o and p, that is, two of the statements $\mu(o) < \mu(p)$, $\mu(o) = \mu(p)$, and $\mu(o) > \mu(p)$ are true, a contradiction. So, the assumption that c is satisfied by only one of I and J is false and hence the lemma is correct. □

As explained in the beginning of the section, informally speaking, a proof of an instance I is a set of evidence that shows a sequence is a solution to I. We now prove in the next lemma that the set of comparison propositions in $C_{\mathcal{A}}(I)$ is actually such a set of evidence.

Lemma 2.10 *Given a comparison-based algorithm $\mathcal{A}$ and an instance I, the set $\mathcal{P}$ of all comparison propositions in $C_{\mathcal{A}}(I)$ is a proof for I.*

Proof. We first prove that, given an instance J with the same signature as I satisfying $\mathcal{P}$, the expansion Σ of the sequence Φ of the output items produced when running $\mathcal{A}$ on I is a solution to J. Then we can show that Σ is certified by $\mathcal{P}$ and hence $\mathcal{P}$ is a proof for J. To prove that Σ is a solution to J, we show that the length n of $C_{\mathcal{A}}(J)$ is at least as large as the length m of $C_{\mathcal{A}}(I)$ and that the operators of the first m comparison propositions in $C_{\mathcal{A}}(I)$ are the same as the operators of the first m comparison propositions in $C_{\mathcal{A}}(J)$, respectively, and then we will use Lemma 2.7.

To prove the above property for $C_{\mathcal{A}}(J)$, we use induction on i, $0 \leq i \leq m$, to prove that the length of $C_{\mathcal{A}}(J)$ is at least i and the operators of the first i comparison propositions in $C_{\mathcal{A}}(I)$ are the same as the operators of the first m comparison propositions in $C_{\mathcal{A}}(J)$. The base case $i = 0$ is trivial. So, we suppose the claim holds for i, $0 \leq i < m$, and we prove it for $i + 1$. Since I and J have the same signature and by induction sequences of operators of the first i comparison propositions in I and in J are equal, we can apply Lemma 2.7 to conclude that since $i + 1 \leq m$, $i + 1 \leq n$. Then, it follows from Lemma 2.7 that the first (the second) operands of the $(i + 1)$st comparison propositions in $C_{\mathcal{A}}(I)$ and $C_{\mathcal{A}}(J)$ are the same. Now suppose $o\,R\,p$ is the $(i + 1)$st comparison proposition in $C_{\mathcal{A}}(I)$. Then, as we proved, o and p are the operands of the $(i + 1)$st comparison propositions in $C_{\mathcal{A}}(J)$ and so (o, p) is the $(i + 1)$st query submitted to $\mathit{Compare}_J$. Also, since $o\,R\,p$ is in $C_{\mathcal{A}}(I)$, by definition $o\,R\,p$ is in $\mathcal{P}$ and thus as $\mathcal{P}$ is satisfied by J, $o\,R\,p$ is satisfied by J. Therefore, while running $\mathcal{A}$ on J, after $\mathcal{A}$ submits (o, p) as its $(i+1)$st query to $\mathit{Compare}_J$, the response of $\mathit{Compare}_J$ is the operator R. As a result, the operators of the $(i + 1)$st comparison propositions in $C_{\mathcal{A}}(I)$ and in $C_{\mathcal{A}}(J)$ are the same, as well. Hence, the claim is true for $i + 1$.

Now we use the claim for $i = m$ to prove that $\mathcal{A}$ produces the same outputs for I and J and then to complete the proof. According to the claim for $i = m$, the length of $C_{\mathcal{A}}(J)$ is

at least m and the operators of the first m comparison propositions in $C_{\mathcal{A}}(I)$ and in $C_{\mathcal{A}}(J)$ are the same. So, since the length m of $C_{\mathcal{A}}(I)$ is not greater than m, by Lemma 2.7 for $i = m$, the output Φ of $\mathcal{A}$ on I is the same as the output of $\mathcal{A}$ on J. Therefore, as Σ is the expansion of Φ, by Observation 2.4 Σ is a solution to J. Consequently, since J was selected as an arbitrary instance with the same signature as I satisfying $\mathcal{P}$, by Definition 2.16 Σ is certified by $\mathcal{P}$ and hence by Definition 2.17 $\mathcal{P}$ is a proof for the signature of I. Moreover, since by assumption every comparison proposition in $\mathcal{P}$ is in $C_{\mathcal{A}}(I)$, by Observation 2.6 I satisfies $\mathcal{P}$ and thus by Definition 2.17 $\mathcal{P}$ is a proof for I. □

2.4 Tools

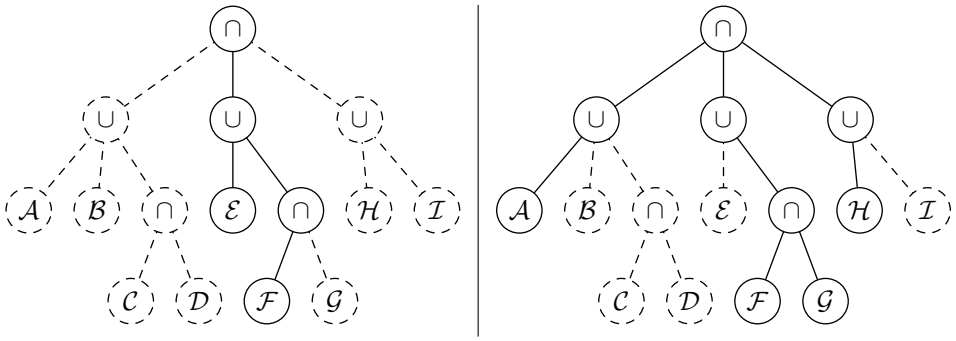
In this part of the thesis we introduce some concepts to facilitate dealing with query trees, proofs, and values throughout the thesis. We first in Section 2.4.1 explore properties of specific subtrees of the query tree. Afterward in Section 2.4.2, given a proof $\mathcal{P}$ of an instance I, we introduce a number of functions for creating new instances satisfying $\mathcal{P}$ by making minor changes to the values of objects in I. These functions will help us in investigating properties of proofs. Finally, we talk more about our universal set $\mathbb{V}$ of values in Section 2.4.3.

2.4.1 The Query Tree

In this section we identify rooted subtrees T of the query tree with this property that knowing that a value a is (or is not) in the value set of every leaf of T, it is concluded that a is (or is not) in the value set of the root. These subtrees will be used for designing the algorithm and also for proving the lower bound.

We now define the subtrees that we talked above. Informally speaking, the query tree Q represents a set expression E as mentioned before. Thus, a complete subtree T of Q represents a subexpression of E. This subexpression can be written as the intersection (union) of a number of union (intersection) terms. The set of nodes of the subtree T representing each of these terms create a sub-union tree (a sub-intersection tree) of T as defined in below.

Figure 2.2 An example of a sub-union tree and a sub-intersection tree



Definition 2.18 *Given a query tree Q and a complete subtree U of Q, a* sub-union tree *(a* sub-intersection tree*)* of U *is a subtree T of U with the following properties:*

1. *It contains the root of U.*
2. *If it contains a union (an intersection) node, it contains all of its children.*
3. *If it contains an intersection (a union) node, it contains exactly one of its children.*

Figure 2.2 shows an example.

Next we define the following notation for decomposing a given sub-union tree (sub-intersection tree) to a number of sub-union trees (sub-intersection trees) with smaller heights and also for composing a number of sub-union trees (sub-intersection trees) to create a new sub-union tree (sub-intersection tree) containing all of the original ones. Consider a query tree Q. If T is a sub-union tree (a sub-intersection tree) of $Q[v]$, for a node v of Q, depending on the type of v, deleting v from T either changes T to one smaller sub-union tree (sub-intersection tree) or splits it to k sub-union trees (sub-intersection trees) where k is the number of children of v. In the first case we denote the new sub-union tree (sub-intersection tree) by $\top_{\ominus}(T)$ and in the second case we denote the sub-union tree (sub-intersection tree) created in this way and rooted at a child u of v by $\top_{\ominus}(T, u)$. Also, if $U = \top_{\ominus}(T)$, for some T and U, we may write $T = \top_{\oplus}(U)$. Similarly, if v is a node with k children $u_1, \ldots, u_k$ and for every i, $1 \leq i \leq k$, we have $T_i = \top_{\ominus}(T, u_i)$, for a sub-union

tree or a sub-intersection tree T, we write $T = \top_{\oplus}(T_1, T_2, \ldots, T_k)$. The following four observations follow by the definition of the function $\top_{\oplus}$ and $\top_{\ominus}$, immediately.

Observation 2.11 *Given an intersection (a union) node v and a sub-union (a sub-intersection) tree T of $Q[v]$, $T = \top_{\oplus}(\top_{\ominus}(T))$.*

Observation 2.12 *Given a child u of an intersection (a union) node v and a sub-union (a sub-intersection) tree T of $Q[u]$, $T = \top_{\ominus}(\top_{\oplus}(T))$.*

Observation 2.13 *Given a union (an intersection) node v with children $u_1, \ldots, u_k$ and a sub-union (a sub-intersection) tree T of $Q[v]$, $T = \top_{\oplus}(\top_{\ominus}(T, u_1), \ldots, \top_{\ominus}(T, u_k))$.*

Observation 2.14 *Given sub-union trees (sub-intersection trees) $T_1, T_2, \ldots, T_k$, and U such that $U = \top_{\oplus}(T_1, \ldots, T_k)$, $T_i = \top_{\ominus}(U, u_i)$ where u_i is the root of T_i, for every i, $1 \leq i \leq k$.*

When T and U are such that $T = \top_{\oplus}(U)$, or equivalently $U = \top_{\ominus}(T)$, by definition the root v of T is a union or an intersection node and U is obtained from T by just removing the node v. So, as v is not a leaf, the next observation is true.

Observation 2.15 *Given two sub-union trees or sub-intersection trees U and T such that $U = \top_{\ominus}(T)$, leaves(T) = leaves(U). Also, nodes(T) = nodes$(U) \cup \{v\}$ and nodes(U) = nodes$(T) - \{v\}$ where v is the root of T.*

Similarly, if $T = \top_{\oplus}(U_1, \ldots, U_k)$, for some $U_1, \ldots, U_k$, and T, roots of all subtrees $U_1, \ldots,$ and U_k are children of an internal node v and except v, which is an internal node, every node of T is in one of $U_1, \ldots,$ or U_k. Also, all nodes of each of U_i's, $1 \leq i \leq k$ appear in T. Hence, as v is not a leaf, the next observation is true.

Observation 2.16 *Given sub-union trees or sub-intersection trees T and $U_1, \ldots, U_k$ such that $T = \top_{\oplus}(U_1, \ldots, U_k)$, leaves$(T) = \bigcup_{i=1}^{k}$ leaves(U_i).*

Now, given a union (an intersection) node v with children $u_1, \ldots, u_k$ and a sub-union (a sub-intersection) tree T of $Q[v]$, by Observation 2.13 $T = \top_{\oplus}(\top_{\ominus}(T, u_1), \ldots, \top_{\ominus}(T, u_k))$ and hence by Observation 2.16 leaves$(T) = \bigcup_{i=1}^{k}$ leaves$(\top_{\ominus}(T, u_1))$. Thus, the next observation is also true.

Observation 2.17 *Given a child u of a union (an intersection) node v and a sub-union tree (sub-intersection tree) T of $Q[v]$, $leaves(\top_{\ominus}(T, u)) \subseteq leaves(T)$.*

In the next two lemmas we investigate properties of intersections of sets of nodes or leaves of sub-union trees and sub-intersection trees of a given subtree. We first prove that, given a node v of a query tree Q, the set of nodes of a sub-intersection tree (sub-union tree) of $Q[v]$ can not be a proper subset of the set of nodes of another sub-intersection tree (sub-union tree) of $Q[v]$. Then in Lemma 2.18 we discuss the intersection of the sets of leaves of a sub-intersection and a sub-union tree of $Q[v]$.

Lemma 2.18 *Consider a query tree Q and suppose T and U are two sub-union trees or two sub-intersection trees of $Q[v]$, for a node v of Q. If $nodes(T) \subseteq nodes(U)$ then $T = U$.*

Proof. We prove the lemma by induction on the height of v. If v is a leaf, the problem is trivial. Otherwise, suppose v is a union node; the case of intersection nodes is similar: it suffices to exchange the terms "union" and "intersection" in the proof. If T and U are sub-intersection trees, we consider the roots u_1 and u_2 of $T' = \top_{\ominus}(T)$ and $U' = \top_{\ominus}(U)$. If $u_1 \neq u_2$, then $u_1 \in nodes(T)$ and $u_1 \notin nodes(U)$ while $nodes(T) \subseteq nodes(U)$, a contradiction. Thus, $u_1 = u_2$. So, T' and U' are sub-intersection trees of $Q[u_1]$. Moreover, applying Observation 2.15 for T' and T, we may conclude that $nodes(T') = nodes(T) - \{v\}$. Likewise, applying Observation 2.15 for U' and U, we may conclude that $nodes(U') = nodes(U) - \{v\}$. Furthermore, as by assumption $nodes(T) \subseteq nodes(U)$, $nodes(T) - \{v\} \subseteq nodes(U) - \{v\}$. Therefore, $nodes(T') = nodes(T) - \{v\} \subseteq nodes(U) - \{v\} = nodes(U')$. Hence, by induction $T' = U'$ and so $T = \top_{\oplus}(T') = \top_{\oplus}(U') = U$.

Now, we consider the other case, that is, T and U are sub-union trees. Then, suppose $u_1, \ldots, u_k$ are children of v. For every child u_i of v, $T_i = \top_{\ominus}(T, u_i)$ and $U_i = \top_{\ominus}(U, u_i)$ are two sub-intersection trees of u_i and $nodes(T_i) \subseteq nodes(U_i)$; otherwise, there exists a node v' of $Q[u_i]$ which appears in T_i but not in U_i and so, it appears in T but not in U and this contradicts the assumption that $nodes(T) \subseteq nodes(U)$. So, for every i, $1 \leq i \leq k$, $T_i = U_i$, by induction. Therefore, $T = \top_{\oplus}(T_1, \ldots, T_k) = \top_{\oplus}(U_1, \ldots, U_k) = U$. □

Lemma 2.19 *Given a query tee Q and a complete subtree U of Q, for every sub-union tree T_1 and every sub-intersection tree T_2 of U, $|leaves(T_1) \cap leaves(T_2)| = 1$.*

Proof. We prove the lemma using induction on the height of the root v of U. The proof is clear when v is a leaf. We consider the case in which v is a union node. The case of intersection nodes is similar and it suffices to exchange the terms "intersection" and "union". We suppose u is the root of $T'_2 = \top_{\ominus}(T_2)$ and we consider the sub-union tree $T'_1 = \top_{\ominus}(T_1, u)$ of $Q[u]$. Since the height of u is less than the height of v, $|\mathit{leaves}(T'_1) \cap \mathit{leaves}(T'_2)| = 1$, by induction. But we know $\mathit{leaves}(T'_2) = \mathit{leaves}(T_2)$. So, $\mathit{leaves}(T_1) \cap \mathit{leaves}(T_2) = \mathit{leaves}(T_1) \cap \mathit{leaves}(T'_2)$. Also, $\mathit{leaves}(T_1) \cap \mathit{leaves}(T'_2) = \mathit{leaves}(T'_1) \cap \mathit{leaves}(T'_2)$ because $\mathit{leaves}(T_1) - \mathit{leaves}(T'_1)$ contains no leaf of $Q[u]$ and so $(\mathit{leaves}(T_1) - \mathit{leaves}(T'_1)) \cap \mathit{leaves}(T'_2) = \emptyset$. Therefore, $|\mathit{leaves}(T_1) \cap \mathit{leaves}(T_2)| = |\mathit{leaves}(T'_1) \cap \mathit{leaves}(T'_2)| = 1$. □

Given a node v of a query tree Q and a subset $\mathcal{S}$ of leaves of Q, in the next two lemmas we prove that either one can find a sub-union tree (sub-intersection tree) of $Q[v]$ whose leaves are all in $\mathcal{S}$ or one can find a sub-intersection tree (sub-union tree) of $Q[v]$ whose leaves are all in $\mathit{leaves}(Q[v]) - \mathcal{S}$. Later, throughout the thesis, by defining $\mathcal{S}$ as the set of all leaves satisfying a property $\wp$, using these lemmas we prove that one can find a sub-union tree or a sub-intersection tree T such that either all leaves of T satisfy $\wp$ or no leaf of T satisfies $\wp$.

Lemma 2.20 *Consider a query tree Q and a node v of Q. Given a subset $\mathcal{S}$ of the set of leaves of Q, if for every sub-union tree U of $Q[v]$ we have $\mathcal{S} \cap \mathit{leaves}(U) \neq \emptyset$, then there is a sub-intersection tree T of $Q[v]$ such that $\mathit{leaves}(T) \subseteq \mathcal{S}$.*

Proof. We use induction on the height v to prove the lemma. If v is a leaf, the proof is trivial. So suppose v is an internal node with k children $u_1, u_2, \ldots, u_k$.

First, we consider the case in which v is a union node. First we prove that there exists a child u_j of v such that for every sub-union tree U of $Q[u_j]$, $\mathcal{S} \cap \mathit{leaves}(U) \neq \emptyset$ and then using this fact we prove the lemma for v. Assume to the contrary that there is no such child u_j. Then, there exists a sub-union tree U_i of the subtree $Q[u_i]$ such that $\mathcal{S} \cap \mathit{leaves}(U_i) = \emptyset$, for every i, $1 \leq i \leq k$. Consider the sub-union tree $U = \top_{\oplus}(U_1, \ldots, U_k)$ of $Q[v]$. Then, as by Observation 2.16 every leaf of U is a leaf of one of U_i's, no leaf of U is in the set $\mathcal{S}$, contradicting the assumption of the lemma. Hence, there is a child u_j of v such that for every sub-union tree U of $Q[u_j]$, $\mathcal{S} \cap \mathit{leaves}(U) \neq \emptyset$. The height of u_j is less than the height

of v. So, by induction $Q[u_j]$ has a sub-intersection tree T' such that $\mathit{leaves}(T') \subseteq \mathcal{S}$. Now, $T = \top_{\oplus}(T')$ is a sub-intersection tree of $Q[v]$ and since $\mathit{leaves}(T) = \mathit{leaves}(T')$, $\mathit{leaves}(T)$ is a subset of $\mathcal{S}$.

Next we consider the other case, in which v is an intersection node. We first show that, for every child u_i of v, $Q[u_i]$ has a sub-intersection tree whose leaves are all in $\mathcal{S}$. Then we compose these sub-intersection trees using the operator $\top_{\oplus}$ and we prove the lemma for v. Given a sub-union tree U of $Q[u_i]$, for a child u_i of v, for the sub-union tree $U' = \top_{\oplus}(U)$ of $Q[v]$ we have $\mathit{leaves}(U') = \mathit{leaves}(U)$ and $\mathit{leaves}(U') \cap \mathcal{S} \neq \emptyset$. Thus, for every child u_i of v and for every sub-union tree U of $Q[u_i]$, $\mathit{leaves}(U) \cap \mathcal{S} \neq \emptyset$. So, by induction, for every child u_i, there exists a sub-intersection tree T_i of the subtree $Q[u_i]$ such that $\mathit{leaves}(T_i) \subseteq \mathcal{S}$. We now consider the sub-intersection tree $T = \top_{\oplus}(T_1, \ldots, T_k)$. Then we have $\mathit{leaves}(T) \subseteq \mathcal{S}$ as $\mathit{leaves}(T) = \bigcup_{i=1}^{k} \mathit{leaves}(T_i)$ and $\mathit{leaves}(T_i) \subseteq \mathcal{S}$, for every i, $1 \leq i \leq k$. □

The following lemma can be proved in a similar fashion, by exchanging the terms "union" and "intersection" in the proof.

Lemma 2.21 *Consider a query tree Q and a node v of Q. Given a subset $\mathcal{S}$ of the set of leaves of $Q[v]$, if for every sub-intersection tree U of $Q[v]$ we have $\mathcal{S} \cap \mathit{leaves}(U) \neq \emptyset$, there is a sub-union tree T of $Q[v]$ such that $\mathit{leaves}(T) \subseteq \mathcal{S}$.* □

Given a node v of a query tree Q, recall from beginning of Section 2.4.1 that, informally speaking, $Q[v]$ represents a set expression E which can be written as the union (intersection) of a number of intersection (union) terms and a sub-intersection tree (sub-union tree) represents one of these intersection (union) terms. Hence, as E is the union (intersection) of the aforementioned terms, a value b is in the result of E if b is in the result of one of (each of) these terms, that is, for one (for every) sub-intersection tree (sub-union tree) T of $Q[v]$, b is in the result of the term represented by T. We prove this fact formally in the next lemma.

Lemma 2.22 *Consider an instance (Q, ω, μ), a complete subtree U rooted at a node v of Q, and a value b in $\mathbb{V}$. The following propositions are equivalent:*

P_1: The value b is in the value set of v.

P_2: There exists a sub-intersection tree T of U such that each leaf of T contains an object of value b.

P_3: For every sub-union tree T of U there exists a leaf of T that contains an object of value b.

Proof. First we show P_2 implies P_1 using induction on the height of v. So, we suppose P_2 is true. If v is a leaf, P_1 follows immediately from P_2. If v is a union node, the height of the root u of $\top_{\ominus}(T)$ is less than the height of v; so b is in the value set of u, by induction. Therefore, b is in the value set of v as u is a child of v. Finally, if v is an intersection node, for every child u of v and every leaf l of $\top_{\ominus}(T, u)$, by Observation 2.17 l is a leaf of T and thus $\omega(l)$ contains an object of value b. Consequently, by induction b is in the value set of u as the height of u is less than that of v. So, b is in the value set of v as it is in the value set of every child of v.

Next we suppose P_3 does not hold and we prove P_1 also does not hold. If P_3 does not hold, there exists a sub-union tree T' of U such that for every leaf l of T', $\omega(l)$ does not contain an object of value b. Again, using induction on the height of nodes v of T' we prove that b is not in the value set of v. If v is a leaf it is obvious. If v is an intersection node, the height of the root u of $\top_{\ominus}(T')$ is less than the height of v. Thus b is not in the value set of u, by induction. So, b is not in the value set of v either as u is a child of v. Finally, if v is a union node, for every child u of v, no leaf of $\top_{\ominus}(T', u)$ contains an object of value b. Therefore, by induction b is not in the value set of u as the height of u is less than that of v. So, b is not in the value set of v as it is not in the value set of any child of v.

Finally, we show P_2 holds providing P_3 holds. For this purpose, we define $\mathcal{S}$ to be the set of all leaves u of U such that b is in the value set of u. According to P_3, for every sub-union tree T of U we have $\mathcal{S} \cap \mathit{leaves}(T) \neq \emptyset$. So, according to Lemma 2.20 there exists a sub-intersection tree T' of U such that $\mathit{leaves}(T') \subseteq \mathcal{S}$, that is every leaf of T' contains an object of value b. Hence, P_2 holds. □

When we talk about *union problems* or *union instances*, actually we are considering the cases in which the root of the query tree is a union node. Similarly, whenever we talk

about an *intersection problem* or an *intersection instance*, we are supposing that the root of the query tree is an intersection node.

2.4.2 Numbering Functions

Throughout the thesis we will need to execute various operations on values in $\mathbb{V}$. Selecting one or more values between two specific values or assigning the kth biggest value to an object are two of these operations. In this section we introduce functions, each one creating a one-to-one order-preserving mapping between a set of integers and a set of values. Using these functions, performing such operations will be easier.

Definition 2.19 *A* numbering function *is an one-to-one function* $\phi : \mathbb{N}_0 \mapsto \mathbb{V}$*, where* $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$*, such that for any integers* i *and* j *such that* $i < j$*, we have* $\phi(i) < \phi(j)$*. We define* $\mathbb{V}_\phi = \{\phi(i) \mid i \in \mathbb{N}_0\}$*.*

Given a numbering function ϕ and a value a, we say ϕ *defines* a if a is in $\mathbb{V}_\phi$. In addition, ϕ *defines a value function* μ if for every object o for which $\mu(o)$ is defined, $\mu(o)$ is in $\mathbb{V}_\phi$.

Lemma 2.23 *Consider a numbering function* ϕ *and two values* a *and* b *defined by* ϕ*. If* $a < b$ *then* $\phi^{-1}(a) < \phi^{-1}(b)$*.*

Proof. If $\phi^{-1}(a) = \phi^{-1}(b)$ then $a = \phi(\phi^{-1}(a)) = \phi(\phi^{-1}(b)) = b$. Thus, $\phi^{-1}(a) \neq \phi^{-1}(b)$ since $a \neq b$. Also, if $\phi^{-1}(a) > \phi^{-1}(b)$ then $a = \phi(\phi^{-1}(a)) > \phi(\phi^{-1}(b)) = b$, by Definition 2.19. Hence $\phi^{-1}(a) \not> \phi^{-1}(b)$ as $a \not> b$. So, $\phi^{-1}(a) < \phi^{-1}(b)$. □

Observation 2.24 *For every finite subset* $\mathcal{S}$ *of* $\mathbb{V}$*, there exists a numbering function defining all members of* $\mathcal{S}$*. Furthermore, for every value function* μ *defined over a finite number of objects, there is a numbering function defining* μ*.*

Using the mapping defined by a numbering function ϕ, we can apply an integer operation on a value b by first mapping the value b to the integer $\phi^{-1}(b)$, executing the operation on $\phi^{-1}(b)$ to obtain a non-negative integer k, and then evaluating $\phi(k)$.

Definition 2.20 *Given a value* b*, an integer* i*, a numbering function* ϕ *defining* b*, and an integer operator* $\circledast \in \{+, -, \times\}$ *such that* $\phi^{-1}(b) \circledast i$ *is a non-negative integer, we define* $b \circledast_\phi i = \phi\left(\phi^{-1}(b) \circledast i\right)$*.*

The following definition allows us to map the set of values assigned by a value function μ to a different set of values, preserving the order of values. This will be useful when we need to select a value between values of two objects and we are not sure if such a value exists. In this situation we can map the set of values of all objects to a different set of values such that there exists a value between every two values assigned to objects.

Definition 2.21 *Consider an instance $I = (Q, \omega, \mu)$, a numbering function ϕ defining μ, an integer i, and an integer operator $\circledast \in \{+, -, \times\}$ such that for every object o of I, $\mu(o) \circledast_\phi i$ is defined. We define the value function $\mu \circledast_\phi i$ as $(\mu \circledast_\phi i)(o) = \mu(o) \circledast_\phi i$.*

Considering values a and b defined by the value function ϕ such that $a \circledast_\phi i$ and $b \circledast_\phi i$ are defined, if $a < b$, by Lemma 2.23 $\phi^{-1}(a) < \phi^{-1}(b)$ and so, $\phi^{-1}(a) \circledast i < \phi^{-1}(b) \circledast i$. Hence, $a \circledast_\phi i = \phi(\phi^{-1}(a) \circledast i) < \phi(\phi^{-1}(b) \circledast i) = b \circledast_\phi i$. So given a value function μ defined by a numbering function ϕ, an integer i, and an integer operator $\circledast \in \{+, -, \times\}$ such that $\mu \circledast_\phi i$ is defined, every comparison proposition satisfied by μ is also satisfied by $\mu \circledast_\phi i$. This yields the following observation.

Observation 2.25 *Given an instance $I = (Q, \omega, \mu)$, a numbering function ϕ defining μ, an integer i, and an integer operator $\circledast \in \{+, -, \times\}$ such that $\mu \circledast_\phi i$ is defined, $(Q, \omega, \mu \circledast_\phi i)$ is an instance and is $Objects(I)$-identical with I.*

The next lemma is in fact an application of Definition 2.21. Let us we give intuition on the result proved in this lemma and how this result can be used. Consider an instance $I = (Q, \omega, \mu)$, a numbering function ϕ defining μ, the value function $\nu = \mu \times_\phi 2$, and the disordered instance $J = (Q, \omega, \nu)$. Due to Observation 2.25, J is an instance $Objects(I)$-identical with I. So, given a proof $\mathcal{P}$ of I, since $Visited(\mathcal{P}) \subseteq Objects(I)$, it follows from Definition 2.15 that I and J are $Visited(\mathcal{P})$-identical and hence by Observation 2.8 $\mathcal{P}$ is a proof of J. But J has the additional property which is if we change the value of any object o of J skipped by $\mathcal{P}$ to $\nu(o) +_\phi 1 = (\mu(o) \times_\phi 2) +_\phi 1$, the resulting value function is valid and still satisfies $\mathcal{P}$, as we show in the next proof. This property will be used to prove that, informally speaking, an algorithm or a proof can not skip too many objects of a given instance I; otherwise, as we will prove formally in Section 4.2, we can adjust values of skipped objects such that the new instance still satisfies $\mathcal{P}$ but has no solution in common with I, contradicting the fact that $\mathcal{P}$ has a certified solution.

Before presenting the lemma, we first introduce the following notation for changing the value of a number of objects to a given value. Given a value function μ, objects o_1, ..., and o_k, and a value a, we define the value function $\mu[o_1, \ldots, o_k \rightarrowtail a]$ as follows:

$$\mu[o_1, \ldots, o_k \rightarrowtail a](p) = \begin{cases} a & \text{if } p \in \{o_1, \ldots, o_k\} \\ \mu(p) & \text{otherwise.} \end{cases}$$

Lemma 2.26 *Consider an instance $I = (Q, \omega, \mu)$ and a numbering function ϕ such that ϕ defines μ. Furthermore, suppose $\mathcal{P}$ is a set of comparison propositions satisfied by μ and $\mathcal{S} = \{o_1, \ldots, o_k\}$ is a set of objects of I all skipped by $\mathcal{P}$ and with the same value a. If $\nu = \mu \times_\phi 2$ and $\xi = \nu[o_1, \ldots, o_k \rightarrowtail (a \times_\phi 2) +_\phi 1]$ then $J = (Q, \omega, \xi)$ is an instance satisfying $\mathcal{P}$.*

Proof. We first prove that every comparison proposition in $\mathcal{P} \cup \mathcal{K}(I)$ is satisfied by μ. Then, we prove an inequality on ξ-values of objects and using that inequality we show that every comparison proposition with one of the two operators "$<$" or "$>$" satisfied by μ is satisfied by ξ. Afterward, we prove that every comparison proposition in $\mathcal{P} \cup \mathcal{K}(I)$ with the operator "$==$" is satisfied by ξ. Having proved the above claims, it is proved that ξ satisfies $\mathcal{P} \cup \mathcal{K}(I)$. Also, as I and J have the same signature, it follows from the definition of the function $\mathcal{K}$ that $\mathcal{K}(I) = \mathcal{K}(J)$. So, we then can conclude that ξ satisfies $\mathcal{P} \cup \mathcal{K}(J)$. By Observation 2.5 this result yields the fact that J is an instance satisfying $\mathcal{P}$. So, in this way the lemma is proved.

The proof of the claim that I satisfies $\mathcal{P} \cup \mathcal{K}(I)$ is simple. Since I is an instance, by Observation 2.5 I satisfies $\mathcal{K}(I)$ and since $\mathcal{P}$ is a proof for I, I satisfies $\mathcal{P}$. Hence, I satisfies every comparison proposition in $\mathcal{K}(I) \cup \mathcal{P}$.

Let us now we prove the inequality on $\xi(p)$ that we talked about. Since $\nu = \mu \times_\phi 2$, by Definitions 2.21 and 2.20, for every object p, $\nu(p) = \mu(p) \times_\phi 2 = \phi(2\phi^{-1}(\mu(p)))$. In addition, by definition $a \times_\phi 2 = \phi(\phi^{-1}(a) \times 2)$ and hence $\phi^{-1}(a \times_\phi 2) = 2\phi^{-1}(a)$. Consequently, $(a \times_\phi 2) +_\phi 1 = \phi(\phi^{-1}(a \times_\phi 2) + 1) = \phi(2\phi^{-1}(a) + 1)$. Now, by the definition of ξ, for every object p, if $p \notin \mathcal{S}$ then $\xi(p) = \nu(p) = \phi(2\phi^{-1}(\mu(p)))$; otherwise $\xi(p) = (a \times_\phi 2) +_\phi 1 = \phi(2\phi^{-1}(a) + 1)$. Also, the μ-value of every object in $\mathcal{S}$ is a. Therefore, for every object p, if $p \notin \mathcal{S}$ then

$$\xi(p) = \phi(2\phi^{-1}(\mu(p))); \tag{2.1}$$

otherwise $\xi(p) = \phi(2\phi^{-1}(\mu(p))+1)$. So, as by Definition 2.19 $\phi(2\phi^{-1}(\mu(p))) < \phi(2\phi^{-1}(\mu(p))+1)$, the following inequality is obtained.

$$\phi(2\phi^{-1}(\mu(p))) \leq \xi(p) \leq \phi(2\phi^{-1}(\mu(p)) + 1). \tag{2.2}$$

Now we consider a comparison proposition c of the form $p < q$ satisfied by μ and we show that c is also satisfied by ξ. Then we show that every comparison proposition of the form $q > p$ satisfied by μ is also satisfied by ξ. Since c is satisfied by μ, $\mu(p) < \mu(q)$. As a result, due to Lemma 2.23, $\phi^{-1}(\mu(p)) < \phi^{-1}(\mu(q))$. So $\phi^{-1}(\mu(p)) + 1 \leq \phi^{-1}(\mu(q))$ since $\phi^{-1}(\mu(p))$ and $\phi^{-1}(\mu(q))$ are integers. Hence, $2\phi^{-1}(\mu(p))+1 < 2\phi^{-1}(\mu(p))+2 \leq 2\phi^{-1}(\mu(q))$ and thus by Definition 2.19 $\phi(2\phi^{-1}(\mu(p)) + 1) < \phi(2\phi^{-1}(\mu(q)))$. So, applying Equation 2.2 for p and q, $\xi(p) \leq \phi(2\phi^{-1}(\mu(p)) + 1) < \phi(2\phi^{-1}(\mu(q))) \leq \xi(q)$. Therefore, c is satisfied by ξ. An argument similar to the above one shows that a comparison proposition of the form $q > p$ satisfied by μ is also satisfied by ξ: it suffices to change the definition of c to $q > p$ and repeat the above argument.

We now prove that ξ satisfies every comparison proposition with the operator "$==$" in $\mathcal{P} \cup \mathcal{K}(I)$. The set $\mathcal{K}(I)$ does not contain any comparison proposition with the operator "$==$". Now consider a comparison proposition of $\mathcal{P}$ of the form $p == q$, for two objects p and q. As members of $\mathcal{S}$ are $\mathcal{P}$-skipped, neither p nor q is in $\mathcal{S}$. Also, since by the assumption of the lemma μ satisfies $\mathcal{P}$, $\mu(p) = \mu(q)$. Now, as p and q are not in $\mathcal{S}$, by Equation 2.1 for p and q, $\xi(p) = \phi(2\phi^{-1}(\mu(p))) = \phi(2\phi^{-1}(\mu(q))) = \xi(q)$. Therefore, $p == q$ is satisfied by ξ. So, the lemma is correct as explained before. □

Definition 2.22 *Given a numbering function ϕ and a value b defined by ϕ, we say b is* ϕ-even *if $\phi^{-1}(b)$ is even; otherwise b is* ϕ-odd. *Also, v is* ϕ-positive *if $\phi^{-1}(b)$ is positive.*

2.4.3 Skirted Sets

In order to have a uniform representation for the set of values greater than a particular value a and the set of values greater than or equal to a, for every value a in $\mathbb{V} - \{-\infty, \infty\}$, we create a new *skirted value* a_+ which is defined to be more than a and less than any

member of $\mathbb{V}$ more than a. In addition, we define

$$\mathbb{V}^+ = \mathbb{V} \cup \bigcup_{a \in \mathbb{V} - \{-\infty, \infty\}} \{a+\}$$

The *main values* of a and $a+$ are defined as a. By saying a is a *main value*, we mean a is the main value of a member of $\mathbb{V}^+$. The next three observations follow immediately from the definition of $\mathbb{V}^+$ and the total order defined on this set.

Observation 2.27 *A member b of $\mathbb{V}^+$ is a main value if and only if b is not a skirted value.*

Observation 2.28 *Given a member a of $\mathbb{V}^+$, $-\infty \leq a \leq \infty$.*

Observation 2.29 *Given two values a and b in $\mathbb{V}$ such that $a > b$, $a > b+$.*

Considering two values a and b in $\mathbb{V}^+$ such that $b < a$, if a is a main value, $b < a = \langle a \rangle$; otherwise, the smallest member of $\mathbb{V}^+$ greater than $\langle a \rangle$ is a and so since by assumption $b < a$, b is not greater than $\langle a \rangle$. So, in either case the next observation is true.

Observation 2.30 *Given two values a and b in $\mathbb{V}^+$ such that $b < a$, $b \leq \langle a \rangle$.*

2.5 Breaking the problem down

In this section we consider the problem in the case in which the root of the query tree of a given instance is a union node and has more than one child that is a leaf. We break down the problem in this case into two subproblems, one of which has been studied earlier and the other one in which the root has at most one leaf child. Then, in the rest of the thesis whenever the root is a union node and has more than a child, we just consider the second subproblem, that is, the subproblem in which the root has at most one leaf child.

Now we explain how the problem is divided. Consider a union instance in which the root of the query tree has n children $u_1, \ldots, u_n$ such that $u_1, \ldots, u_k$ are leaves and the rest of the u_i's are intersection nodes. Furthermore, suppose $\mathcal{S}_i$ denotes the value set of the child u_i, for every i, $1 \leq i \leq n$. An algorithm has to evaluate $\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{S}_i$. We divide the

problem of evaluation of this expression to the problems of evaluation of $\mathcal{T} = \bigcup_{i=1}^{k} \mathcal{S}_i$ and then evaluation of $\mathcal{T} \cup \bigcup_{i=k+1}^{n} \mathcal{S}_i$. The discussion on optimality of this way of evaluation of $\mathcal{S}$ is beyond this thesis. As we will explain later in this section, the problem of evaluating $\mathcal{T}$ has been studied before and so we focus on the problem of evaluating $\mathcal{S}$ when $\mathcal{T}$ is already known.

As explained, we break the problem only when the root of the query tree is a union node and the number of leaves that are children of the root is more than one. We now talk more about this type of leaf.

Definition 2.23 *A leaf v of the query tree Q is a* speedy leaf *if the root of Q is a union node and v is a child of the root of Q.*

Observation 2.31 *A leaf l of a query tree has no intersection ancestor if and only if l is a speedy leaf.*

If a query tree Q contains exactly one speedy leaf, we denote that leaf by $sl(Q)$. We denote the tree created by deleting all speedy leaves from Q by $N(Q)$. If no speedy leaf exists, $N(Q)$ is the same as Q. Also, given the instance $I = (Q, \omega, \mu)$ we define $N(I) = (N(Q), \omega, \mu)$.

Observation 2.32 *Given an instance $I = (Q, \omega, \mu)$, the value set of the root in I equals the union of the value set of the root in $N(I)$ and value sets of all speedy leaves of Q.*

As we prove in Lemma 2.34, a property of non-speedy leaves l that makes them different from speedy leaves is that, given a proof $\mathcal{P}$, an object of $\omega(l)$ skipped by $\mathcal{P}$ may not appear in a solution certified by $\mathcal{P}$. This property holds if every intersection node has at least two children

Definition 2.24 *An object o is* hidden *if there exists a proof $\mathcal{P}$ and a solution Σ certified by $\mathcal{P}$ such that $\mathcal{P}$ skips o and o is in Σ.*

Definition 2.25 *A query tree Q is* intersection-normalized *if every intersection node of Q has at least two children. An instance (Q, ω, μ) is* intersection-normalized *if Q is intersection-normalized.*

Note that as $N(Q)$, for a query tree Q, differs from Q only if the root of Q is a union node and $N(Q)$ is created from Q by only deleting speedy leaves. So, if Q is intersection-normalized, $N(Q)$ is also intersection-normalized.

Observation 2.33 *Given a intersection-normalized query tree Q, $N(Q)$ is intersection-normalized. Also, if I is an intersection-normalized instance, $N(I)$ is also intersection-normalized.*

We now prove the property explained before Definition 2.24.

Lemma 2.34 *Given an intersection-normalized instance $I = (Q, \omega, \mu)$, any hidden object o of I belongs to the sequence associated with a speedy leaf.*

Proof. Given a hidden object o of I, we prove that the leaf l containing o has no intersection ancestor and so by Observation 2.31 l is a speedy leaf. By Definition 2.24 there is a proof $\mathcal{P}$ for I and a solution Σ certified by $\mathcal{P}$ such that o is $\mathcal{P}$-skipped and is in Σ. Also, by Observation 2.24 there is a numbering function ϕ defining μ.

The idea is to create from I a new instance $J = (Q, \omega, \xi)$ satisfying $\mathcal{P}$ by changing only the value of o. Then we show that Σ is a solution to J and we conclude that $\xi(o)$ is in the value set of the root. Also, we show that o is the only object of ξ-value $\xi(o)$ in J. Using these facts we prove that l does not have any intersection ancestor. We now define the instance J as explained in Lemma 2.26: we define $\nu = \mu \times_\phi 2$, $\xi = \nu[o \rightarrowtail (\mu(o) \times_\phi 2) +_\phi 1]$, and as stated above, $J = (Q, \omega, \xi)$. Then, as by assumption o is $\mathcal{P}$-skipped, by Lemma 2.26 for $a = \mu(o)$, J is an instance satisfying $\mathcal{P}$. Hence, Σ is a solution to J, due to Definition 2.16. Therefore, $\xi(o)$ is in the value set of the root in J as o is in Σ.

To prove that J has no object of ξ-value $\xi(o)$ other than o, we consider an object p of J and prove that $\phi^{-1}(\xi(p))$ is even if and only if $p \neq o$. By the definition of ξ, $\xi(p) = \nu(p)$ if $p \neq o$; otherwise $\xi(p) = (\mu(o) \times_\phi 2) +_\phi 1$. Also, $\nu(p) = (\mu \times_\phi 2)(p) = \mu(p) \times_\phi 2 = \phi(\phi^{-1}(\mu(p)) \times 2)$ and hence $\phi^{-1}(\nu(p)) = 2\phi^{-1}(\mu(p))$. Therefore, if $p \neq o$ then $\phi^{-1}(\xi(p)) = \phi^{-1}(\nu(p)) = 2\phi^{-1}(\mu(p))$; otherwise $\xi(p) = (\mu(o) \times_\phi 2) +_\phi 1 = (\mu(p) \times_\phi 2) +_\phi 1 = \nu(p) +_\phi 1 = \phi(\phi^{-1}(\nu(p)) + 1) = \phi(2\phi^{-1}(\mu(p)) + 1)$ and thus $\phi^{-1}(\xi(p)) = 2\phi^{-1}(\mu(p)) + 1$. Consequently, $\phi^{-1}(\xi(p))$ is even if and only if $p \neq o$. So, o is the only object whose ξ-value is not ϕ-even and hence J has no object of ξ-value $\xi(o)$ other than o.

Now we can prove that l has no intersection ancestor. As we proved $\xi(o)$ is in the value set of the root in J, by Lemma 2.22 there is a sub-intersection tree T of Q such that every leaf of T has an object of ξ-value $\xi(o)$. Hence, as the ξ-value of no object other than o equals $\xi(o)$, o is an object of every leaf of T. So, as by Definition 2.4 o is the object of only one leaf of Q, T has no leaf other than l. Also, by Definition 2.18, if T contains an internal node v, it contains a child of v too, and recursively applying this fact, we may conclude that if T contains a node v then T contains a leaf in $Q[v]$. Thus, as by Definition 2.18 T contains the root, T contains a leaf of Q and so, since T has no leaf other than l, l is a leaf of T. Therefore, as T is a subtree containing the root and l, every node on the path connecting l to the root of Q is in T and so all ancestors of l are in T. Now assume to the contrary that l has an intersection ancestor w. As Q is intersection-normalized, w has at least two children u_1 and u_2. Since w is an ancestor of l, w is in T and thus by Definition 2.18 u_1 and u_2 are also in T. So, a leaf from each of $Q[u_1]$ and $Q[u_2]$ is in T, contradicting the fact that l does not have more than one leaf. Therefore, l has no intersection ancestor and hence the lemma is true. □

We now formally divide the problem of solving SimpleEval for an instance with a query tree with more than one speedy leaf to two subproblems. Given a union instance $I = (Q, \omega, \mu)$ in which $l_1, \ldots, l_k$ are all speedy leaves of Q and $k > 1$, we define the instances $\mathit{First}(I)$ and $\mathit{Second}(I)$ as follows.

1. The instance $\mathit{First}(I)$ is defined as $(\mathit{First}(Q), \omega_F, \mu)$ where $\mathit{First}(Q)$ is a query tree of height one having k leaves $v_1, \ldots, v_k$ and for every i, $1 \leq i \leq k$, $\omega_F(v_i) = \omega(l_i)$.

2. The instance $\mathit{Second}(I)$ is defined as $(\mathit{Second}(Q), \omega_S, \mu)$, where $\mathit{Second}(Q)$ is created by omitting all leaves $l_2, l_3, \ldots, l_k$ from Q and ω_S is defined as follows:

$$\omega_S(l) = \begin{cases} \omega(l) & \text{if } l \neq l_1 \\ \Sigma_1 & \text{if } l = l_1 \end{cases}$$

   where Σ_1 is an arbitrary solution to $\mathit{First}(I)$.

As $\mathit{Second}(Q)$ is created from Q by omitting $k-1$ of its speedy leaves, it is clear that $\mathit{Second}(Q)$ has exactly one speedy leaf. Also, $\mathit{First}(Q)$ is of height one. Note that in order

to construct *Second*(I) one need to first solve *First*(I) and obtain a solution Σ to *First*(I). We now prove that a solution to *Second*(I) is a solution to I.

Lemma 2.35 *Consider a union instance $I = (Q, \omega, \mu)$ in which Q has more than one speedy leaf. Then, a solution to Second(I) is a solution to I.*

Proof. Defining $l_1, \ldots, l_k$ as all speedy leaves of Q and $u_l, \ldots, u_m$ as non-leaf children of the root in Q, we first prove that the value sets of u_i in I and in *Second*(I) are the same, for every i, $1 \leq i \leq m$. Then we show that the value set of the root in I is the same as the value set of the root in *Second*(I). After that, using these facts we prove the lemma.

Now let us prove that given an integer i, $1 \leq i \leq m$, the value sets of u_i in I and in *Second*(I) are the same. Since *Second*(Q) has created from Q by just deleting some leaf children of the root, *Second*$(Q)[u_i] = Q[u_i]$. Also, for every leaf l of $Q[u_i]$, $l \neq l_1$ and thus $\omega_S(l) = \omega(l)$. Moreover, the value functions of I and *Second*(I) are the same. Hence, by applying Lemma 2.2 for $v = u_i$, we may conclude that the value sets of u_i in I and in *Second*(I) are the same.

We now show that *RootValueSet*(I) = *RootValueSet*(*Second*(I)). Suppose $\mathcal{S}_1, \ldots, \mathcal{S}_k$ are sets of values of objects in $\omega(l_1), \ldots, \omega(l_k)$, respectively. As $\omega(l_i) = \omega_F(l_i)$, by Definition 2.9 value sets of l_i in I and in *First*(I) are both equal to $\mathcal{S}_i$, for every i, $1 \leq i \leq k$. Thus, defining $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{S}_i$, by definition *RootValueSet*(*First*(I)) $= \mathcal{S}$. Therefore, as by definition $\omega_S(l_1)$ is a solution to *First*(I), the set of values of objects in $\omega_S(l_1)$ is $\mathcal{S}$ and so the value set of l_1 in *Second*(I) equals $\mathcal{S}$. Also, defining $\mathcal{T}_i$ as the value set of u_i in I, as we proved, $\mathcal{T}_i$ is the value set of u_i in *Second*(I), for every i, $1 \leq i \leq m$. Consequently, as $l_1, u_1, \ldots, u_m$ are the children of the root in *Second*(Q), *RootValueSet*(*Second*(I)) $= \mathcal{S} \cup \bigcup_{j=1}^{m} \mathcal{T}_j = \bigcup_{i=1}^{k} \mathcal{S}_i \cup \bigcup_{j=1}^{m} \mathcal{T}_j$. But $l_1, \ldots, l_k, u_1, \ldots,$ and u_m are children of the root in I and we proved $\mathcal{S}_i$ is the value set of l_i in I, for every i, $1 \leq i \leq k$, and by assumption $\mathcal{T}_j$ is the value set of u_j in I, for every j, $1 \leq j \leq m$. Hence, *RootValueSet*$(I) = \bigcup_{i=1}^{k} \mathcal{S}_i \cup \bigcup_{j=1}^{m} \mathcal{T}_j =$ *RootValueSet*(*Second*(I)).

We now show that a solution Σ_S to *Second*(I) is a solution to I. By Definition 2.10 objects in Σ_S have distinct μ-values and appear in Σ_S in increasing order of μ-value and the set of values of objects in Σ_S equals *RootValueSet*(*Second*(I)) = *RootValueSet*(I). Hence, Σ_S satisfies all conditions of Definition 2.10 for I and thus is a solution to I. □

Now there are three remaining issues to be discussed. The first issue is that how to solve the instance $\mathit{First}(I)$. This instance is a union instance in which the query tree is of height one. The problem in this case has been studied extensively by Demaine et al. [DLOM00] and we refer the reader to their work.

The second issue is that once SimpleEval is solved for $\mathit{First}(I)$, we still need a solution Σ_F to $\mathit{First}(I)$ to create the instance $\mathit{Second}(I)$. By definition, the output of an algorithm executed on $\mathit{First}(I)$ is a sequence Φ_F of output items such that the expansion of Φ_F is a solution to $\mathit{First}(I)$. Hence, one approach is to create a solution as described in Definition 2.11. But the time that this approach takes is linear in the number of objects of the solution of $\mathit{First}(I)$ while neither the algorithm proposed by Demaine et al. for solving $\mathit{First}(I)$ nor the algorithm that we describe in this thesis for solving $\mathit{Second}(I)$ need that much time to solve $\mathit{First}(I)$ and $\mathit{Second}(I)$. So, this is not an ideal approach and hence this part remains as an open problem to be discussed.

The last issue is that once we created $\mathit{Second}(I)$ and solved SimpleEval for $\mathit{Second}(I)$, we have a sequence Φ_S of output items for $\mathit{Second}(I)$ such that the expansion Σ_S of Φ_S is a solution to $\mathit{Second}(I)$ and thus by Lemma 2.35 Σ_S is a solution to I. Now, the problem is that Φ_S is a sequence of output items for $\mathit{Second}(I)$, not for I. So, we must design a method for constructing a sequence of output items for I with the same expansion as Φ_S. Again one solution is to evaluate the expansion of Φ_S and then corresponding to every object o in this expansion create an output item for I. But due to the same reason as above this is not an ideal solution and this issue remains to be discussed, as well.

In the rest of the thesis we will discuss the problem in the case in which the query tree does not have more than one speedy leaf.

2.6 Difficulty

This section defines the difficulty of an instance in which the query tree has at most one speedy leaf. We use a model similar to the difficulty of instances of height one defined by Demaine et al. [DLOM00], but more complex. We define a parameterized class of difficulty functions. Then we present a general lower bound on the adaptive running time of algorithms for any difficulty function in this class.

Given an algorithm $\mathcal{A}$ and an instance I, the set of comparison propositions in $C_{\mathcal{A}}(I)$ is a proof for I, by Lemma 2.10. Informally speaking, this means that given an instance I, a comparison-based algorithm computes a proof for I. Because of this, we define the cost of a proof $\mathcal{P}$ of I as the effort needed to compute $\mathcal{P}$ and then we define the difficulty of an instance I as the minimum cost of any proof of I.

Now we informally explain the idea behind the definition of the cost of a proof. As is proved in the next chapter, every proof $\mathcal{P}$ consists of a number of subproofs, each proving a value a is (or is not) in the value set of the root. Now in a more general setting, we consider a comparison-based algorithm $\mathcal{A}$, a proof $\mathcal{P}$ for an instance $I = (Q, \omega, \mu)$, and a subproof of $\mathcal{P}$ proving a value a is or is not in the value set of a node v of Q. Then we discuss the effort needed to compute such a subproof in each of the two cases in which v is a leaf and v is an internal node, separately.

First we consider the case in which v is a leaf. If o and p are two consecutive objects of $\omega(v)$ such that $\mu(o) \leq a$ and $a < \mu(p)$, the two propositions $\mu(o) < a$ and $a < \mu(p)$ (or just $a = \mu(o)$) suffice to prove a is not (or is) in the value set of v. Also, in order to prove a is not in the value set of v, as we will prove, $\mathcal{P}$ has to visit o. Now suppose there are g consecutive $\mathcal{P}$-skipped objects just before o. Among o and the g objects before o, o is the only one visited by $\mathcal{P}$. Intuitively, according to information that $\mathcal{A}$ has gathered before $\mathcal{A}$ visits any of these g objects, potentially each of these objects can be the biggest object with a value of at most a. So, $\mathcal{A}$ has to find the biggest object with a value of at most a among at least $1 + g$ candidates and then compare it to a. This requires at least $1 + \log_3 g$ comparisons in the worst case because there are three choices for the comparison proposition resulting from each comparison and so, informally speaking, every comparison eliminates at most $\frac{2}{3}$ of these $1 + g$ possibilities in the worst case. As we define more formally in the next definition, these $1 + g$ objects create a $\mathcal{P}$-gap.

Definition 2.26 *Suppose $\mathcal{P}$ is a proof for the instance $I = (Q, \omega, \mu)$, l is a leaf of Q and $\omega(l) = o_1, \ldots, o_n$. Also, $o_{x_1}, o_{x_2}, \ldots, o_{x_m}$ is the sequence of all objects of $\omega(l)$ that are visited by $\mathcal{P}$ where $x_j < x_{j+1}$ for every j, $1 \leq j < m$. If $m > 0$, each of the sequences $o_1, o_2, \ldots, o_{x_1}$ and for every j, $1 < j \leq m$, $o_{x_{j-1}+1}, o_{x_{j-1}+2}, \ldots, o_{x_j}$ is a $\mathcal{P}$-gap.*

Now imagine we want to prove that a value a is not in the value set of an internal node v. If v is a union node, it is clear what we have to do: we need to prove that a is not in the

value set of any child of v. So consider the case in which v is an intersection node. Then, we have to find a child w of v such that a is not in the value set of w. But as we do not know which child is w, we have to start solving the problem for all children of v until the problem is solved for w and so we know a is not in the value set of w. Thus, when defining the cost of a proof, we must consider the overhead caused by processing other children of v. This can be done by counting every object of $Q[v]$ visited by $\mathcal{P}$ k times, where k is the number of children of v, if we suppose the processing time is equivalently divided among the subtrees rooted at the k children of v. We put it in a more general way. Let us define $\mathcal{S}$ to be the set of children of v. For every child u of v, we consider a weight $L^-(u)$ and we define $L^-(v) = \sum_{u \in \mathcal{S}} L^-(u)$, meaning that our attempt to prove that a is not in the value set of u would take $\frac{L^-(u)}{L^-(v)}$ of the whole time spent for proving that a is not in the value set of v. Then we count every object of $Q[u]$ visited by $\mathcal{P}$, for a child u of v, $\frac{L^-(v)}{L^-(u)}$ times as the time spent for v is $\frac{L^-(v)}{L^-(u)}$ of the time spent for u.

Now consider the situation in which we want to prove that a value a is in the value set of a node v. This time if v is a union node we are in a situation in which we do not know which child of v should be investigated and hence we have some overhead. Using an idea similar to what we explained for the previous case, we consider an arbitrary weight $L^+(w)$, for every child w of a union node v, and we define $L^+(v) = \sum_{w \in \mathcal{S}} L^+(w)$ where $\mathcal{S}$ is the set of children of v. Then we count every object of $Q[w]$ visited by $\mathcal{P}$, for a child w of v, $\frac{L^+(v)}{L^+(w)}$ times.

Motivated by the reasons we mentioned, we define the parametric family of difficulty functions where the parameter is a function defining arbitrary weights as discussed earlier. This parameter is defined formally as follows.

Definition 2.27 *A* negative (positive) workload function for a query tree Q *is a function* L^- (L^+, *respectively) assigning a positive number to every node of* Q *such that the following properties hold:*

1. *The number assigned to every leaf is one.*
2. *The number assigned to an intersection node (a union node, respectively)* u *equals the sum of the numbers assigned to children of* u.

3. *The number assigned to a union node (an intersection, respectively) u is at least one and is at most the sum of numbers assigned to children of u.*

Now, the parameter is a *workload setting L for Q* which is defined to be a pair (L^-, L^+) where L^- and L^+ are a negative workload function and a positive workload function for Q, respectively. Given a query tree Q and a workload setting $L = (L^-, L^+)$ for Q, we define two functions W_L^- and W_L^+ where for every node v each returns a real number. Values of these functions for a leaf l will be used to determine how many times a visited objects of l should be counted.

Definition 2.28 *Given a query tree Q, a workload setting $L = (L^-, L^+)$ for Q, and a node v of Q, we define $W_L^-(v)$ and $W_L^+(v)$ as follows. If v is the root of Q, $W_L^-(v)$ and $W_L^+(v)$ both are defined as 1. Otherwise, if u is the parent of v, $W_L^-(v)$ and $W_L^+(v)$ are defined in the following manner.*

$$W_L^-(v) = \begin{cases} W_L^-(u) & \text{if } u \text{ is a union node} \\ W_L^-(u) \times \frac{L^-(u)}{L^-(v)} & \text{otherwise} \end{cases}$$

$$W_L^+(v) = \begin{cases} W_L^+(u) \times \frac{L^+(u)}{L^+(v)} & \text{if } u \text{ is a union node} \\ W_L^+(u) & \text{otherwise} \end{cases}$$

Moreover, given a node v of Q, we define $W_L(v) = W_L^-(v) + W_L^+(v)$.

Now, given a proof $\mathcal{P}$, as explained before Definition 2.27, we count a visited object o of a leaf l of Q $W_L(l)(1 + \log_3 g)$ times where g is the length of the $\mathcal{P}$-gap ending with o. Since we do not mind constant factors here, we consider a factor $W_L(l)(1 + \log g)$, instead of $W_L(l)(1 + \log_3 g)$. So, the *L-gap cost of the proof $\mathcal{P}$* is defined as follows.

$$D_L(\mathcal{P}) = \sum_{l \in \text{leaves}(Q)} \left(W_L(l) \sum_{1 \le i \le m_l} 1 + \log g_l[i] \right) \tag{2.3}$$

where for a leaf l of Q, m_l is the number of visited objects in $\omega(l)$, and $g_l[1]$, $g_l[2]$, ..., $g_l[m_l]$ denote lengths of $\mathcal{P}$-gaps of l. Then, the difficulty of the instance I in which the

query tree has at most one speedy leaf is defined as

$$D_L(I) = \min_{\mathcal{P} \in \mathcal{T}} D_L(\mathcal{P}), \tag{2.4}$$

where $\mathcal{T}$ is the set of all proofs of I. Given an instance $I = (Q, \omega, \mu)$ such that Q has more than one speedy leaf and a workload setting L for $\mathit{Second}(Q)$, the difficulty function D_L is defined as $D_L(I) = D^{\cup}(\mathit{First}(I)) + D_L(\mathit{Second}(I))$.

In the next two sections we show that how a proof can be converted to a sequence of subproofs each showing a group of objects is or is not in the value set of the root and then in Chapter 5 we will use the structure of that type of sequence to develop our algorithm.

Chapter 3

Eliminating Proofs

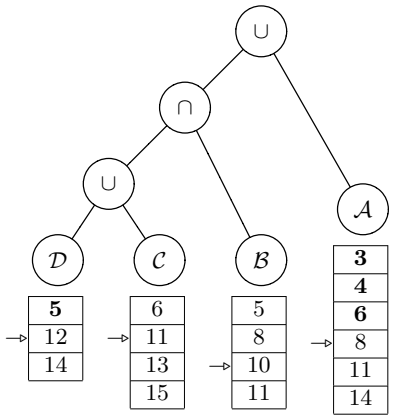
As noted in the last chapter, the set of objects visited by a proof can form a number of subproofs, each proving a specific value (or a range of values) is or is not in the value set of the root. In this chapter we show how a proof can be rearranged to a form called an eliminating proof that is a sequence of such subproofs. The idea of this structure will be used in the next chapter to develop the algorithm. We will also use this structure in Chapter 6 to analyze the running time of the algorithm.

Let us first explain the idea of the structure. We construct a solution to an instance I by considering an empty sequence and then by adding new objects to it, step by step, such that after each step we have a partial solution, defined formally as follows.

Definition 3.1 *Given an instance $I = (Q, \omega, \mu)$, a* partial solution to I *is a sequence Σ of objects of I such that the value of every object in Σ is in the value set of the root and for every two consecutive objects o_1 and o_2 of Σ, $\mu(o_1) < \mu(o_2)$.*

Now suppose we are in the middle of accomplishing the task of adding new objects to construct a solution. Consider the set $\mathcal{S}$ of all values a such that an object of value a is in our current partial solution. Objects are added to the partial solution in order of value so that a variable $\mathsf{e_s}$, whose value is a member of $\mathbb{V}^+$, marks a boundary between values in the value set of the root that are in $\mathcal{S}$ and those that are not in $\mathcal{S}$. As a result, for every value a less than $\mathsf{e_s}$, if a is in the value set of the root, an object of value a already has been added to the partial solution. Consequently, informally speaking, an object o of a value

Figure 3.1 An eliminating configuration



$\mathsf{e} = 10$, $\mathsf{e_s} = 8$, $\Sigma = 3, 4, 5, 6$

$\Phi = (\mathcal{A}, 1, 2), (\mathcal{D}, 1, 1), (\mathcal{A}, 3, 3)$

less than $\mathsf{e_s}$ is not going to be used to extend the current partial solution to a solution and so o can be ignored from now on. As initially no object is added to our partial solution, we set $\mathsf{e_s} = -\infty$ so that $\mathsf{e_s}$ is less than values of all objects.

We define an additional variable e that plays the same role as $\mathsf{e_s}$, but for $N(I)$ (defined in Section 2.5) instead of I, which means that for every value a in the value set of the root in $N(I)$, an object of value a has already been added to the partial solution if and only if $a < \mathsf{e}$. Now, given an object o of $N(I)$ with a value less than e, as we will prove in this chapter, we can ignore o from now on without missing any value in the value set of the root in I. In our structure, due to the reason that we now explain, we define e to be a value at least as large as $\mathsf{e_s}$. One can observe that, given a value a in the value set of the root in $N(I)$, by Observation 2.32 a is in the value set of the root in I and so an object of value a is in the partial solution if and only if $a < \mathsf{e_s}$. So, by setting $\mathsf{e} := \mathsf{e_s}$ the property described for e is not violated. Moreover, the bigger the value of e is, the more objects of $N(I)$ we can ignore from now on. So, we suppose $\mathsf{e} \geq \mathsf{e_s}$.

Figure 3.1 shows an example. In the figure, Σ is the partial solution generated so far. Each object is shown as a rectangle and the value inside the rectangle is the value of the object. Objects above the object indicated by the arrow shape in each leaf are objects

that can be ignored. The expansion of the sequence Φ of output items (Definition 2.11) in the figure is the partial solution Σ as can be seen. This sequence of output items and the current values of e and $\mathsf{e_s}$ create our current eliminating configuration as defined in the following definition. Note that if the query tree has no speedy leaf, e and $\mathsf{e_s}$ have the same meaning and so we always set $\mathsf{e_s} = \mathsf{e}$.

Definition 3.2 *Given a signature $G = (Q, \omega)$ and an instance $I = (Q, \omega, \mu)$, an* eliminating configuration for G *or* for I *is a triple $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$, where e and $\mathsf{e_s}$ are values in $\mathbb{V}^+$ and Φ is a sequence of output items for I such that if Q has no speedy leaf, $\mathsf{e_s} = \mathsf{e}$ and otherwise, $\mathsf{e_s} \leq \mathsf{e}$. In addition, e, $\mathsf{e_s}$, and Φ are called the* e-value*, the* $\mathsf{e_s}$-value*, and the* output of C.

Now let us formally we define which objects can be ignored. As we proceed, we complete our partial solution and during this progress, all objects of $N(I)$ with values less than the e-value of the current eliminating configuration C can be ignored. An object of I that is not in $N(I)$, and so is an object of a speedy leaf, can be ignored only if its value is less than the $\mathsf{e_s}$-value of C. Considering these facts, given an eliminating configuration C, we define C-eliminated objects as objects that can be ignored in the future. The next definition defines this concept more precisely.

Definition 3.3 *Given an instance I, an eliminating configuration C of I, and an object o of I, o is* C-eliminated *if it is an object of $N(I)$ and has a value less than e or it is an object of a speedy leaf and has a value less than $\mathsf{e_s}$. An object that is not C-eliminated is a* C-remaining object.

We may omit C in terms "C-eliminated" and "C-remaining" when C is clear from the context (for example in an informal argument when we are talking about the current eliminating configuration).

Now we introduce a function for evaluating the smallest C-remaining object of a leaf l. In order to evaluate the smallest C-remaining object of l, for a non-speedy leaf l (a speedy leaf) and an eliminating configuration C, one must find the smallest object in the sequence associated with l of value at least e (at least $\mathsf{e_s}$, respectively) by the previous definition. This object can be found using the functions defined in the next definition. Recall from Definition 2.7 the definition of $\Sigma[i]$, for a sequence Σ of objects and an integer i.

Definition 3.4 *Suppose $I = (Q, \omega, \mu)$ is an instance, l is a leaf of Q, and b is a value in $\mathbb{V}^+$. Also, suppose $\omega(l) = o_1, \ldots, o_k$. We define $pos(I, l, b) = k + 1$ if values of all objects of $\omega(l)$ are less than b and otherwise, we define $pos(I, l, b)$ to be the minimum i, $1 \leq i \leq k$, such that $\mu(o_i) \geq b$. Also, we define $find(I, l, b) = \omega(l)[pos(I, l, b)]$.*

We now present an equivalent definition for the function *find*. Given an instance $I = (Q, \omega, \mu)$, a leaf l of Q, and a value b in $\mathbb{V}^+$, if there is any object of value at least b in $\omega(l)$, $pos(I, l, b)$ is the index i of the smallest object o of value at least b in $\omega(l)$ and so $find(I, l, b) = \Sigma[i] = o$. On the other hand, if there is no object of value at least b, $pos(I, l, b)$ is one plus the length k of $\omega(l)$ and so $find(I, l, b) = \omega(l)[1 + k] = \text{END}$. This proves the next lemma.

Lemma 3.1 *Given an instance $I = (Q, \omega, \mu)$, a leaf l of Q, and a value b in $\mathbb{V}^+$, if there is any object of value at least b in $\omega(l)$, $find(I, l, b)$ is the smallest object of value at least b in $\omega(l)$; otherwise, $find(I, l, b) = \text{END}$.* □

As a special case of Lemma 3.1 consider a leaf l of an instance I having an object o of value a, for some a. Then o is the smallest object of l with a value at least a and thus by Lemma 3.1 $find(I, l, a) = o$. This yields the following lemma.

Lemma 3.2 *Given an instance $I = (Q, \omega, \mu)$, a leaf l of Q, and a value a in the value set of l, $find(I, l, a)$ is an object of value a in $\omega(l)$.* □

The next lemma shows that, given an instance $I = (Q, \omega, \mu)$, a leaf l of Q, and a member b of $\mathbb{V}^+$, $pos(I, l, b)$ marks a boundary between indices of objects in $\omega(l)$ with values less than b and indices of the rest of objects in $\omega(l)$.

Lemma 3.3 *Consider an instance $I = (Q, \omega, \mu)$, a value b in $\mathbb{V}^+$, and a leaf l of Q where $\omega(l) = o_1, \ldots, o_k$. Given an integer i, $1 \leq i \leq k$, $pos(I, l, b) \leq i$ if and only if $b \leq \mu(o_i)$.*

Proof. We prove the lemma by considering separately each of the two cases considered in Definition 3.4. First suppose there is no object with a value at least b in $\omega(l)$ and hence $b \not\leq \mu(o_i)$. Then, by Definition 3.4 $pos(I, l, \mathsf{e}) = k + 1$ and so as by the assumption of the lemma $i \leq k$, $pos(I, l, \mathsf{e}) \not\leq i$, proving the lemma in this case. Now suppose there is an object of value at least b in $\omega(l)$ and hence by Definition 3.4 $pos(I, l, \mathsf{e})$ is the smallest

j satisfying the inequality $b \leq \mu(o_j)$. Therefore, $b \leq \mu(o_i)$ if and only if $j \leq i$. So, as $j = pos(I, l, \mathsf{e})$, the lemma is true in this case, as well. □

The next lemma establishes a relationship between the functions *pos* and *find* when they are applied to different instances.

Lemma 3.4 *Given two instances $I = (Q, \omega, \mu)$ and $J = (Q, \omega, \nu)$, a leaf l of Q, and two values a and b in $\mathbb{V}^+$, if $find(I, l, a) = find(J, l, b)$ then $pos(I, l, a) = pos(J, l, b)$.*

Proof. We prove the lemma in the two cases $find(I, l, a) = \text{END}$ and $find(I, l, a) \neq \text{END}$ separately. We suppose $\omega(l) = o_1, \ldots, o_n$. First suppose $find(I, l, a) = \text{END}$ and so $find(J, l, b) = \text{END}$. Then, as $find(I, l, a) = \omega(l)[pos(I, l, a)]$ and $find(J, l, b) = \omega(l)[pos(J, l, b)]$ (Definition 3.4), $\omega(l)[pos(I, l, a)] = \omega(l)[pos(J, l, b)] = \text{END}$. Hence, it follows from Definition 2.7 that $pos(I, l, a) = n + 1$ and $pos(J, l, b) = n + 1$. So the claim is true in this case.

Next consider the case in which $find(I, l, a) \neq \text{END}$ and hence by Lemma 3.1 $find(I, l, a) = o_i$, for an object o_i of $\omega(l)$. Thus, $find(I, l, a) = find(J, l, b) = o_i$, where $1 \leq i \leq n$, and so by Definition 3.4 $\omega(l)[pos(I, l, a)] = find(I, l, a) = \omega(l)[i]$ and $\omega(l)[pos(J, l, b)] = find(J, l, b) = \omega(l)[i]$. Therefore, $pos(I, l, a) = i = pos(J, l, b)$ as by Definition 2.7 $j = i$ the only choice for j such that $\omega(l)[j] = o_i$. □

Now using Lemma 3.1 we show how the function *find* can be used to evaluate the smallest remaining object of a leaf. Given an instance $I = (Q, \omega, \mu)$, an eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for I, and a non-speedy leaf l of Q, by Definition 3.3, there is a C-remaining object in $\omega(l)$ if and only if there is an object of value at least e in $\omega(l)$, that is, $find(I, l, b) \neq \text{END}$. Furthermore, the smallest remaining object of $\omega(l)$, if there is any, is the same as the smallest object of $\omega(l)$ of value at least e, that is, $find(I, l, \mathsf{e})$. This proves the following lemma.

Lemma 3.5 *Given an eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ of an instance $I = (Q, \omega, \mu)$ and a leaf l of $N(Q)$, if there is any C-remaining object in $\omega(l)$, the smallest C-remaining object of $\omega(l)$ is $find(I, l, \mathsf{e})$; otherwise, $find(I, l, \mathsf{e}) = \text{END}$.* □

The next lemma is a result similar to the previous lemma, but for speedy leaves instead of

non-speedy leaves. To prove it, it suffices to replace all occurrences of the term "e" in the argument by "$\mathsf{e_s}$".

Lemma 3.6 *Given an eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for an instance $I = (Q, \omega, \mu)$, if Q has one speedy leaf and there is any C-remaining object in $\omega(sl(Q))$, $find(I, sl(Q), \mathsf{e_s})$ is the smallest C-remaining object of $\omega(sl(Q))$; otherwise, $find(I, sl(Q), \mathsf{e_s}) = \text{END}$.* □

The following definition establishes formally the connection between values of variables e and $\mathsf{e_s}$ and values of objects in the current partial solution.

Definition 3.5 *An eliminating configuration $(\mathsf{e}, \mathsf{e_s}, \Phi)$ for a signature (Q, ω) is* valid for an instance $I = (Q, \omega, \mu)$ *if the following conditions hold.*

1. *For every value a in the value set of the root, an object of value a is in the expansion of Φ if and only if $a < \mathsf{e_s}$.*

2. *For every value a in $RootValueSet(N(I))$, an object of value a is in the expansion of Φ if and only if $a < \mathsf{e}$.*

3. *The expansion of Φ is a partial solution to I.*

We now discuss some properties of Definition 3.5. One property is that this definition depends on the sequence of values of objects in the expansion of the output of the eliminating configuration rather than actual objects in this expansion. We prove this fact in the next lemma.

Lemma 3.7 *Consider two eliminating configurations $C_1 = (\mathsf{e}, \mathsf{e_s}, \Phi_1)$ and $C_2 = (\mathsf{e}, \mathsf{e_s}, \Phi_2)$ such that the sequence Γ_1 of values of objects in the expansion of Φ_1 is the same as the sequence Γ_2 of values of objects in the expansion of Φ_2. Then, C_1 is valid if and only if C_2 is valid.*

Proof. We consider the three conditions of Definition 3.5 one by one and for each one we prove C_1 satisfies that condition if and only if C_2 satisfies it. The proof of this claim for the first two conditions is trivial since it follows from the assumption of the lemma that, given a value a, an object of value a is in the expansion of Φ_1 if and only if an object of value a in the expansion of Φ_2. To prove the claim for the third condition we must prove that the

expansion of Φ_1 is a partial solution if and only if the expansion of Φ_2 is a partial solution. It follows from Definition 3.1 that the sequence Φ_1 (the sequence Φ_2) is a partial solution if and only if every element of Γ_1 (of Γ_2) is in the value set of the root and elements of Γ_1 (of Γ_2) appear in increasing order. Hence, as by assumption $\Gamma_1 = \Gamma_2$, Φ_1 is a partial solution if and only if Φ_2 is a partial solution. So, the lemma is correct. □

Another property of valid eliminating configurations is that, as the next lemma shows, for every value b, $\mathsf{e_s} \leq b < \mathsf{e}$, b is not in $\mathit{RootValueSet}(N(I))$ and hence, due to Observation 2.32, if b is in the value set of the root then b is in the value set of a speedy leaf. This fact will be used to prove that the remaining objects of the speedy leaf (if it exists) with values less than e are the next objects that should be added to our partial solution.

Lemma 3.8 *Given an instance I, a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$, and a value b in $\mathit{RootValueSet}(N(I))$, if $b \geq \mathsf{e_s}$ then $b \geq \mathsf{e}$.*

Proof. The proof is almost an immediate conclusion of the first and the second conditions of Definition 3.5. As $b \in \mathit{RootValueSet}(N(I))$, $b \in \mathit{RootValueSet}(I)$ since by Observation 2.32 $\mathit{RootValueSet}(I)$ is the union of $\mathit{RootValueSet}(N(I))$ and value sets of speedy leaves. So, since $b \geq \mathsf{e_s}$, by the first condition of Definition 3.5, no object of value b is in the expansion of Φ and hence $b \geq \mathsf{e}$, by the second condition in Definition 3.5. □

In the next section we explain how we create a valid eliminating configuration C of an instance I so that all objects of I are C-eliminated. Then, as the next lemma shows, the expansion Σ of the output of C is a solution to I and so by constructing C we have created a solution to I.

Lemma 3.9 *Suppose C is a valid eliminating configuration for an instance I such that all objects of I are C-eliminated. Then, the expansion Σ of the output of C is a solution to I.*

Proof. We prove that Σ satisfies the three conditions of the definition of solutions (Definition 2.10) in the following manner. We first show that as Σ is a partial solution, it satisfies the last two conditions of Definition 2.10 and, defining $\mathcal{S}$ as the set of values of objects in Σ, $\mathit{RootValueSet}(I) \subseteq \mathcal{S}$. Then, we prove that $\mathcal{S} \subseteq \mathit{RootValueSet}(I)$ in two cases. In

this way, it is proved that $\mathit{RootValueSet}(I) = \mathcal{S}$ and thus Σ satisfies the first condition of Definition 2.10, as well.

We first prove that Σ satisfies the first two conditions of Definition 2.10 and also that $\mathcal{S} \subseteq \mathit{RootValueSet}(I)$. The sequence Σ is a partial solution to I by the last condition of Definition 3.5. Hence, the last two conditions of Definition 2.10 follow immediately from the second condition of Definition 3.1 for the output Σ. Moreover, by Definition 3.1 $\mu(o)$ is in the value set of the root, for every object o of Σ. So, $\mathcal{S} \subseteq \mathit{RootValueSet}(I)$.

Now we prove that $\mathit{RootValueSet}(I) \subseteq \mathcal{S}$ by considering a member a of the value set of the root and by showing that there is an object of value a in Σ. We consider two cases based on a being in $\mathit{RootValueSet}(N(I))$ or not. First we consider the case $a \in \mathit{RootValueSet}(N(I))$. Consider an arbitrary sub-union tree T of $N(Q)$. Since a is in $\mathit{RootValueSet}(N(I))$, by Lemma 2.22 there is a leaf l of T having an object o of value a. As all objects are C-eliminated and o is an object of the leaf l of a sub-union tree of $N(Q)$, o is a C-eliminated object of $N(I)$ and thus by Definition 3.3 $a = \mu(o) < \mathsf{e}$. Therefore, since by assumption $a \in \mathit{RootValueSet}(N(I))$, by the second condition of Definition 3.5 an object of value a is in Σ.

Next we consider the case $a \notin \mathit{RootValueSet}(N(I))$. Then, since $a \in \mathit{RootValueSet}(I)$, a is in the value set of a speedy leaf, due to Observation 2.32, and hence a is the value of an object p of a speedy leaf. So, since by the assumption of the lemma p is C-eliminated, by Definition 3.3 $a = \mu(p) < \mathsf{e_s}$ and thus as a is in the value set of the root, by the first condition of Definition 3.5 an object of value a is in Σ in this case as well. Hence, the lemma is correct. □

In the next section we discuss how to eliminate all objects.

3.1 Eliminating Steps

In this part we define a number of functions that, given an instance I and a valid eliminating configuration of I, create a new valid eliminating configuration with a bigger e-value or $\mathsf{e_s}$-value so that more objects are eliminated. Therefore, by repeatedly applying these functions eventually we get an eliminating configuration C such that there is no C-remaining object and so by Lemma 3.9 the output of C is a solution to the instance.

Definition 3.6 *Given a signature $G = (Q, \omega)$ in which Q has at most one speedy leaf, an* eliminating function for G *is a function $\pi : \mathcal{F}(G) \times \mathcal{C}(G) \mapsto \mathcal{C}(G)$, where $\mathcal{F}(G)$ and $\mathcal{C}(G)$ are the set of all ordered value functions μ for G and the set of all eliminating configurations for G, respectively.*

We define three types of eliminating functions each increasing e or $\mathsf{e_s}$ in its own way. We first describe these three types very briefly and in an informal manner and then we define each of them more precisely in the next sections. An important fact that will be used in the following discussion is that given a valid eliminating configuration $(\mathsf{e}, \mathsf{e_s}, \Phi)$ of an instance I, for every value a less than e, if a is in $\mathit{RootValueSet}(N(I))$, an object of value a has already been added to the expansion of Φ, according to the second condition in Definition 3.5. As a result, if we want to add some objects to the expansion of Φ to build a solution to I, we can ignore objects of $N(I)$ of values less than e.

When using the first type of eliminating function, we first prove that every member of $\mathit{RootValueSet}(N(I))$ that is at least e is also at least a, for a value a, $\mathsf{e} < a$. Hence, in addition to objects of $N(I)$ with values less than e, objects of $N(I)$ with values b, such that $\mathsf{e} \leq b < a$, can be ignored since their values are not in $\mathit{RootValueSet}(N(I))$. Thus, we can set e equal to a as all objects of $N(I)$ with values less than a can be ignored.

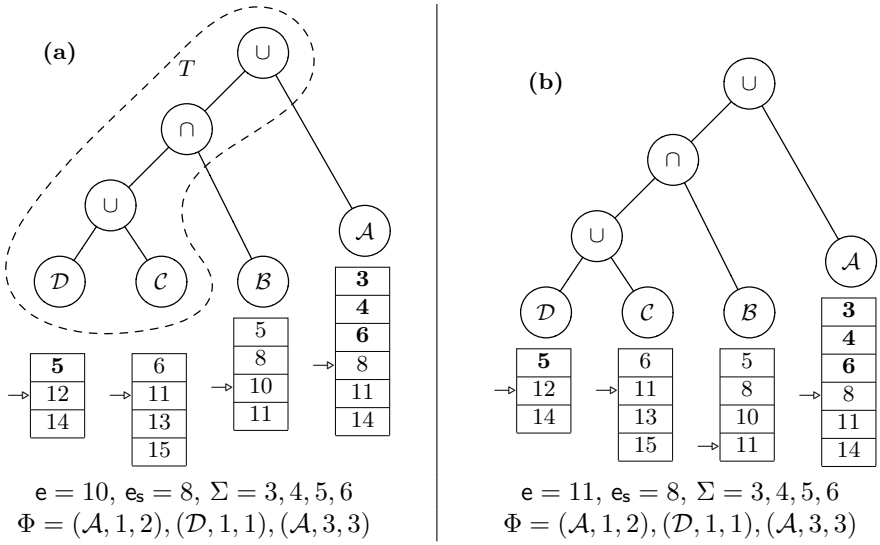
When using an eliminating function of the second type, we first prove that e is in the value set of the root and we add an object of value e to the current partial solution. Then e is set equal to $\mathsf{e}+$ since after adding an object of value e to our partial solution, in addition to objects with values less than e, objects of value e can be ignored.

The third type of eliminating function is for the situation in which all objects of $N(I)$ are eliminated. So all remaining objects belong to speedy leaves. These objects are added to our partial solution and then they are eliminated by setting $\mathsf{e_s} = \mathsf{e} = \infty$. Thus, no remaining object remains after applying these functions.

3.1.1 Eliminating Functions of the First Type

We first explain eliminating functions of the first type. These functions eliminate more objects of $N(I)$ by setting e equal to a value a, $a > \mathsf{e}$. In order to increase e, we first prove that there is no value b in $\mathit{RootValueSet}(N(I))$ such that $\mathsf{e} \leq b < a$. Then since for

Figure 3.2 An eliminating configuration



members b of *RootValueSet*$(N(I))$ less than e an object of value b has already been added to the partial solution, we can claim that for members b of *RootValueSet*$(N(I))$ less than a an object of value b has already been added to the partial solution and so we can set $\mathsf{e} := a$. To prove that there is no value b in *RootValueSet*$(N(I))$ such that $\mathsf{e} \leq b < a$, we find a sub-union tree U of $N(Q)$ such that no leaf of U has an object of value b, $\mathsf{e} \leq b < a$. Then Lemma 2.22 guarantees that no value in b, $\mathsf{e} \leq b < a$, is in *RootValueSet*$(N(I))$.

We now use an example to clarify the idea. Consider the eliminating configuration of Figure 3.2(a) and the sub-union tree T of $N(Q)$ in the figure. As can be seen, the smallest object of value at least e in leaves of T is of value 11. Thus, there is no value b, $\mathsf{e} \leq b < 11$ in *RootValueSet*$(N(I))$ as no leaf of U has an object with any such value. Thus, in addition to objects of $N(Q)$ with values less than e, objects of $N(Q)$ with values at least e and less than 11 can be ignored. So, we set e equal to 11. Figure 3.2(b) demonstrates the eliminating configuration after updating the value of e. As can be seen, since the definition of an object of a speedy leaf being eliminated does not depend on e, the set of eliminated objects of the speedy leaf in the figure does not change.

Generally, when applying an eliminating function of the first type, a sub-union tree U

of $N(Q)$ is considered and e is increased in the following way. We consider the minimum value a of any remaining object of leaves of U. For every value b, $\mathsf{e} \leq b < a$, no leaf of U has an object of value b since by Definition 3.3 an object of value b is remaining (as $\mathsf{e} \leq b$) and the value of every remaining object of a leaf of U is at least a. Thus, according to Lemma 2.22, for every b, $\mathsf{e} \leq b < a$, b is not in $\mathit{RootValueSet}(N(I))$ as there is no object of value b in any leaf of U. Hence objects of $N(I)$ with values less than a can be ignored. We set e equal to a.

In order to define eliminating functions of the first type, we first define a function returning the minimum value of any remaining object of leaves of a given sub-union tree. Recall from Lemma 3.5 that the smallest remaining object of a non-speedy leaf can be found using the function *find*.

Definition 3.7 *Given an eliminating configuration C of an instance $I = (Q, \omega, \mu)$ and a sub-union tree T of $N(Q)$, $\mathrm{hmin}(\mu, C, T)$ is defined as ∞ if no leaf l of T has a C-remaining object; otherwise $\mathrm{hmin}(\mu, C, T)$ equals the minimum of $\mu(\mathit{find}(I, l, \mathsf{e}))$ over all leaves l of T with $\mathit{find}(I, l, \mathsf{e}) \neq \mathrm{END}$.*

The next lemma shows how one can evaluate the minimum of values of all remaining objects of leaves of a given sub-union tree using the function *hmin*.

Lemma 3.10 *Consider a valid eliminating configuration C of an instance $I = (Q, \omega, \mu)$ and a sub-union tree T of $N(Q)$. If a leaf of T has a C-remaining object, $\mathrm{hmin}(\mu, C, T)$ is the minimum value of any C-remaining object of any leaf of T.*

Proof. We prove that for every C-eliminated object o of every leaf of T, $\mathit{hmin}(\mu, C, T) \leq \mu(o)$ and then we conclude that $\mathit{hmin}(\mu, C, T) \leq b$ where b is the minimum value of any C-remaining object of any leaf of T. Then, to complete the proof of the lemma, we show $b \leq \mathit{hmin}(\mu, C, T)$ by proving $\mathit{hmin}(\mu, C, T)$ is the value of a C-remaining object in $\omega(l)$, for a leaf l of T.

First considering a leaf v of T and a C-remaining object o of $\omega(v)$, we prove that $\mu(o) \geq \mathit{hmin}(\mu, C, T)$. Since T is a sub-union tree of $N(Q)$, v is a leaf of $N(Q)$ and hence, v is not a speedy leaf. Also, due to the existence of the C-remaining object o, there is at least one C-remaining object in $\omega(v)$. Therefore, by Lemma 3.5 $\mathit{find}(I, v, \mathsf{e})$ is

a C-remaining object of $\omega(v)$ of minimum value and thus,

$$\mathit{find}(I, v, \mathsf{e}) \neq \text{END}. \tag{3.1}$$

Now o is a C-remaining object of $\omega(v)$ and $\mathit{find}(I, v, \mathsf{e})$ is a C-remaining object of $\omega(v)$ with the minimum value. Hence,

$$\mu(\mathit{find}(I, v, \mathsf{e})) \leq \mu(o). \tag{3.2}$$

Also, as v is a leaf of T and o is a C-remaining object of $\omega(v)$, by Definition 3.7 $\mathit{hmin}(\mu, C, T)$ is the minimum of $\mu(\mathit{find}(I, u, \mathsf{e}))$ over all leaves u of T with $\mathit{find}(I, u, \mathsf{e}) \neq \text{END}$. So, as due to Equation 3.1 $\mathit{find}(I, v, \mathsf{e}) \neq \text{END}$,

$$\mathit{hmin}(\mu, C, T) \leq \mu(\mathit{find}(I, v, \mathsf{e})). \tag{3.3}$$

Due to Equations 3.2 and 3.3, $\mathit{hmin}(\mu, C, T) \leq \mu(o)$. Therefore, as o has been selected as an arbitrary C-remaining object of leaves of T, the value of every C-remaining object of each leaf of T is at least $\mathit{hmin}(\mu, C, T)$. Thus,

$$\mathit{hmin}(\mu, C, T) \leq b \tag{3.4}$$

as b is the value of a C-remaining object of a leaf of T by assumption.

Next we prove that $\mathit{hmin}(\mu, C, T)$ is the value of a C-remaining object and then we conclude that $b \geq \mathit{hmin}(\mu, C, T)$. As by assumption there is a C-remaining object in a leaf of T, by Definition 3.7 $\mathit{hmin}(\mu, C, T) = \mu(\mathit{find}(I, l, \mathsf{e}))$, for a leaf l of T with $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$. Since l is a leaf of a sub-union tree of $N(Q)$, l is in $N(Q)$ and hence, l is not a speedy leaf. So as $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$, $\omega(l)$ has a C-remaining object and $\mathit{find}(I, l, \mathsf{e})$ is the smallest C-remaining object of $\omega(l)$ by Lemma 3.5. But we chose l such that $\mathit{hmin}(\mu, C, T) = \mu(\mathit{find}(I, l, \mathsf{e}))$. Therefore, $\mathit{hmin}(\mu, C, T)$ is the value of the smallest C-remaining object of $\omega(l)$. Hence, $b \leq \mathit{hmin}(\mu, C, T)$ as b is the minimum value of any C-remaining object of any leaf of T. Considering this result together with Equation 3.4, we obtain the equality $b = \mathit{hmin}(\mu, C, T)$ as claimed by the lemma. □

Now we can define the first type of eliminating function using the function *hmin*.

Definition 3.8 *Consider an instance $I = (Q, \omega, \mu)$ and suppose T is a sub-union tree of $N(Q)$. Given a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for I, we define $\pi_T^-(\mu, C) = (\mathsf{e}', \mathsf{e_s}', \Phi)$ where $\mathsf{e}' = hmin(\mu, C, T)$. Also, $\mathsf{e_s}' = \mathsf{e_s}$ if Q has a speedy leaf; otherwise $\mathsf{e_s}' = \mathsf{e}'$.*

Next we prove an eliminating function of the first type never decreases the e-value of an eliminating configuration.

Lemma 3.11 *Given an eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ of an instance $I = (Q, \omega, \mu)$ and a sub-union tree T of $N(Q)$, the e-value of $\pi_T^-(\mu, C)$ is at least as large as e.*

Proof. Defining e' to be the e-value of $\pi_T^-(\mu, C)$, we prove the lemma in each of the two cases $\mathsf{e}' = \infty$ and $\mathsf{e}' \neq \infty$ separately. In the first case, the correctness of the lemma is trivial as ∞ is greater than every member of $\mathbb{V}^+$. For $\mathsf{e}' \neq \infty$, by Definition 3.8, $\mathsf{e}' = hmin(\mu, C, T)$ and hence $hmin(\mu, C, T) \neq \infty$. By Definition 3.7, $hmin(\mu, C, T)$ is the value of a C-remaining object o of a leaf l of T and thus $\mu(o) = \mathsf{e}'$. The leaf l is in $N(Q)$ as it is in a sub-union tree of $N(Q)$. Thus, the value of every C-remaining object of l, including o, is at least e (Definition 3.3) Hence, $\mathsf{e}' = hmin(\mu, C, T) = \mu(o) \geq \mathsf{e}$. □

The next lemma shows that the result of applying an eliminating function to a valid eliminating configuration is a valid eliminating configuration.

Lemma 3.12 *Suppose $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ is a valid eliminating configuration for an instance $I = (Q, \omega, \mu)$ and T is a sub-union tree of $N(Q)$. The $\pi_T^-(\mu, C)$ is valid eliminating configuration.*

Proof. Defining $C' = (\mathsf{e}', \mathsf{e_s}', \Phi)$ as the result of $\pi_T^-(\mu, C)$, we prove the lemma by first showing that C' is an eliminating configuration and then by proving that C' satisfies each of the three conditions of Definition 3.5, separately. We first prove the second condition, then the first condition, and finally the third condition of Definition 3.5 for C'.

We now prove that C' is an eliminating configuration by considering two cases based on Q having a speedy leaf or not. If Q has no speedy leaf, by Definition 3.8 $\mathsf{e_s}' = \mathsf{e}'$ and so by Definition 3.2 C' is an eliminating configuration. Now suppose Q has a speedy leaf.

Then, $\mathsf{e_s}' = \mathsf{e_s}$ and thus as $\mathsf{e} \leq \mathsf{e}'$ (Lemma 3.11) and $\mathsf{e_s} \leq \mathsf{e}$ (Definition 3.2 for C), $\mathsf{e_s}' \leq \mathsf{e}'$. Hence, by Definition 3.2 C' is an eliminating configuration.

We now justify the correctness of the second condition for C'. We first prove that the statements $a < \mathsf{e}'$ and $a < \mathsf{e}$ are equivalent, for any value a in $\mathit{RootValueSet}(N(I))$. Then, as we will show, since the second condition of Definition 3.5 holds for C, that condition also holds for C'.

Now we consider a value a in $\mathit{RootValueSet}(N(I))$ and we prove that $a < \mathsf{e}'$ if and only if $a < \mathsf{e}$. The correctness of the "if" part follows from the fact that by Lemma 3.11 $\mathsf{e} \leq \mathsf{e}'$. Now to prove the "only if" part, suppose $a < \mathsf{e}'$. By definition 3.8, $\mathsf{e}' = \mathit{hmin}(\mu, C, T)$. Thus as $a < \mathsf{e}'$,

$$a < \mathit{hmin}(\mu, C, T). \tag{3.5}$$

According to Lemma 2.22, there exists a leaf l of T such that $\omega(l)$ has an object o of value a as a is selected as a value in $\mathit{RootValueSet}(N(I))$. Due to Lemma 3.10, when there is a C-remaining object in a leaf of T, $\mathit{hmin}(\mu, C, T)$ is the value of a C-remaining object of leaves of T of the minimum value. Hence, if there is any C-remaining object in leaves of T, the value of every C-remaining object of leaves of T is at least $\mathit{hmin}(\mu, C, T)$. So, o can not be C-remaining because o is in $\omega(l)$, l is a leaf of T, and $\mu(o) = a < \mathit{hmin}(\mu, C, T)$ by Equation 3.5. Also, as l is a leaf of a sub-union tree of $N(Q)$, it is a leaf of $N(Q)$, and hence, l is not a speedy leaf. Thus, according to Definition 3.3, $a = \mu(o) < \mathsf{e}$ as o is a C-eliminated object of a non-speedy leaf. Therefore, the "only if" part is also true.

Now using the above results we prove that C' meets the second condition of Definition 3.5. Due to that condition for C, there is an object of value a in $N(I)$ if and only if $a < \mathsf{e}$, for every a in the value set the root in $N(I)$. Also, we proved $a < \mathsf{e}'$ is equivalent to $a < \mathsf{e}$, for every a in $\mathit{RootValueSet}(N(I))$. Considering these facts together leads us to the fact that given an a in $\mathit{RootValueSet}(N(I))$, there is an object of value a in $N(I)$ if and only if $a < \mathsf{e}'$. Hence, C' satisfies the second condition in Definition 3.5.

Next we consider the first condition of Definition 3.5. If Q has no speedy leaf, by definition $I = N(I)$ and by Definition 3.2 $\mathsf{e_s}' = \mathsf{e}'$. Thus in this case the first condition of Definition 3.5 is equivalent to the second condition of Definition 3.5 which was proved above. In addition, if Q has a speedy leaf, $\mathsf{e_s}' = \mathsf{e_s}$, according to Definition 3.8, and so the first condition of Definition 3.5 for C' is equivalent to the same condition for C. Hence, as

C satisfies all conditions of Definition 3.5, in this case also the first condition is satisfied by C'.

Finally, the last condition of Definition 3.5 holds for C' since according the same condition for C, Φ is a partial solution. Thus, $C' = \pi_T^-(\mu, C)$ meets all three conditions of Definition 3.5 and hence, $\pi_T^-(\mu, C)$ is valid. □

3.1.2 Eliminating Functions of the Second Type

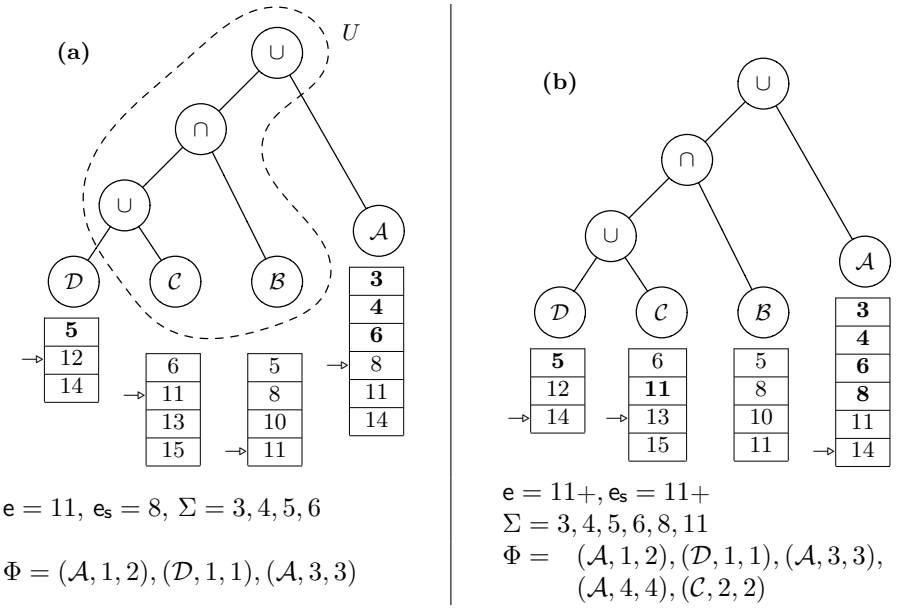
This section introduces eliminating functions of the second type which are designed for the situation in which e is in the value set of the root. In this situation, by Lemma 2.22 there is a sub-intersection tree T of the query tree such that every leaf of T has an object of value e. In such cases, we add an object of value e to our partial solution and we increase e to e_+ so that objects of value e are eliminated.

Now we discuss details that should be considered before adding an object of value e to our partial solution. As we are adding objects to the partial solution in order of value, before adding an object of value e, we have to ensure that for every member a of the value set of the root such that $a < \mathsf{e}$, there is an object of value a in the partial solution. According to the first condition of Definition 3.5, for values a less than $\mathsf{e_s}$ this is true but for values a, $\mathsf{e_s} \leq a < \mathsf{e}$, there is no object of value a in the current partial solution.

Now, defining $\mathcal{S}$ as the set of all values a in $\mathit{RootValueSet}(I)$ with $\mathsf{e_s} \leq a < \mathsf{e}$, we explain how we add objects with values in $\mathcal{S}$ to our partial solution before adding an object of value e. For every a in $\mathcal{S}$, $a \geq \mathsf{e_s}$, but $a \not\geq \mathsf{e}$ by definition of $\mathcal{S}$. So, a is not in $\mathit{RootValueSet}(N(I))$, due to Lemma 3.8, for every member a of $\mathcal{S}$. Then, as members of $\mathcal{S}$ are in $\mathit{RootValueSet}(I)$, it follows from Observation 2.32 that every value in $\mathcal{S}$ is the value of an object in a speedy leaf. Thus, our approach is to add objects of speedy leaves (if there is any speedy leaf) with values at least $\mathsf{e_s}$ and less than e to our partial solution, before adding an object of value e.

We now present an example to clarify the method. Consider the eliminating configuration of Figure 3.3(a). As can be seen, U is a sub-intersection tree of the query tree in which each leaf has an object of value $\mathsf{e} = 11$. So we can add an object of value 11 to our partial solution. As Figure 3.3(b) demonstrates, before adding the object of $\mathcal{C}$ of value 11,

Figure 3.3 An application of a second type eliminating function



we add the only object with a value at least $\mathsf{e_s} = 8$ and less than 11 to the expansion Σ of Φ, namely the object of value 8. After that, we add an object of value 11 to Σ and set $\mathsf{e} = \mathsf{e_s} = 11+$ so that objects with values at most 11 of the instance are eliminated.

We first define a function Φ_{sl} that, given a value b such that $b \leq \mathsf{e}$, adds objects of speedy leaves with values a, $\mathsf{e_s} \leq a \leq b$ to our partial solution. Applying this function for $b = \mathsf{e}$, we can add objects of speedy leaves with values in the set $\mathcal{S}$, defined above, to our partial solution. Having defined this function in this more general form, we will be able to use it in designing our algorithm for the situation in which the algorithm has not found all objects of speedy leaves with values less than e and it is going to add just those objects of speedy leaves that have values less than b, for some b, $\mathsf{e_s} \leq b \leq \mathsf{e}$.

Since eliminating functions are defined to be applied to instances in which the query tree has at most one speedy leaf (Definition 3.6), it is enough to define the function Φ_{sl} so that it only works for this kind of instance. Now let us explain how this function is defined. As is shown in the next lemma, the set of objects of the speedy leaf (if it exists) with values a, $\mathsf{e_s} \leq a < b$, is the set of objects with an index i in $\omega(sl(Q))$ such

that $pos(I, sl(Q), \mathsf{e_s}) \leq i \leq pos(I, sl(Q), b) - 1$. Because of this, Φ_{sl} appends the output item $(sl(Q), pos(I, sl(Q), \mathsf{e_s}), pos(I, sl(Q), b) - 1)$ to the output of the current eliminating configuration. In this way, as we will show, an object of value a is added to our partial solution, for each a in $\mathcal{S}$.

Lemma 3.13 *Suppose* $(\mathsf{e}, \mathsf{e_s}, \Phi)$ *is an eliminating configuration for an instance* $I = (Q, \omega, \mu)$, Q *has exactly one speedy leaf,* $\omega(sl(Q)) = o_1, \ldots, o_k$, *and* b *is a member of* $\mathbb{V}^+$ *such that* $\mathsf{e_s} \leq b$. *For every object* o_i *of* $\omega(sl(Q))$, $pos(I, sl(Q), \mathsf{e_s}) \leq i \leq pos(I, sl(Q), b) - 1$ *if and only if* $\mathsf{e_s} \leq \mu(o_i) < b$.

Proof. This lemma is proved by showing the correctness of the following claims for every object o_i of $sl(Q)$.

1. $i \leq pos(I, sl(Q), b) - 1$ if and only if $\mu(o_i) < b$.
2. $pos(I, sl(Q), \mathsf{e_s}) \leq i$ if and only if $\mathsf{e_s} \leq \mu(o_i)$.

We now prove these two claims using Lemma 3.3. Applying Lemma 3.3 for $l = sl(Q)$ results in the fact that for any integer i, $1 \leq i \leq k$, $pos(I, sl(Q), b) \not\leq i$ if and only if $b \not\leq \mu(o_i)$, that is, $i < pos(I, sl(Q), b)$ if and only if $\mu(o_i) < b$. Since i and the result of the function pos are integers, we can conclude that $i \leq pos(I, sl(Q), \mathsf{e_s}) - 1$ if and only if $\mu(o_i) < b$ and hence the first claim is proved. The correctness of the second claim follows immediately from Lemma 3.3. □

Before defining the function Φ_{sl}, we explain when Φ_{sl} adds a new output item to the output. Given an instance $I = (Q, \omega, \mu)$ with a speedy leaf where $\omega(sl) = o_1, \ldots, o_k$, using Lemma 3.13 it can be proved that the speedy leaf has an object o_i satisfying $\mathsf{e_s} \leq \mu(o_i) < b$ if and only if there is an integer i between $pos(I, sl(Q), \mathsf{e_s})$ and $pos(I, sl(Q), b) - 1$. Hence, as can be seen in the next definition, Φ_{sl} adds a new output item only if $pos(I, sl(Q), \mathsf{e_s}) \neq pos(I, sl(Q), b)$ and thus there is an integer i between $pos(I, sl(Q), \mathsf{e_s})$ and $pos(I, sl(Q), b) - 1$, inclusive. Recall the operator "$\odot$" from Definition 2.2.

Definition 3.9 *Given an eliminating configuration* $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ *for an instance* (Q, ω, μ) *in which* Q *has at most one speedy leaf and a value* b *in* $\mathbb{V}^+$, $\mathsf{e_s} \leq b \leq \mathsf{e}$, *we define*

the sequence $\Phi_{sl}(C, \mu, b)$ *of output items to be equal to* Φ *if* Q *has no speedy leaf or* $pos(I, sl(Q), \mathsf{e_s}) = pos(I, sl(Q), b)$ *and we define it to be equal to*

$$\Phi \odot (sl(Q), pos(I, sl(Q), \mathsf{e_s}), pos(I, sl(Q), b) - 1)\,,$$

otherwise.

In the next lemma we show that the definition of Φ_{sl} matches our intention in defining this function, that is, if we apply the function Φ_{sl} on an eliminating configuration C and a value b then Φ_{sl} adds all objects of the speedy leaf (if it exists) with values at least the $\mathsf{e_s}$-value of C and less than b to the output of C.

Lemma 3.14 *Consider a valid eliminating configuration* $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ *of an instance* (Q, ω, μ) *and a value* b *in* $\mathbb{V}^+$ *such that* Q *has exactly one speedy leaf and* $\mathsf{e_s} \leq b \leq \mathsf{e}$*. We define the subsequence* Σ *of* $\omega(sl(Q))$ *consisting of all objects* o *of* $\omega(sl(Q))$ *satisfying* $\mathsf{e_s} \leq \mu(o) < b$*. The expansion of* $\Phi_{sl}(C, \mu, b)$ *is the concatenation of the expansion of* Φ *and* Σ*.*

Proof. We prove the lemma for each of the two cases $pos(I, sl(Q), \mathsf{e_s}) = pos(I, sl(Q), b)$ and $pos(I, sl(Q), \mathsf{e_s}) \neq pos(I, sl(Q), b)$, separately. In the first case we show that Σ is empty and so the lemma is true as the function returns the input Φ without any change. In the second case using Lemma 3.13 we will prove that the set of objects that are appended to the expansion of Φ due to the last output item of $\Phi_{sl}(C, \mu, b)$ is the same as the set of objects in Σ. We will use $o_1, \ldots, o_k$ to denote $\omega(sl(Q))$.

First we consider the case $pos(I, sl(Q), \mathsf{e_s}) = pos(I, sl(Q), b)$. In this cases there is no i such that $pos(I, sl(Q), \mathsf{e_s}) \leq i \leq pos(I, sl(Q), b) - 1$ since $pos(I, sl(Q), \mathsf{e_s}) \not\leq pos(I, sl(Q), b) - 1$. Hence, by Lemma 3.13 o_i does not satisfy the inequation $\mathsf{e_s} \leq \mu(o_i) < b$ and thus o_i is not in Σ, for every i, $1 \leq i \leq k$. Therefore, Σ is the empty sequence. Moreover, since by Definition 3.9 in this case $\Phi_{sl}(C, \mu, b) = \Phi$, the expansion of $\Phi_{sl}(C, \mu, b)$ equals the expansion of Φ. Thus, the expansion of $\Phi_{sl}(C, \mu, b)$ is the concatenation of the expansion of Φ and the empty sequence Σ. Therefore, the lemma is true when $pos(I, sl(Q), \mathsf{e_s}) = pos(I, sl(Q), b)$.

Next we consider the case $pos(I, sl(Q), \mathsf{e_s}) \neq pos(I, sl(Q), b)$. In this case, by Definition 3.9, $\Phi_{sl}(C, \mu, b) = \Phi \odot (sl(Q), m, n)$ where $m = pos(I, sl(Q), \mathsf{e_s})$ and $n = pos(I, sl(Q), b) -$

1. Thus, by Observation 2.3 the expansion of $\Phi_{sl}(C, \mu, b)$ is the concatenation of the expansion of Φ and the sequence

$$\omega(sl(Q))[m], \omega(sl(Q))[m+1], \ldots, \omega(sl(Q))[n] = o_m, o_{m+1}, \ldots, o_n. \tag{3.6}$$

So, in order to prove the correctness of the lemma, it suffices to show that $\Sigma = o_m, o_{m+1}, \ldots, o_n$. Hence, as Σ and $o_m, \ldots, o_n$ both are subsequences of $\omega(sl(Q))$, if we prove the set of objects in Σ is the same as the set of objects in $o_m, \ldots, o_n$, the lemma is proved.

We now complete the proof of the lemma by considering an object o_j of $sl(Q)$ and by proving that o_j is in Σ if and only if o_j is in the sequence $o_m, \ldots, o_n$. Since m and n are defined as $pos(I, sl(Q), \mathsf{e_s})$ and $pos(I, sl(Q), b) - 1$, respectively, an object o_j of $sl(Q)$ is in $o_m, \ldots, o_n$ if and only if $pos(I, sl(Q), \mathsf{e_s}) \leq j \leq pos(I, sl(Q), b) - 1$. By Lemma 3.13, $pos(I, sl(Q), \mathsf{e_s}) \leq j \leq pos(I, sl(Q), b) - 1$ if and only if $\mathsf{e_s} \leq \mu(o_j) < b$, for every o_j, $1 \leq j \leq k$. Moreover, by the definition of Σ, o_j is in Σ if and only if $\mathsf{e_s} \leq \mu(o_j) < b$, for all objects o_j of $sl(Q)$. Hence an object o_j of $sl(Q)$ is in Σ if and only if o_j is in $o_m, \ldots, o_n$. This proves the lemma as explained. □

We now define the second type of eliminating function. This function is defined such that it first adds objects with values at least $\mathsf{e_s}$ and less than e to the expansion of the output using the function Φ_{sl}. After that, an object of value e is added to the end of the expansion of the output.

To realize how an object of value e is found, recall that when an eliminating function of the second type is being applied, the value e is in the value set of the root. Hence when defining this function we may use the fact that by Lemma 2.22 there is a sub-intersection tree T of the query tree such that each leaf of T has an object of value e. The function considers a leaf l of T and adds the output item $(l, pos(I, l, \mathsf{e}), pos(I, l, \mathsf{e}))$ to the output. Since l has an object of value e, by Lemma 3.2 $find(I, l, a)$ is an object of value e. Thus, as by Definition 3.4 $find(I, l, a) = \omega(l)[pos(I, l, a)]$, $pos(I, l, a)$ is the index of an object of value e in $\omega(l)$. Hence, by adding the output item $(l, pos(I, l, \mathsf{e}), pos(I, l, \mathsf{e}))$ to the output, an object of value e is appended to the expansion of the output, as we will prove more precisely in Lemma 3.20.

Before presenting the definition of the eliminating functions of the second type, it is

worth mentioning that an eliminating function is not necessarily defined on the whole of its domain.

Definition 3.10 *Consider an instance $I = (Q, \omega, \mu)$ and suppose T is a sub-intersection tree of Q. Given a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for I, $\pi_T^+(\mu, C)$ is defined if e is a main value and $\omega(l)$ contains an object of value e, for every leaf l of T. Then we define $\pi_T^+(\mu, C) = (\mathsf{e}_+, \mathsf{e}_+, \Phi')$ where $\Phi' = \Phi_{sl}(C, \mu, \mathsf{e}) \odot (l_T, pos(I, l_T, \mathsf{e}), pos(I, l_T, \mathsf{e}))$ and l_T is a fixed arbitrary leaf of T depending only on T.*

In the rest of the section we show that after applying an eliminating function on a valid eliminating configuration we get a valid eliminating configuration. Since by Definition 3.10 the output Φ of the eliminating configuration resulting from applying an eliminating function of the second type has the result of the function Φ_{sl} as a subsequence, we first show that the sequence resulting from applying Φ_{sl} is a partial solution. This will help to prove that Φ is a partial solution. To prove that the expansion of the result of the function Φ_{sl} is a partial solution, we first show that the concatenation of two partial solutions satisfying a certain property is a partial solution and then, using Lemma 3.14, in Lemma 3.16 we prove that the expansion of the result of the function Φ_{sl} is such a concatenation.

Lemma 3.15 *Consider two partial solutions Σ_1 and Σ_2 to an instance $I = (Q, \omega, \mu)$ such that the value of every object in Σ_2 is greater than the value of every object in Σ_1. Then, the sequence $\Sigma_1 \circ \Sigma_2$ is a partial solution to I.*

Proof. By Definition 3.1 it suffices to prove the following claims: first, the value of each object in $\Sigma_1 \circ \Sigma_2$ is in the value set of the root and second, $\mu(p) < \mu(q)$, for every two consecutive objects p and q of $\Sigma_1 \circ \Sigma_2$. The first claim is true since every object o of $\Sigma_1 \circ \Sigma_2$ is in Σ_1 or in Σ_2 and so it is in the value set of the root as both Σ_1 and Σ_2 are partial solutions to I. We now prove the second claim. If p and q both are in Σ_1 or both are in Σ_2 by Definition 3.1 for Σ_1 and Σ_2 $\mu(p) < \mu(q)$; otherwise p is an object of Σ_1 and q is an object of q and hence by the assumption of the lemma $\mu(p) < \mu(q)$. So, both properties required for $\Sigma_1 \circ \Sigma_2$ to be a partial solution are satisfied. □

Lemma 3.16 *Given a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for an instance $I = (Q, \omega, \mu)$ and a value b in $\mathbb{V}^+$ such that Q has exactly one speedy leaf and $\mathsf{e_s} \leq b$, the expansion of $\Phi_{sl}(C, \mu, b)$ is a partial solution to I.*

Proof. Since by Lemma 3.14 the expansion of $\Phi_{sl}(C, \mu, b)$ is the concatenation of the expansion Σ_Φ of Φ and a sequence Σ, by Lemma 3.15 it suffices to prove that each of the sequences Σ_Φ and Σ is a partial solution and values of all objects in Σ are greater than values of all objects in Σ_Φ. We first will argue that as C is valid, Σ_Φ is a partial solution and then we will use the definition of Σ to prove that Σ is a partial solution too. After that, we will show that values of objects in Σ are at least $\mathsf{e_s}$ and values of objects in Σ_Φ are less than $\mathsf{e_s}$ and hence values of objects of Σ are greater than values of all objects in Σ_Φ. We now discuss each part of the proof in detail.

First we prove that Σ is a partial solution to I by showing that Σ satisfies the two conditions of Definition 3.1. Let us first identify objects appearing in Σ. As mentioned, the expansion of $\Phi_{sl}(C, \mu, b)$ is the concatenation of the expansion of Φ and Σ where Σ is the subsequence of $\omega(sl(Q))$ consisting of objects of $sl(Q)$ with values at least $\mathsf{e_s}$ and less than b (Lemma 3.14). As a result, every object of Σ is an object of the speedy leaf and so its value is in the value set of the speedy leaf. Furthermore, by Observation 2.32 the value set of every speedy leaf is a subset of the value set of the root. Therefore, values of all objects of Σ are in the value set of the root. Also, since Σ is a subsequence of $\omega(sl(Q))$, every two consecutive objects p and q of Σ are consecutive objects of $\omega(sl(Q))$. Hence, as by Definition 2.6 μ is ordered, $\mu(p) < \mu(q)$ by Definition 2.5, for every two consecutive objects p and q of Σ. Thus, Σ if a partial solution to I as Σ satisfies both conditions of Definition 3.1.

Next we show that Σ_Φ is a partial solution. By the assumption of the lemma C is valid. Moreover, Σ_Φ is defined as the expansion of the output of C. So, the correctness of the claim that Σ_Φ is a partial solution follows from the third condition of Definition 3.5 for C.

Finally, we prove that values of all of the objects of Σ are greater than values of objects in Σ_Φ. As we proved, Σ is the subsequence of $\omega(sl(Q))$ consisting of objects of $sl(Q)$ with values at least $\mathsf{e_s}$ and less than b. Also, as Σ_Φ is the expansion of the output of the valid eliminating configuration C, values of all of its objects are less than $\mathsf{e_s}$ by the first condition of Definition 3.5. Hence, for every object p of Σ_Φ and object q of Σ we have

$\mu(p) < \mathsf{e_s} \leq \mu(q)$. Therefore, as explained in the very beginning of the proof of the lemma, the expansion of $\Phi_{sl}(C, \mu, b)$ is a partial solution to I by Lemma 3.15. □

Given an instance $I = (Q, \omega, \mu)$ and a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for I, in order to prove that the eliminating configuration C' resulting from applying an eliminating function of the second type on C is valid, we need to prove that the output Φ' of C' is a partial solution to I because of the third condition in Definition 3.5. To show that Φ' is a partial solution, in the next lemma we prove that Φ' is the concatenation of the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$ with a sequence containing just one object of value e and after that, in Lemma 3.18 we prove that values of all objects in the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$ are less than e. This will enable us to use Lemma 3.15 to prove that Φ' is a partial solution to I.

Lemma 3.17 *Consider an instance $I = (Q, \omega, \mu)$, a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for I, and a sub-intersection tree T of Q such that $\pi^+_T(\mu, C)$ is defined. The expansion of the output Φ_π of $\pi^+_T(\mu, C)$ is $\Sigma_{sl} \odot p$ where Σ_{sl} is the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$ and p is an object of value e.*

Proof. We first show that the expansion of Φ_π is equal to $\Sigma_{sl} \odot \mathit{find}(I, l_T, \mathsf{e})$ where l_T is the leaf of T defined in Definition 3.10 and then we prove that $\mathit{find}(I, l_T, \mathsf{e})$ is an object of value e. According to Definition 3.10, $\Phi_\pi = \Phi_{sl}(C, \mu, \mathsf{e}) \odot (l_T, m, m)$ where $m = pos(I, l_T, \mathsf{e})$. So, by Observation 2.3, the expansion of Φ_π is $\Sigma_2 \circ \Sigma_1$ where Σ_2 is the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$ and Σ_1 contains just one object $\omega(l_T)[m]$. Hence, by Observation 2.1 the expansion of Φ_π equals $\Sigma_2 \odot \omega(l_T)[m]$. As Σ_2 and Σ_{sl} both are expansions of $\Phi_{sl}(C, \mu, \mathsf{e})$, $\Sigma_2 = \Sigma_{sl}$. Furthermore, $\omega(l_T)[m] = \omega(l_T)[pos(I, l_T, \mathsf{e})]$ by the definition of m. According to Definition 3.4, $\omega(l_T)[pos(I, l_T, \mathsf{e})] = \mathit{find}(I, l_T, \mathsf{e})$. Hence, the expansion of Φ_π equals $\Sigma_2 \odot \omega(l_T)[m] = \Sigma_{sl} \odot \omega(l_T)[pos(I, l_T, \mathsf{e})] = \Sigma_{sl} \odot \mathit{find}(I, l_T, \mathsf{e})$.

Now we complete the proof of the lemma by showing that $\mu(\mathit{find}(I, l_T, \mathsf{e})) = \mathsf{e}$. Since by assumption $\pi^+_T(\mu, C)$ is defined, by Definition 3.10 every leaf of T has an object of value e. So, as l_T is a leaf of e, $\omega(l_T)$ has an object of value e. Consequently, the value of $\mathit{find}(I, l_T, \mathsf{e})$ equals e by Lemma 3.2. Thus, we proved the expansion of Φ_π is $\Sigma_{sl} \odot \mathit{find}(I, l_T, \mathsf{e})$ and the value of $\mathit{find}(I, l_T, \mathsf{e})$ is e. □

Lemma 3.18 *Given an eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ of an instance $I = (Q, \omega, \mu)$ and a value b in $\mathbb{V}^+$ such that Q has at most one speedy leaf and $\mathsf{e_s} \leq b \leq \mathsf{e}$, the value of every object in the expansion of $\Phi_{sl}(C, \mu, b)$ is less than b.*

Proof. We first show that the value of each object in the expansion Σ_Φ of Φ is less than b. Then we consider two cases based on Q having a speedy leaf or not and in each case we complete the proof of the lemma separately.

The correctness of the claim that values of object in Σ_Φ are less than b follows from the two following facts. First, by assumption Σ_Φ is the expansion of the output Φ of C and thus by the first condition of Definition 3.5 for C the value of every object in Σ_Φ is less than $\mathsf{e_s}$. Second, by the assumption of the lemma $\mathsf{e_s} \leq b$.

Now we prove the lemma using the fact that we proved above. First let us consider the case in which Q has no speedy leaf. Then, $\Phi_{sl}(C, \mu, \mathsf{e}) = \Phi$ by Definition 3.9 and so Σ_Φ is the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$ as Σ_Φ is the expansion of Φ. Thus, the correctness of the lemma results from the fact that the value of every object in Σ_Φ is less than b, as we proved. Next we consider the other case, in which Q has a speedy leaf. Then, by Lemma 3.14, the expansion of $\Phi_{sl}(C, \mu, b)$ equals $\Sigma_\Phi \circ \Sigma$ where Σ is a subsequence of $\omega(sl(Q))$ in which the value of every object is less than b. Also, we showed that values of object in Σ_Φ are less than b. Hence, the value of every object of the expansion of $\Phi_{sl}(C, \mu, b)$ is less than b. □

Given an eliminating configuration C' resulting from applying an eliminating function of the second type on a valid eliminating configuration $(\mathsf{e}, \mathsf{e_s}, \Phi)$, a step toward proving that C' is valid is to prove that C' satisfies the first condition of Definition 3.8. By Definition 3.10, $C' = (\mathsf{e}{+}, \mathsf{e}{+}, \Phi')$, for some Φ', and by Lemma 3.17 the expansion of Φ' is evaluated by appending an object o of value e to the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$. To show the correctness of the first condition of Definition 3.5 for C', as the $\mathsf{e_s}$-value of C' is $\mathsf{e}{+}$, we have to show that there is an object of value a in the expansion of Φ', for every value a in the value set of the root less than $\mathsf{e}{+}$. Since every value a in the value set of the root is a member of $\mathbb{V}$ (Definition 2.9), a is a main value and hence if $a \leq \mathsf{e}{+}$ then a is either e or less than e. As o is an object of value e in the expansion of Φ', to prove the first condition of Definition 3.5, it suffices to prove that there is an object of value a in the expansion $\Phi_{sl}(C, \mu, \mathsf{e})$, for every value a in the value set of the root less than e. We can prove this claim by applying the

next lemma for $b = \mathsf{e}$.

Lemma 3.19 *Consider a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ of an instance $I = (Q, \omega, \mu)$ and values a and b in the value set of the root and in $\mathbb{V}^+$, respectively, such that Q has at most one speedy leaf and $a < b \leq \mathsf{e}$. There is an object of value a in the expansion of $\Phi_{sl}(C, \mu, b)$.*

Proof. We prove the lemma by considering several cases based on a being in the value set of the root in $N(I)$ or not and also on a being less than $\mathsf{e_s}$ or not. If a is in the value set of the root in $N(I)$ or $a < \mathsf{e_s}$ then we prove that there is an object of value a in the expansion of $\Phi_{sl}(C, \mu, b)$ by showing that there is one in the expansion of Φ. Then we will argue that if a is not in the value set of the root and $a \geq \mathsf{e_s}$, there is an object of value a in the expansion of $\Phi_{sl}(C, \mu, b)$ due to the last output item of $\Phi_{sl}(C, \mu, b)$.

Case 1: The first case is that a is in the value set of the root in $N(I)$. Since by the assumption of the lemma $a < \mathsf{e}$, considering the first condition of Definition 3.5 for C, we can conclude that there is an object o of value a in the expansion of Φ. Now we consider two subcases based on Q having a speedy leaf or not and in each subcase we prove that o is in the expansion of $\Phi_{sl}(C, \mu, b)$. If Q has a speedy leaf, $\Phi_{sl}(C, \mu, b) = \Phi$ by Definition 3.9 and thus o in the expansion of $\Phi_{sl}(C, \mu, b)$ as o is in the expansion of Φ. The second subcase is that Q has no speedy leaf and thus the expansion Σ_Φ of $\Phi_{sl}(C, \mu, b)$ is the concatenation of the expansion of Φ with another sequence, according to Lemma 3.14. Hence in this subcase the expansion of Φ is a subsequence of the expansion of $\Phi_{sl}(C, \mu, b)$. Therefore, o appears in the expansion of $\Phi_{sl}(C, \mu, b)$ as o appears in the expansion of Φ. So, in either case o is in the expansion of $\Phi_{sl}(C, \mu, b)$ while $\mu(o) = a$.

Case 2: The second case is that $a < \mathsf{e_s}$. Then, by the first condition of Definition 3.5 for C there is an object of value a in the expansion of Φ and so, there is one in the expansion of Σ_Φ by the same argument of the previous case.

Case 3: The last case is that a is not in the value set of the root in $N(I)$ and $a \geq \mathsf{e_s}$. Thus, since a is selected as a member of the value set of the root and here we are considering the case in which a is not in the value set of the root in $N(I)$, a is in the value set of a

speedy leaf by Observation 2.32. Hence, Q has a speedy leaf and a is the value of an object o of the speedy leaf. Since $\mu(o) = a$ and by the assumption of the lemma $a < b$ and also we are considering a case in which $\mathsf{e_s} \leq a$, $\mathsf{e_s} \leq a = \mu(o) < b$. Moreover by Lemma 3.14 the expansion of $\Phi_{sl}(C, \mu, b)$ is the concatenation of the expansion of Φ and the sequence Σ of all objects of the speedy leaf (if it exists) with values at least $\mathsf{e_s}$ and less than b. Hence Σ includes the object o as o is the object of the speedy leaf and we proved $\mathsf{e_s} \leq \mu(o) < b$. Therefore, o is in the expansion of $\Phi_{sl}(C, \mu, b)$ because the expansion of $\Phi_{sl}(C, \mu, b)$ is the concatenation of a sequence with Σ and o is in Σ. Thus, the expansion of $\Phi_{sl}(C, \mu, b)$ has an object of value $\mu(o) = a$.

In all cases we have now proved that an object of value a is in the expansion of $\Phi_{sl}(C, \mu, b)$. So the lemma is proved. □

Using lemmas that we have proved, now we can prove that after applying an eliminating configuration of the second type, the eliminating configuration remains valid. Note that if $\pi^+_T(\mu, C)$, for some μ, C, and T, is defined, by definition the e-value and the $\mathsf{e_s}$-value of $\pi^+_T(\mu, C)$ are the same and so by Definition 3.2 $\pi^+_T(\mu, C)$ is an eliminating configuration.

Lemma 3.20 *Suppose $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ is a valid eliminating configuration for an instance $I = (Q, \omega, \mu)$, T is a sub-intersection tree of Q, and Q has at most speedy leaf. If $\pi^+_T(\mu, C)$ is defined then $\pi^+_T(\mu, C)$ is a valid eliminating configuration.*

Proof. In order to prove that $C' = \pi^+_T(\mu, C)$ is valid, we first prove that the expansion Σ_Φ of the output of C is a concatenation of two partial solutions such that values of objects in the second sequence are greater than values of all objects in the first one. According to Lemma 3.15, these results leads us to the fact that Σ_Φ is a partial solution and so C' satisfies the third condition of Definition 3.5. Then after proving that C' satisfies the first condition of Definition 3.5, we show that the correctness of the condition one of Definition 3.5 for C' implies the correctness of the second condition of that definition for C'.

Before starting the argument let us explain some facts about C'. In the following, $(\Phi', \mathsf{e}', \mathsf{e_s}')$ equals C'. Hence, by Definition 3.10, $\mathsf{e}' = \mathsf{e_s}' = \mathsf{e}_+$. Furthermore, we will use the fact that by Lemma 3.17 the expansion Σ_Φ of Φ' is obtained by appending an object

o of value e to the very end of the expansion Σ_{sl} of $\Phi_{sl}(C, \mu, \mathsf{e})$, that is,

$$\Sigma_{\Phi} = \Sigma_{sl} \odot o. \tag{3.7}$$

Now we prove the three conditions of Definition 3.5 for C' in the order that we mentioned.

The third condition: Defining Γ to be the sequence containing just one object o, after proving the following three claims we can use Lemma 3.15 to prove that Σ_{Φ} is a partial solution. *Claim 1:* Σ_{Φ} is the concatenation of Σ_{sl} and Γ. *Claim 2:* Σ_{sl} and Γ are partial solutions to I. *Claim 3:* The value of the only object of Γ is greater than values of all objects in Σ_{sl}. The first claim is true as $\Sigma_{\Phi} = \Sigma_{sl} \odot o$ by Equation 3.7 and thus $\Sigma_{\Phi} = \Sigma_{sl} \circ \Gamma$ by Observation 2.1. We now prove the other two claims.

Claim 2 consists of two parts. The part stating that Σ_{sl} is a partial solution follows from Lemma 3.16. Now we prove the other part, claiming that Γ is a partial solution, by first showing that the value of the only object of Γ is in the value set of the root and then by concluding that Γ meets both conditions of Definition 3.1. By assumption the value of o equals e. Since by assumption $\pi^{+}_{T}(\mu, C)$ is defined, every leaf of the sub-intersection tree T has an object of value e by Definition 3.10. Thus, e is in the value set of the root by Lemma 2.22. Hence, $\mu(o)$ is in $\mathit{RootValueSet}(I)$ as $\mu(o) = \mathsf{e}$. Also o is the only object of Γ. Consequently, both conditions of Definition 3.1 are met and so Γ is a partial solution.

Now we prove Claim 3. Since Σ_{sl} is the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$, by Lemma 3.18 values of all objects of Σ_{sl} are less than e. Moreover, o is the only object of Γ and $\mu(o) = \mathsf{e}$. Thus, the value of each object of Γ is greater than the value of each object of Σ_{sl}.

Having proved the three above claims, it follows from Lemma 3.15 that Σ_{Φ} is partial solution to I. Therefore, the first condition is satisfied as Σ_{Φ} is the output of $\pi^{+}_{T}(\mu, C)$.

The first condition: To prove the first condition for C', we first prove that values of all objects of Σ_{Φ} are less than $\mathsf{e_s}'$ and then we show that, given a value a in the value set of the root, if $a < \mathsf{e_s}'$, a is the value of an object in Σ_{Φ}. Once we have proved these two claims, we can conclude that there is an object of value a in Σ_{Φ} if and only if $a < \mathsf{e_s}'$, for every a in the value set of the root. Since Σ_{Φ} is the expansion of Φ', this yields the fact that C' satisfies the first condition of Definition 3.5.

Now let us first prove that the value of each object in Σ_Φ is less than $\mathsf{e_s}'$. Since we proved Σ_Φ is a partial solution, by Definition 3.1 objects appear in order of value in Σ_Φ and hence the value of every object in Σ_Φ is at most the value of the last object in this sequence. Also, since $\Sigma_\Phi = \Sigma_{\mathsf{sl}} \odot o$ (Equation 3.7), o is the last object of Σ_Φ by Definition 2.2. Furthermore, by assumption $\mu(o) = \mathsf{e} < \mathsf{e}_+ = \mathsf{e_s}'$. Therefore, the value of every object in Σ_Φ is less than $\mathsf{e_s}'$.

Now considering a value a in the value set of the root less than $\mathsf{e_s}'$, we prove that there is an object of value a in Σ_Φ. We first show that $a \leq \mathsf{e}$. Then we consider the two cases $a = \mathsf{e}$ and $a < \mathsf{e}$, separately. In the first case we prove that the value of o is a and the second case we show that there is an object of value a in Σ_{sl}. Then, in both of these two cases, we use Equation 3.7 to conclude that there is an object of value a in Σ_Φ.

Now we explain the proof of there being an object of value a in Σ_Φ in detail. The value a is in $\mathbb{V}$ since a is in the value set of the root. Thus, as by definition the greatest value of $\mathbb{V}$ less than $\mathsf{e_s}' = \mathsf{e}_+$ is e and by assumption $a < \mathsf{e_s}'$, $a \leq \mathsf{e}$. Now as mentioned we consider two cases $a = \mathsf{e}$ and $a < \mathsf{e}$. The case $a = \mathsf{e}$ is trivial since o is an object of value e and by Equation 3.7 o is in Σ_Φ. So we consider the case $a < \mathsf{e}$. Then, since a is a member of the the value set of the root and $a < \mathsf{e}$, applying Lemma 3.19 for $b = \mathsf{e}$, we can conclude that an object p of value a is in the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$. The object p is in Σ_{sl} as Σ_{sl} is defined as the expansion of $\Phi_{sl}(C, \mu, \mathsf{e})$. Furthermore, because of Equation 3.7, Σ_{sl} is a subsequence of Σ_Φ. Hence, p is in Σ_Φ and we have $\mu(p) = a$. Now in both of the cases $a = \mathsf{e}$ and $a < \mathsf{e}$ we have proved that there is an object of value a is in Σ_Φ. But a was chosen as an arbitrary member of the value set of the root less than $\mathsf{e_s}'$. Consequently, every value in the value set of the root less than $\mathsf{e_s}'$ is the value of an object in Σ_Φ. So, as explained, the correctness of the first condition for C' is proved.

The second condition: The proof of the correctness of the second condition is similar to the proof of correctness of the first condition. As $\mathsf{e}' = \infty$, the condition is equivalent to the fact that there is an object of value a in Σ_Φ, for every value a in $\mathit{RootValueSet}(N(I))$. So, considering an arbitrary member b of $\mathit{RootValueSet}(N(I))$, we now prove there is an object of value b in Σ_Φ. By Observation 2.32 $\mathit{RootValueSet}(N(I))$ is a subset of the $\mathit{RootValueSet}(I)$. Thus, b is in $\mathit{RootValueSet}(I)$ as b is in $\mathit{RootValueSet}(N(I))$. Further-

more, as $\mathsf{e_s}' = \infty$, $b < \mathsf{e_s}'$ and hence as we proved the first condition of Definition 3.5 holds for C', there is an object of value b in Σ_Φ. So, C' meets the the second condition.

Now we can conclude that $C' = \pi^+(\mu, C)$ is valid as all the three conditions of Definition 3.5 are satisfied by C'. □

3.1.3 Eliminating Functions of the Third Type

The last way of updating the eliminating configuration is the situation in which $\mathsf{e} = \infty$ and $\mathsf{e_s} < \infty$. Since $\mathsf{e_s} < \mathsf{e}$, the query tree has a speedy leaf by Definition 3.2. Also, as $\mathsf{e} = \infty$, except objects of the speedy leaf, all objects are eliminated by Definition 3.3. In this situation, we easily add all remaining objects of the speedy leaf to our partial solution and we set $\mathsf{e_s} = \infty$. Remaining objects of the speedy leaf, by Definition 3.3, are objects of the speedy leaf with values at least $\mathsf{e_s}$. To add all objects of the speedy leaf with values at least $\mathsf{e_s}$ we use the function Φ_{sl} defined before.

Definition 3.11 *Given an instance $I = (Q, \omega, \mu)$ in which Q has at most one speedy leaf and a valid eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for I, $\pi^\infty(\mu, C)$ is defined if $\mathsf{e} = \infty$. Then, we define $\pi^\infty(\mu, C) = (\infty, \infty, \Phi_{sl}(C, \mu, \infty))$.*

Now we prove that the eliminating configuration resulting from applying the third type of eliminating function on a valid eliminating configuration is valid.

Lemma 3.21 *Given an eliminating configuration $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ for an instance $I = (Q, \omega, \mu)$ such that Q has at most one speedy leaf and $\pi^\infty(\mu, C)$ is defined, $\pi^\infty(\mu, C)$ is a valid eliminating configuration.*

Proof. As by Definition 3.11 the e-value and the $\mathsf{e_s}$-value of $\pi^\infty(\mu, C)$ are the same, by definition 3.2 $\pi^\infty(\mu, C)$ is an eliminating configuration and thus it remains to prove that $\pi^\infty(\mu, C)$ is valid. We prove that the conditions defined in Definition 3.5 are met by $\pi^\infty(\mu, C)$, one by one, starting from the third one. Then we prove the first and the second conditions, in order. We will use the fact that by Definition 3.11, the e-value, the $\mathsf{e_s}$-value, and the output of $\pi^\infty(\mathsf{e}, \mathsf{e_s}, \Phi)$ are, ∞, ∞, and $\Phi_{sl}(C, \mu, \infty)$, respectively.

To prove the third condition, we consider two cases based on Q having a speedy leaf or not. If Q has no speedy leaf, by Definition 3.9 $\Phi_{sl}(C, \mu, \infty) = \Phi$ and Φ is a partial solution by the third condition of Definition 3.5 for C. If Q has a speedy leaf then it follows immediately from Lemma 3.16 that $\Phi_{sl}(C, \mu, \infty)$ is a partial solution to I.

Now we prove the first condition. Since the $\mathsf{e_s}$-value of $\pi^{\infty}(\mathsf{e}, \mathsf{e_s}, \Phi)$ is ∞, all members of the value set of the root are less than the $\mathsf{e_s}$-value of $\pi^{\infty}(\mathsf{e}, \mathsf{e_s}, \Phi)$. Hence, the first condition is equivalent to the claim that for every member a of the value set of the root, there is an object of value a in the expansion of $\Phi_{sl}(C, \mu, \infty)$. The correctness of this claim follows from Lemma 3.19 for $b = \infty$.

Finally, we consider the second condition of Definition 3.5. As the e-value and the $\mathsf{e_s}$-value of $\pi^{\infty}(\mathsf{e}, \mathsf{e_s}, \Phi)$ are the same and by Observation 2.32 $\mathit{RootValueSet}(N(I))$ is a subset of the $\mathit{RootValueSet}(I)$, the correctness of the second condition follows from the correctness of the first condition.

Having proved $\pi^{+}(\mu, C)$ satisfies all the three conditions of Definition 3.5, now we can say $\pi^{+}(\mu, C)$ is valid. □

An eliminating function of one of the three types we defined in this section eliminates a number of objects of the instance and thus it can be seen as a step towards reaching an eliminating configuration in which all objects are eliminated and so the expansion of its output is a solution to the instance by Lemma 3.9. Motivated in this way, we define an eliminating step as as an eliminating function of one of these three types.

Definition 3.12 *An eliminating function π is an* eliminating step *if $\pi = \pi^{-}_{T}$ for a sub-union tree T of $N(Q)$, $\pi = \pi^{+}_{U}$ for a sub-intersection tree U of Q, or $\pi = \pi^{\infty}$.*

3.2 Eliminating Proofs

In this section we define an eliminating proof intuitively as a sequence of eliminating steps proving a certain sequence of objects is a solution of the instance. Recall that by Lemmas 3.12, 3.20, and 3.21 an eliminating step of each of the three types that we defined results in a valid eliminating configuration if it is applied to a valid eliminating configuration. So, starting with a valid eliminating configuration, after applying a number

of eliminating steps we may reach a valid eliminating configuration $(\mathsf{e}, \mathsf{e_s}, \Phi)$ in which $\mathsf{e} = \mathsf{e_s} = \infty$ and hence all objects are eliminated by Definition 3.3. Then, the expansion Σ of Φ is a solution for the instance by Lemma 3.9. In this way, informally speaking, using a sequence of eliminating steps we have proved that Σ is a solution to the instance. Thus, we define an eliminating proof as a sequence of eliminating steps that, by eliminating all objects, proves a certain sequence of objects is a solution to the instance.

Definition 3.13 *Given an instance $I = (Q, \omega, \mu)$, we define a valid sequence of eliminating steps for I and the result of a valid sequence of eliminating steps on I as follows.*

1. *An empty sequence is* valid for I *and* its result on I *is the eliminating configuration $C_0 = (-\infty, -\infty, \Phi_0)$ where Φ_0 is the empty sequence.*

2. *A sequence $\Pi = \pi_1, \ldots, \pi_n$ is* valid for I *if $\Pi' = \pi_1, \ldots, \pi_{n-1}$ is valid for I and providing C is the result of Π' on I, $\pi_n(\mu, C)$ is defined and is a valid eliminating configuration. Then, the* result of Π on I *is defined as $\pi_n(\mu, C)$.*

The result of a valid sequence Π of eliminating steps of $I = (Q, \omega, \mu)$ is denoted by result(I, Π). A valid sequence Π of eliminating steps of I is an eliminating proof for I *if result$(I, \Pi) = (\infty, \infty, \Phi)$, for some Φ.*

For the purpose of conciseness, given a sequence $\Pi = \pi_1, \ldots, \pi_n$ of eliminating steps, we use $\Pi^{(0)}$ to denote the empty sequence and use $\Pi^{(i)}$ to denote the sequence $\pi_1, \ldots, \pi_i$, for every i, $1 \leq i \leq n$. As the next lemma shows, the sequences $\Pi^{(i)}$, $0 \leq i \leq n$, are valid.

Lemma 3.22 *Consider an eliminating proof $\Pi = \pi_1, \ldots, \pi_n$ for an instance I. Then, $\Pi^{(i)}$ is a valid sequence of eliminating steps for I, for every i, $1 \leq i \leq n$.*

Proof. We prove the lemma by induction on i. By Definition 3.13, the lemma is true for $i = n$. Also, when the lemma is correct for $i + 1$, $\Pi^{(i+1)} = \pi_1, \ldots, \pi_{i+1}$ is valid for I and thus by Definition 3.13 $\pi_1, \ldots, \pi_i = \Pi^{(i)}$ is valid for I, for every i, $0 \leq i < n$. Hence, the lemma is true for all i, $0 \leq i \leq n$. □

Given a sequence Π of eliminating steps for an instance I, by the *output of* Π *on* I we mean the output of the result of Π on I. As we prove in the next lemma, the result of an

eliminating proof on an instance I is valid. Using this lemma together with Lemma 3.9 we will be able to show that the output of an eliminating proof of an instance I on I is a solution to I.

Lemma 3.23 *The result of a valid sequence $\Pi = \pi_1, \ldots, \pi_n$ of eliminating steps on an instance I is a valid eliminating configuration.*

Proof. We use induction on the length of Π to prove the lemma. For the base case, suppose $n = 0$. Then by Definition 3.13 the result of Π on I is $C = (\mathsf{e}, \mathsf{e_s}, \Phi)$ where $\mathsf{e} = \mathsf{e_s} = -\infty$ and Φ is the empty sequence. Hence, the inequality $\mathsf{e_s} \leq \mathsf{e}$ holds and thus by Definition 3.2 C is an eliminating configuration. Furthermore, for every value a which is in the value set of the root in I or is in the value set of the root in $N(I)$, $a \not< -\infty = \mathsf{e}$, $a \not< -\infty = \mathsf{e_s}$, and a is not the value of any object in the expansion of Φ as Φ is the empty sequence. So, the first two conditions of Definition 3.5 are met. Also, since by Definition 3.1 the empty sequence is a partial solution, the third condition is also satisfied. Therefore, C is valid.

Now we suppose the claim is true for $n - 1$ and we prove it for n. Since Π is valid, by Definition 3.13 $\pi_1, \ldots, \pi_{n-1} = \Pi^{(n-1)}$ is valid and thus by the induction hypothesis the result of $\Pi^{(n-1)}$ on I is a valid eliminating configuration C. Also, by assumption π_n is an eliminating step and hence by Definition 3.12 it is an eliminating function of the first, the second, or the third type. Moreover, as C is the result of $\pi_1, \ldots, \pi_{n-1}$ on I and Π is valid, by Definition 3.13 $\pi_n(\mu, C)$ is defined. Consequently, depending on π_n being of the first, the second, or the third type, it follows from Lemma 3.12, from Lemma 3.20, or from Lemma 3.21 that $\pi_n(\mu, C)$ is valid. So, as by Definition 3.13 the result of Π on I is $\pi_n(\mu, C)$, the lemma is correct for n. □

Given an instance I and an eliminating proof Π for I, we define the set of objects visited by Π in such a way that one can verify that Π is an eliminating proof for I and determine the output of Π on I by looking only at relative values of objects in this set. As a result, intuitively, if we change values of objects that are not visited by Π such that the value function of the instance remains ordered, Π remains an eliminating proof for the instance.

Since the definition of eliminating steps is based on the definition of functions *pos* and *find*, we first introduce sets of objects such that knowing relative values of objects in these sets, one can evaluate results of functions *pos* and *find*. Suppose we are given an instance

Figure 3.4 Boundary-sets of (I, l, a) for different values of a.

The sequence associated to l: (2 5) 7 (8) 10 11 (13 16) (18)

$a = 3$ $a = 8$ $a = 13+$ $a = 20$

$I = (Q, \omega, \mu)$, a leaf l of Q, and a value a in $\mathbb{V}^+$ and we require to evaluate *find*(I, l, a). Intuitively, if we find an object q of $\omega(l)$ such that $\mu(q) = a$ or find two consecutive objects p and q of $\omega(l)$ such that $\mu(p) < a < \mu(q)$, we can claim that q is the smallest object of $\omega(l)$ with a value at least a and thus *find*$(I, l, a) = q$ by Lemma 3.1. The object p in the second case is the biggest object of value at most a in $\omega(l)$. So in each of the two cases we can see that knowing values of the smallest object of value at least a (q in both cases) and the biggest object of value at most a (q in the first and p in the second case) is enough to determine *find*(I, l, a). The set of these objects is the boundary-set of (I, l, a) as defined more formally below. Figure 3.4 shows several examples of boundary-sets.

Definition 3.14 *Given an instance $I = (Q, \omega, \mu)$, a leaf l of Q, and a member b of $\mathbb{V}^+$, the set consisting of the biggest object in $\omega(l)$ with a value at most b (if there is any) and the smallest object in $\omega(l)$ with a value at least b (if there is any) is defined as the* boundary-set of (I, l, b).

In the next chapter we will prove that when a is a skirted value we can evaluate *find*(I, l, a) if we know relative values of objects in the boundary-set of $(I, l, \langle a \rangle)$ and a.

Before formally defining the set of objects visited by a sequence of eliminating steps, we intuitively describe which objects are going to be included in this set and why these objects are selected to be there. Consider an instance $I = (Q, \omega, \mu)$ and an eliminating proof $\Pi = \pi_1, \ldots, \pi_n$ of I. If we evaluate the output Φ of Π on I then we know that the result of $\Pi = \Pi^{(n)}$ on I is (∞, ∞, Φ), according to Definition 3.13. Moreover, informally speaking, to evaluate the result of $\Pi^{(i)}$ on I, for some i, the result C_i of $\Pi^{(i-1)}$ on I should be determined first as by Definition 3.13 C_i is evaluated by applying π_i on the result of $\Pi^{(i-1)}$ on I. Thus, one should be able to determine the result of the sequence $\Pi^{(i)}$ on I using only values of objects visited by Π, for every i, $1 \leq i \leq n$, because the output of Π on I should be determined by only looking at values of objects visited by Π, as mentioned earlier.

Motivated by what we explained, we now describe which objects should be in the set of objects visited by Π such that knowing objects visited by Π and the result C_{i-1} of $\Pi^{(i-1)}$, one can evaluate the result of $\Pi^{(i)}$ on I, for every i, $1 \leq i \leq n$. We start this discussion with eliminating steps of the first type. Suppose $\pi_i = \pi_T^-$, for a sub-union tree T of $N(Q)$ and an integer i, $1 \leq i \leq n$, and also suppose $C_j = (\mathsf{e}_j, \mathsf{e}_{\mathsf{s}j}, \Phi_j)$ is the result of $\Pi^{(j)}$ on I, for every j, $1 \leq j \leq n$. By Definition 3.13 $C_i = \pi_T^-(\mu, C_{i-1})$. Hence, according to Definition 3.8, $\mathsf{e}_i = \mathit{hmin}(\mu, C_{i-1}, T)$. Furthermore, according to Definition 3.7, $\mathit{hmin}(\mu, C_{i-1}, T)$ is determined by evaluating the minimum value of $\mu(\mathit{find}(I, l, \mathsf{e}_{i-1}))$, for all leaves l of T such that $\mathit{find}(I, l, \mathsf{e}_{i-1}) \neq \text{END}$. In addition, as we will prove in the next chapter, in order to evaluate $\mathit{find}(I, l, a)$, for a leaf l and a value a in $\mathbb{V}^+$, it suffices to know the relative values of the objects in the boundary set of $(I, l, \langle a \rangle)$ and a. Therefore, for all such leaves l of T, we include the object $\mathit{find}(I, l, \mathsf{e}_{i-1})$ (if it is not equal to END) and also the boundary-set of $(I, l, \langle \mathsf{e}_{i-1} \rangle)$ in the set of objects visited by Π to make it possible to verify that $\mathit{find}(I, l, \mathsf{e}_{i-1})$ is evaluated correctly using values of objects visited by Π.

Next we explain how we determine which objects are included in the set of objects visited by Π because of an eliminating step of the second type. Suppose $\pi_i = \pi_T^+$, for a sub-intersection tree T of Q, is an eliminating step in Π and consider the eliminating configuration $C_{i-1} = (\mathsf{e}_{i-1}, \mathsf{e}_{\mathsf{s}i-1}, \Phi_{i-1})$. In order to verify that $\pi_T^+(\mu, C_{i-1})$ is defined, one has to verify that every leaf of T has an object of value e_{i-1}, according to Definition 3.10. So, the object in $\omega(l)$ that is of value e_{i-1} will be in the set of objects visited by Π, for every leaf l of T. In addition, the output of $\pi_T^+(\mu, C_{i-1})$ is evaluated as $\Phi_{sl}(C_{i-1}, \mu, \mathsf{e}_{i-1}) \odot o$, for an object o, according to Definition 3.10. Thus, by Lemma 3.14, if the query tree has exactly one speedy leaf, all objects of the speedy leaf with values less than e_{i-1} and at least $\mathsf{e}_{\mathsf{s}i-1}$ are added to the solution in this step. Hence, the smallest object of the speedy leaf with a value at least e_{i-1} in addition to the biggest object p of the speedy leaf of a value at most e_{i-1} will be included in the set of objects visited by Π so that one can verify that p is the smallest object of $\mathit{sl}(Q)$ with a value at least e_{i-1}.

Finally, given a step π_i of the third type in Π, because of the reason that we now explain, we will define the set of objects visited by π_i in Π as the empty set. According to Definition 3.11, C_i is defined as $(\infty, \infty, \Phi_{sl}(C_{i-1}, \mu, \infty))$ and by Lemma 3.14 $\Phi_{sl}(C_{i-1}, \mu, \infty)$ is evaluated by appending objects of the speedy leaf (if it exists) of values at least $\mathsf{e}_{\mathsf{s}i-1}$

to the output of C_{i-1}. Thus, if we can confirm that the smallest object with a value at least $\mathsf{e}_{\mathsf{s}i-1}$ in the speedy leaf is the smallest object that is appended to the output by this step then we can easily verify if C_i is evaluated in the right manner. We now explain how this fact is confirmed. By Lemma 3.1 the smallest object with a value at least $\mathsf{e}_{\mathsf{s}i-1}$ is determined by evaluating $\mathit{find}(I, l, \mathsf{e}_{\mathsf{s}i-1})$ and, as mentioned earlier, we will prove that by looking at relative values of objects in the boundary-set of $(I, l, \langle \mathsf{e}_{\mathsf{s}i-1} \rangle)$ and $\mathsf{e}_{\mathsf{s}i-1}$, one can verify if $\mathit{find}(I, l, \mathsf{e}_{\mathsf{s}i-1})$ is evaluated correctly. Also, as we will prove in Lemma 3.24, when there is a speedy leaf, the boundary-set of $(I, l, \langle \mathsf{e}_{\mathsf{s}j} \rangle)$ is in the set of objects visited by $\Pi^{(j)}$, for any j, $0 \leq j \leq n$. Therefore, knowing values of objects that are already added to the set of objects visited by Π, we can verify the correctness of the evaluation of C_i. Hence, it is not necessary to include any new object in the set of objects visited by Π.

We now formally introduces the set of objects visited by an eliminating proof and then we prove Lemma 3.24, as explained above.

Definition 3.15 *Suppose $\Pi = \pi_1, \ldots, \pi_n$ is a valid sequence of eliminating steps for an instance $I = (Q, \omega, \mu)$ and also suppose $C_i = (\mathsf{e}_i, \mathsf{e}_{\mathsf{s}i}, \Phi_i)$ is the result of $\Pi^{(i)}$ on I, for every i, $1 \leq i \leq n$. For every i, $1 \leq i \leq n$, we define the set $\mathit{Visited}_i(I, \Pi)$ as follows.*

If $\pi_i = \pi_T^-$, for a sub-union tree T of $N(Q)$, $\mathit{Visited}_i(I, \Pi)$ is the set

$$\bigcup_{l \in \mathit{leaves}(T)} \{\mathit{find}(I, l, \mathsf{e}_{i-1})\} \cup \mathcal{S}_l - \{\text{END}\},$$

where $\mathcal{S}_l$ is the boundary-set of $(I, l, \langle \mathsf{e}_{i-1} \rangle)$, for every leaf l of T.

If $\pi_i = \pi_T^+$, for a sub-intersection tree T of Q, $\mathit{Visited}_i(I, \Pi)$ is the set

$$\mathcal{T} \cup \bigcup_{l \in \mathit{leaves}(T)} \{\mathit{find}(I, l, \mathsf{e}_{i-1})\},$$

where $\mathcal{T}$ is empty if Q has no speedy leaf and otherwise, $\mathcal{T}$ is the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_{i-1})$.

Finally, if $\pi_i = \pi^\infty$, $\mathit{Visited}_i(I, \Pi) = \emptyset$.

The set of objects visited by Π *is defined as* $\mathit{Visited}(I, \Pi) = \bigcup_{i=1}^{n} \mathit{Visited}_i(I, \Pi)$.

Lemma 3.24 *Suppose $\Pi = \pi_1, \ldots, \pi_n$ is an eliminating proof for an instance $I = (Q, \omega, \mu)$ and Q has a speedy leaf. Furthermore suppose $(\mathsf{e}_i, \mathsf{e}_{\mathsf{s}i}, \Phi_i)$ is the result of $\Pi^{(i)}$ on I for every*

i, $0 \leq i \leq n$. *Then, the boundary-set of* $(I, sl(Q), \langle \mathsf{e}_{s_i} \rangle)$ *is a subset of* $Visited(I, \Pi)$ *if* $\mathsf{e}_{s_i} \notin \{\infty, -\infty\}$, *for every i,* $0 \leq i \leq n$.

Proof. We use induction on i to prove the claim. The claim is true for $i = 0$ since by Definition 3.13 $\mathsf{e}_{s_0} = -\infty$. So suppose the claim holds for $i-1$, $1 \leq i \leq n$. If $\pi_i = \pi_T^-$, for some T, the claim is true since $\mathsf{e}_{s_i} = \mathsf{e}_{s_{i-1}}$ by Definition 3.8. Furthermore, if $\pi_i = \pi^\infty$ the claim is correct as by Definition 3.11 $\mathsf{e}_{s_i} = \infty$. Now, suppose $\pi_i = \pi_T^+$, for some T. Then, by Definition 3.10 $\mathsf{e}_{s_i} = \mathsf{e}_{i-1+}$ and hence $\langle \mathsf{e}_{s_i} \rangle = \mathsf{e}_{i-1}$. Also, as $\pi_i = \pi_T^+$, by Definition 3.15 the boundary set of $(I, sl(Q), \mathsf{e}_{i-1})$ is a subset of $Visited_i(I, \Pi)$. Thus, the claim is true for i in this case, as well. □

Given a valid sequence Π of m eliminating steps, as it is clear from Definition 3.15, the definition of the set $Visited_i(I, \Pi)$, for some i, $1 \leq i \leq m$, depends only on the the instance I, the ith step of Π, and the e-value of the result of $\Pi^{(i-1)}$ on I. Now consider two valid sequences of eliminating steps Π_1 and Π_2 such that $\Pi_1^{(i)} = \Pi_2^{(i)} = \pi_1, \ldots, \pi_i$, for some i. Then the ith steps of Π_1 and Π_2 both are π_i. Also, $\Pi_1^{(i-1)} = \pi_1, \ldots, \pi_{i-1} = \Pi_2^{(i-1)}$and hence $result(I, \Pi_1^{(i-1)}) = result(I, \Pi_2^{(i-1)})$. Therefore, the e-values of the results of $\Pi_1^{(i-1)}$ and $\Pi_2^{(i-1)}$ on I as well as the ith steps of Π_1 and Π_2 are the same. So, the equation $Visited_i(I, \Pi_1) = Visited_i(I, \Pi_2)$ holds. This proves the next lemma.

Lemma 3.25 *Given two valid sequences* Π_1 *and* Π_2 *of eliminating steps of an instance I and an integer i such that* $\Pi_1^{(i)} = \Pi_2^{(i)}$, $Visited_i(I, \Pi_1) = Visited_i(I, \Pi_2)$. □

The following lemma will be used for a couple of purposes in the next chapters one of which is to obtain a lower bound on the size of the set of objects visited by an eliminating proof when analyzing the running time of our algorithm.

Lemma 3.26 *Consider a valid sequence* $\Pi = \pi_1, \ldots, \pi_n$ *of eliminating steps of an instance I and suppose* e_i *is the* e*-value of the result of* $\Pi^{(i)}$ *on I, for every i,* $0 \leq i \leq n$. *Then, given an integer m,* $0 \leq m \leq n$, $\mathsf{e}_m \in \{-\infty, \infty\}$ *or there is an object o of I in* $Visited(I, \Pi)$ *such that* $\langle \mathsf{e}_m \rangle = \mu(o)$.

Proof. To prove the lemma we consider several possibilities based on $m = 0$ or not and if $m \neq 0$, based on the type of the step π_m. Since $\Pi^{(0)}$ is empty, by Definition 3.13 the result

of $\Pi^{(0)}$ on I is $(-\infty, -\infty, \Phi_0)$, for some Φ_0, and thus $\mathsf{e}_0 = -\infty$. So the lemma for $m = 0$ is true.

Now we suppose $m > 0$. Then, by Definition 3.13 the result of $\Pi^{(m)}$ is evaluated as $\pi_m(\mu, C)$ where C is the result of the $\Pi^{(m-1)}$. Hence, e_m is the e-value of $\pi_m(\mu, C)$ and e_{m-1} is the e-value of C. We now consider different possibilities based on the type of π_m.

If $\pi_m = \pi_T^-$, for some T, then by Definition 3.8, $\mathsf{e}_m = \mathit{hmin}(\mu, C, T)$. By Definition 3.7 $\mathit{hmin}(\mu, C, T)$ is either ∞ or $\mu(\mathit{find}(I, l, \mathsf{e}_{m-1}))$, for a leaf l of T, such that $\mathit{find}(I, l, \mathsf{e}_{m-1}) \neq$ END and hence by Lemma 3.1 $\mathit{find}(I, l, \mathsf{e}_{m-1})$ is an object of l. Also, given a leaf l of T such that $\mathit{find}(I, l, \mathsf{e}_{m-1}) \neq$ END, by Definition 3.15 $\mathit{find}(I, l, \mathsf{e}_{m-1})$ is in $\mathit{Visited}_m(I, \Pi)$ and thus as by Definition 3.15 $\mathit{Visited}(I, \Pi) = \bigcup_{m=1}^{n} \mathit{Visited}_m(I, \Pi)$, $\mathit{find}(I, l, \mathsf{e}_{m-1})$ is in $\mathit{Visited}(I, \Pi)$. Therefore, in this case $\mathsf{e}_m = \mathit{hmin}(\mu, C, T)$ is either ∞ or the value of an object of I in $\mathit{Visited}(I, \Pi)$. So, e_m is a main value and hence $\langle \mathsf{e}_m \rangle = \mathsf{e}_m$. Consequently, either $\mathsf{e}_m = \infty$ or $\langle \mathsf{e}_m \rangle$ is the value of an object of I in $\mathit{Visited}(I, \Pi)$.

Now suppose $\pi_m = \pi_T^+$, for a sub-intersection T of Q. We prove the lemma by showing that $\mathsf{e}_{m-1} = \langle \mathsf{e}_m \rangle$ and $\mathit{find}(I, l, \mathsf{e}_{m-1})$ is an object of value e_{m-1} in $\mathit{Visited}(I, \Pi)$, for every leaf l of T. Since $\pi_m = \pi_T^+$, by Definition 3.10 $\mathsf{e}_m = \mathsf{e}_{m-1+}$ and hence $\mathsf{e}_{m-1} = \langle \mathsf{e}_m \rangle$. Furthermore, given a leaf l of T, by Definition 3.15 $\mathit{find}(I, l, \mathsf{e}_{m-1})$ is in $\mathit{Visited}_m(I, \Pi)$ and thus is in $\mathit{Visited}(I, \Pi)$. In addition, since by Lemma 3.22 $\Pi^{(m)}$ is valid, by Definition 3.13 $\pi_m(\mu, C) = \pi_T^+(\mu, C)$ is define and so by Definition 3.10 every leaf of T has an object of value e_{m-1}. As a result, by Lemma 3.2 $\mathit{find}(I, l, \mathsf{e}_{m-1})$ is an object of l of value e_{m-1}, for every leaf l of T. So, the claim is correct in this case, as well.

Finally, if $\pi_m = \pi^\infty$, by Definition 3.11 $\mathsf{e}_m = \infty$. Consequently, the claim is correct for m in all cases. □

In the next chapter, given a proof $\mathcal{P}$ for an instance I, we prove that an eliminating proof for I not visiting any object outside $\mathit{Visited}(\mathcal{P})$ exists.

Chapter 4

Eliminating Proofs versus Proofs

In this chapter we prove that, given a subset $\mathcal{V}$ of $Objects(I)$, an instance I has an eliminating proof Π such that $Visited(I, \Pi) \subseteq \mathcal{V}$ if and only if I has a proof $\mathcal{P}$ such that $Visited(\mathcal{P}) \subseteq \mathcal{V}$. The importance of proving this fact is that the definition of our difficulty functions is based on sets of objects visited by proofs while the running time of our algorithm will be expressed in a form based on sets of objects visited by eliminating proofs. It is thus necessary to provide a connection between these two families of sets of objects. We first in Section 4.1 prove the "only if" part of this claim and then in the following section we prove the 'if" part of the claim.

4.1 Building a Proof

In this section we show that, having an eliminating proof Π for an instance I, one can build a proof for I with the same set of visited objects as Π. To prove this fact, we first show that the expansion Σ of the output of Π on I is a solution to every instance $Visited(I, \Pi)$-identical with I and then we use Lemma 2.9 to construct a proof not visiting any object outside $Visited(I, \Pi)$. Given an instance J that is $Visited(I, \Pi)$-identical with I, in order to prove that Σ is a solution to J, we show that Π is an eliminating proof for J and that the result of Π on I is the same as the result of Π on J and then we use Lemma 3.9 to conclude that Σ is a solution to J.

We now describe the idea of the proof of the claim that, given an eliminating proof

$\Pi = \pi_1, \ldots, \pi_n$ for an instance $I = (Q, \omega, \mu)$ and an instance $J = (Q, \omega, \nu)$ such that I and J are *Visited*(I, Π)-identical, Π is an eliminating proof for J and the result of Π on J is the same as the result of Π on I. Informally speaking, we will inductively prove the following claims for for every i, $0 \leq i \leq n$: First, $\Pi^{(i)}$ is valid for J. Second, the output of the result C_I of $\Pi^{(i)}$ on I is the same as the output of the result C_J of $\Pi^{(i)}$ on J. Third, the e-values (the $\mathsf{e_s}$-values) of C_I and C_J can be described in the same way in terms of the values of objects visited by Π, meaning that as an example, if the e-value (the $\mathsf{e_s}$-value) of C_I equals the μ-value of an object o in *Visited*(I, Π) then the e-value (the $\mathsf{e_s}$-value) of C_J equals the ν-value of the same object o. Having proved these facts for $i = n$, it will be proved that the results of $\Pi^{(n)}$ on I and on J are the same and so, as explained above, we can create the proof $\mathcal{P}$ not visiting any object outside *Visited*(I, Π).

To prove that e-values and $\mathsf{e_s}$-values of C_I and C_J can be described in the same way in terms of the values of objects in the set $\mathcal{S} = \mathit{Visited}(I, \Pi)$, we now formally define the concept of a value in $\mathbb{V}^+$ being described in terms of values objects in the set $\mathcal{S}$. Informally speaking, a value a in $\mathbb{V}^+$ is described using μ-values of objects in a set $\mathcal{S}$ if $a \in \{-\infty, \infty\}$ or $\langle a \rangle$ is the μ-value of an object in $\mathcal{S}$. Hence, aside from $-\infty$ and ∞, only those values that equal $\mu(o)$ or $\mu(o)_+$, for an object o in $\mathcal{S}$, can be described.

Definition 4.1 *Given a set $\mathcal{S}$ of objects of a signature G and a value function ξ for G, we define* apparent$(\mathcal{S}, \xi) = \{\infty, -\infty\} \cup \bigcup_{o \in \mathcal{S}} \{\xi(o), \xi(o)_+\}$.

We now formally define how values in *apparent*$(\mathcal{S}, \xi)$ are described. We introduce a number of descriptors each corresponding to a member a in the set *apparent*$(\mathcal{S}, \xi)$ and storing information indicating whether a is a main value or not and if $a \notin \{\infty, -\infty\}$, specifying an object o such that $a = \xi(o)$ or $a = \xi(o)_+$.

Definition 4.2 *Given a set $\mathcal{S}$ of objects, an* $\mathcal{S}$-descriptor *is a member of $\{\infty, -\infty\}$ or is a pair (o, i) where $o \in \mathcal{S}$ and $i \in \{0, 1\}$.*

The next definition shows how a value described by an $\mathcal{S}$-descriptor is determined.

Definition 4.3 *Given a set $\mathcal{S}$ of objects of a signature G, a value function ξ for G, and a $\mathcal{S}$-descriptor y, the* ξ-meaning of y, *denoted by $mean_\xi(y)$, is defined as follows.*

$$mean_\xi(y) = \begin{cases} y & \text{if } y \in \{-\infty, \infty\} \\ \xi(o) & \text{if } y = (o, 0), \text{ for some object } o \\ \xi(o)+ & \text{if } y = (o, 1), \text{ for some object } o \end{cases}$$

Recall that, as we explained, we inductively will prove for any i, $0 \leq i \leq n$, that the e-value (the $\mathsf{e_s}$-value) of the result C_I of $\Pi^{(i)}$ on I and the e-value (the $\mathsf{e_s}$-value) of the result C_J of $\Pi^{(i)}$ on J can be described in the same way in terms of the values of objects visited by Π. We now explain, how this fact is proved for i and, having proved it for i, how we will prove it for $i+1$. As we will prove in Theorem 4.5, there is a $Visited(I, \Pi)$-descriptor y such that (z such that) e-values ($\mathsf{e_s}$-values) of C_I and C_J are the μ-meaning and the ν-meaning of y (of z), respectively. The next lemma proves that this result will yield the fact that I and J are the same in terms of relative values of objects visited by Π and the e-value and the $\mathsf{e_s}$-value of the result of $\Pi^{(i)}$. Also, we have defined the set of objects visited by Π such that the result of $\Pi^{(i+1)}$ is determined using these relative values. These facts will enable us to prove the induction claims for $i + 1$, as we will see.

Lemma 4.1 *Given a set $\mathcal{S}$ of objects of a signature (Q, ω), a $\mathcal{S}$-descriptor y, and two instances $I = (Q, \omega, \mu)$ and $J = (Q, \omega, \nu)$ that are $\mathcal{S}$-identical, the statements $\mu(o) < mean_\mu(y)$, $\mu(o) = mean_\mu(y)$, and $\mu(o) > mean_\mu(y)$ are equivalent to the statements $\nu(o) < mean_\nu(y)$, $\nu(o) = mean_\nu(y)$, and $\nu(o) > mean_\nu(y)$, respectively, for every object o of $\mathcal{S}$.*

Proof. We prove the lemma by considering different possibilities for y. We first consider the cases $y = \infty$ and $y = -\infty$. Then, we consider the case in which $y = (p, 0)$, for an object p in $\mathcal{S}$, and finally we prove the lemma for the case $y = (p, 1)$, for some p in $\mathcal{S}$.

We now discuss the case $y = \infty$ and then we consider the case $y = -\infty$. We prove the lemma in this case by proving that the μ-meaning and the ν-meaning of y are greater than the μ-value and the ν-value of o, respectively. By Definition 4.3 $mean_\mu(y) = mean_\nu(y) = \infty$. Also, by Definition 2.5 $\mu(o) \neq \infty$ and $\nu(o) \neq \infty$. Thus, since both $\mu(o)$ and $\nu(o)$ are in $\mathbb{V}$ and by definition ∞ is the greatest value of $\mathbb{V}$, $\mu(o) < \infty = mean_\mu(y)$ and $\nu(o) < \infty = mean_\nu(y)$. Hence, the lemma is proved in this case as the two statements

$\mu(o) < \mathit{mean}_\mu(y)$ and $\nu(o) < \mathit{mean}_\nu(y)$ are true and so the other four statements stated in the lemma are false. The proof for the case in which y is equal to $-\infty$ is also similar: it suffices to replace all occurrences of "∞", "$<$", and " biggest" in the argument by "$-\infty$", "$>$", and " smallest", respectively.

Now we prove the lemma in the cases in which y is not in $\{-\infty, \infty\}$, and hence by Definition 4.2 there is an object p in $\mathcal{S}$ and an integer i in $\{0, 1\}$ such that $y = (p, i)$. First we consider the case $i = 0$ and after that we will discuss the case $i = 1$. By Definition 4.3 $\mathit{mean}_\mu(y) = \mu(p)$ and $\mathit{mean}_\nu(y) = \nu(p)$. Since by assumption p and o are in $\mathcal{S}$ and I and J are $\mathcal{S}$-identical, by Definition 2.15 each of the three comparison propositions $p < o$, $p == o$, and $p > o$ is satisfied by μ if and only if that comparison proposition is satisfied by ν. Hence, the statements $\mu(o) < \mu(p)$, $\mu(o) = \mu(p)$, and $\mu(o) > \mu(p)$ are equivalent to the statements $\nu(o) < \nu(p)$, $\nu(o) = \nu(p)$, and $\nu(o) > \nu(p)$, respectively. So as $\mu(p) = \mathit{mean}_\mu(y)$ and $\nu(p) = \mathit{mean}_\nu(y)$, the statements $\mu(o) < \mathit{mean}_\mu(y)$, $\mu(o) = \mathit{mean}_\mu(y)$, and $\mu(o) > \mathit{mean}_\mu(y)$ are equivalent to the statements $\nu(o) < \mathit{mean}_\nu(y)$, $\nu(o) = \mathit{mean}_\nu(y)$, and $\nu(o) > \mathit{mean}_\nu(y)$, respectively.

Finally we consider the case in which $i = 1$. Thus, by Definition 4.3 the $\mathit{mean}_\mu(y) = \mu(p)+$ and $\mathit{mean}_\nu(y) = \nu(p)+$. In this last case, we consider three subcases based on the relative values of $\mu(o)$ and $\mu(p)$.

Case 1: $\mu(o) = \mu(p)$**.**

In this subcase, as by assumption p and o are in $\mathcal{S}$ and I and J are $\mathcal{S}$-identical, by Definition 2.15 the correctness of the equation $\mu(o) = \mu(p)$ yields the correctness of the equation $\nu(o) = \nu(p)$. By definition, for every value a in $\mathbb{V}$, $a < a+$. Thus, $\mu(o) = \mu(p) < \mu(p)+ = \mathit{mean}_\mu(y)$ and $\nu(o) = \nu(p) < \nu(p)+ = \mathit{mean}_\nu(y)$. Therefore, the two statements $\mu(o) < \mathit{mean}_\mu(y)$ and $\nu(o) < \mathit{mean}_\nu(y)$ are true and so the other four statements stated in the lemma are false. Hence, the lemma is true in this subcase.

Case 2: $\mu(o) > \mu(p)$**.**

We have $\nu(o) > \nu(p)$ as by assumption p and o are in $\mathcal{S}$ and I and J are $\mathcal{S}$-identical. Thus, as $\mu(o) > \mu(p)$ and $\nu(o) > \nu(p)$, by Observation 2.29 $\mu(o) > \mu(p)+ = \mathit{mean}_\mu(y)$ and $\nu(o) > \nu(p)+ = \mathit{mean}_\nu(y)$. Consequently, the two statements $\mu(o) > \mathit{mean}_\mu(y)$

and $\nu(o) > mean_\nu(y)$ are true and hence the other four statements stated in the lemma are false. Thus, the lemma is true in this subcase.

Case 3: $\mu(o) < \mu(p)$**.**

By changing all ">" operators in the argument of the previous subcase to "<", the lemma is proved in this subcase also.

Now, in all cases we have proved the lemma. □

As explained, a result of Lemma 4.1 is that given two $\mathcal{S}$-identical instances I and J, for a set $\mathcal{S}$, and a $\mathcal{S}$-descriptor y, the relative values of μ-values of the objects of $\mathcal{S}$ and $mean_\mu(y)$ are the same as the relative values of ν-values of the objects in $\mathcal{S}$ and $mean_\nu(y)$. In the next two lemmas we show that existence of the $\mathcal{S}$-descriptor μ proves more similarities between $mean_\mu(y)$ and $mean_\nu(y)$. After proving the next lemma, by applying Lemma 4.1, we will be able to prove the additional result that in such situations the relative values of μ-values of the objects in $\mathcal{S}$ and $\langle mean_\mu(y)\rangle$ are the same as the relative values of ν-values of that objects and $\langle mean_\nu(y)\rangle$. Also, by proving Lemma 4.3 we will know that if one of $mean_\mu(y)$ and $mean_\nu(y)$ is a main value, the other one is also a main value.

Lemma 4.2 *Given a set $\mathcal{S}$ of objects of a signature G and a $\mathcal{S}$-descriptor y, there is a $\mathcal{S}$-descriptor z such that for any value function μ for G, $mean_\mu(z) = \langle mean_\mu(y)\rangle$.*

Proof. We consider different cases depending on y being a member of $\{-\infty, \infty\}$ or not. If $y \in \{-\infty, \infty\}$, by Definition 4.3 the μ-meaning of y is also in $\{-\infty, \infty\}$, for every value function μ for G. Hence, in this case, the μ-meaning of y is a main value as $-\infty$ or ∞ are in $\mathbb{V}$. Thus, by definition $mean_\mu(y) = \langle mean_\mu(y)\rangle$ and so for the choice $z = y$, $mean_\mu(z) = \langle mean_\mu(y)\rangle$. Hence, the lemma is true in this case.

Now we consider the other case, in which $y = (o, i)$ for an object o of $\mathcal{S}$ and an integer i. Hence, by Definition 4.3, $mean_\mu(y)$ is $\mu(o)$ or $\mu(o)+$ and thus $\langle mean_\mu(y)\rangle = \mu(o)$, for any value function μ for G. Hence, as $o \in \mathcal{S}$, by Definition 4.2 $(o, 0)$ is a $\mathcal{S}$-descriptor and by Definition 4.3, $mean_\mu((o, 0)) = \mu(o) = \langle mean_\mu(y)\rangle$ for any value function μ for G. So, the choice $z = (o, 0)$ proves the lemma. □

Lemma 4.3 *Given a set $\mathcal{S}$ of objects of a signature G, two value functions μ and ν for G, and a $\mathcal{S}$-descriptor y, $mean_\mu(y)$ is a main value if and only if $mean_\nu(y)$ is a main value.*

Proof. We prove the lemma by considering different possibilities for y. By Definition 4.3 if $y \in \{\infty, -\infty\}$ then $mean_\mu(y)$ and $mean_\nu(y)$ both are in $\{\infty, -\infty\}$ and thus both are main values. If $y \notin \{\infty, -\infty\}$, by Definition 4.2 y is of the form (o, i). If $i = 0$, according to Definition 4.3, $mean_\mu(y)$ and $mean_\nu(y)$ are the μ-value and the ν-value of o, respectively. Hence, in this case, by Definition 2.5 both are in $\mathbb{V}$ and so both are main values. Finally, if $i = 1$, by Definition 4.3 $mean_\mu(y) = \mu(o)+$ and $mean_\nu(y) = \nu(o)+$ and thus both of $mean_\mu(y)$ and $mean_\nu(y)$ are skirted values. So the lemma is true in all cases. □

Recall that when motivating the definition of the set of objects visited by an eliminating proof of an instance $I = (Q, \omega, \mu)$, we claimed that $find(I, l, a)$ can be determined if the relative values of objects in the boundary set of $(I, l, \langle a \rangle)$ and a is known, for every leaf l of Q and member a of $\mathbb{V}^+$. By proving the next lemma and using Lemma 4.1 we will be able to prove such a fact.

Lemma 4.4 *Consider a set $\mathcal{S}$ of objects of a signature (Q, ω), a $\mathcal{S}$-descriptor y, two instances $I = (Q, \omega, \mu)$ and $J = (Q, \omega, \nu)$ that are $\mathcal{S}$-identical, and a leaf l of Q such that the boundary-set of $(I, l, \langle mean_\mu(y) \rangle)$ is a subset of $\mathcal{S}$. Then the following claims are true.*

Claim 1. $find(I, l, mean_\mu(y)) = find(J, l, mean_\nu(y))$.

Claim 2. *The boundary-set of $(I, l, \langle mean_\mu(y) \rangle)$ equals the boundary-set of $(J, l, \langle mean_\nu(y) \rangle)$.*

Proof. We prove the two claims of the lemma by considering two cases based on $\omega(l)$ having an object of μ-value $\langle mean_\mu(y) \rangle$ or not. In the following argument, we define z to be an $\mathcal{S}$-descriptor such that $mean_\mu(z) = \langle mean_\mu(y) \rangle$ and $mean_\nu(z) = \langle mean_\nu(y) \rangle$; as y is an $\mathcal{S}$-descriptor, by Lemma 4.2 such an $\mathcal{S}$-descriptor exists.

Case 1: $\omega(l)$ has an object o of μ-value $\langle mean_\mu(y) \rangle$.

We first show that $\nu(o) = \langle mean_\nu(y) \rangle$, then we prove Claim 2, and finally in order to prove Claim 1 we will consider two subcases depending on $mean_\mu(y)$ being a main value or a skirted value.

We show that $o \in \mathcal{S}$ and then we use Lemma 4.1 to prove that $\nu(o) = \langle mean_\nu(y)\rangle$. Since μ is ordered, the μ-value of each object placed after o in $\omega(l)$ is greater than $\langle mean_\mu(y)\rangle$ and hence o is the biggest object of l of value at most $\langle mean_\mu(y)\rangle$. Therefore, by Definition 3.14 o is in the boundary-set of $(I, l, \langle mean_\mu(y)\rangle)$ and so $o \in \mathcal{S}$ as by assumption the boundary-set of $(I, l, \langle mean_\mu(y)\rangle)$ is a subset of $\mathcal{S}$. In addition, by the definition of z, $mean_\mu(z) = \langle mean_\mu(y)\rangle = \mu(o)$. Thus, as $o \in \mathcal{S}$, by Lemma 4.1 $mean_\nu(z) = \nu(o)$. So, as by the definition of z, $mean_\nu(z) = \langle mean_\nu(y)\rangle$,

$$\langle mean_\nu(y)\rangle = mean_\nu(z) = \nu(o). \tag{4.1}$$

Now we prove Claim 2. Since μ and ν are ordered, o is the biggest (smallest) object of l with a μ-value at most (at least) $\mu(o) = \langle mean_\mu(y)\rangle$ and also the biggest (smallest) object of l with a ν-value at most (at least) $\nu(o) = \langle mean_\nu(y)\rangle$. So, by Definition 3.14 both boundary-sets of $(I, l, \langle mean_\mu(y)\rangle)$ and $(J, l, \langle mean_\nu(y)\rangle)$ are $\{o\}$. Thus, Claim 2 is true.

Now, to prove Claim 1 we have different arguments depending on $mean_\mu(y)$ being a main value or a skirted value.

Case 1a: $mean_\mu(y)$ is a main value.

We prove Claim 1 by showing that both objects $find(I, l, mean_\mu(y))$ and $find(J, l, mean_\nu(y))$ equal o. Since $mean_\mu(y)$ is a main value, by Lemma 4.3 $mean_\nu(y)$ is also a main value and hence $mean_\mu(y) = \langle mean_\mu(y)\rangle$ and $mean_\nu(y) = \langle mean_\nu(y)\rangle$. Therefore, since by assumption $\mu(o) = \langle mean_\mu(y)\rangle$ and by Equation 4.1 $\nu(o) = \langle mean_\nu(y)\rangle$, we may conclude that $mean_\mu(y) = \mu(o)$ and $mean_\nu(y) = \nu(o)$. As a result, l has an object of μ-value $mean_\mu(y)$ and an object of ν-value $mean_\nu(y)$. Consequently, by Lemma 3.2 $find(I, l, mean_\mu(y))$ and $find(J, l, mean_\nu(y))$ are objects of μ-value $mean_\mu(y) = \mu(o)$ and ν-value $mean_\nu(y) = \nu(o)$, respectively. Therefore, as $\omega(l)$ has no two objects with the same μ-values or with the same ν-values, $find(I, l, mean_\mu(y))$ and $find(J, l, mean_\nu(y))$ both are equal to o. So, Claim 1 is true.

Case 1b: $mean_\mu(y) = \langle mean_\mu(y)\rangle+$.

We show that the inequalities $\mu(o) < mean_\mu(y)$ and $\nu(o) < mean_\nu(y)$ hold and then using this result in each of the two cases that o is the last object of $\omega(l)$ and o is not the

last object, we prove Claim 1. Since $mean_\mu(y)$ is a skirted value, by Lemma 4.3 $mean_\nu(y)$ is a skirted value too and so $mean_\nu(y) = \langle mean_\nu(y)\rangle_+$. Therefore,

$$\mu(o) = \langle mean_\mu(y)\rangle < \langle mean_\mu(y)\rangle_+ = mean_\mu(y) \tag{4.2}$$

and due to Equation 4.1

$$\nu(o) = \langle mean_\nu(y)\rangle < \langle mean_\nu(y)\rangle_+ = mean_\nu(y). \tag{4.3}$$

Hence, if o is the last object of $\omega(l)$, the μ-values and ν-values of all objects of $\omega(l)$ are less than $mean_\mu(y)$ and $mean_\nu(y)$, respectively and we will have $find(I, l, mean_\mu(y)) = find(J, l, mean_\nu(y)) = \text{END}$ by Lemma 3.1.

Now we prove Claim 1 when o is not the last object of $\omega(l)$. In this case suppose p is the first object after o in $\omega(l)$. As μ and ν are ordered, $\mu(p) > \mu(o) = \langle mean_\mu(y)\rangle$ and $\nu(p) > \nu(o) = \langle mean_\nu(y)\rangle$. Thus, by Observation 2.29 $\mu(p)$ and $\nu(p)$ are at least $\langle mean_\mu(y)\rangle_+ = mean_\mu(y)$ and at least $\langle mean_\nu(y)\rangle_+ = mean_\nu(y)$, respectively. So, p is the first object of μ-value at least $mean_\mu(y)$ and also is the first object of ν-value at least $mean_\nu(y)$ as $\mu(o) < mean_\mu(y)$ (Equation 4.2) and $\nu(o) < mean_\nu(y)$ (Equation 4.3). Therefore, by Lemma 3.1, $find(I, l, mean_\mu(y))$ and $find(J, l, mean_\nu(y))$ both are equal to p.

Case 2: there is no object of μ-value $\langle mean_\mu(y)\rangle$ in $\omega(l)$.

We consider three subcases depending on whether there is any object with a μ-value less (greater) than $mean_\mu(y)$. Note that since there is no object of μ-value $\langle mean_\mu(y)\rangle$ in $\omega(l)$ and by definition μ-values of all objects are main values, there is no object of μ-value $mean_\mu(y)$ in $\omega(l)$, as well.

Case 2a: μ-values of all objects in $\omega(l)$ are less than $mean_\mu(y)$.

In this case, by Observation 2.30 μ-values of all objects of l are at most $\langle mean_\mu(y)\rangle$. We first show that μ-values and the ν-values of all objects of l are less than $\langle mean_\mu(y)\rangle$ and $\langle mean_\nu(y)\rangle$, respectively, and then using this fact we prove Claims 2 and 1, in that order.

We now show that $\mu(o) < \langle mean_\mu(y)\rangle$ and $\nu(o) < \langle mean_\nu(y)\rangle$, for any object o of l, and

then we prove Claim 2. As we supposed there is no object of μ-value $\langle mean_\mu(y)\rangle$ in $\omega(l)$ and we proved in this subcase values of all object of l are at most $\langle mean_\mu(y)\rangle$, the μ-value of the last object of $\omega(l)$ is less than $\langle mean_\mu(y)\rangle$. As a result, the last object q of $\omega(l)$ is the biggest object of l with a value at most $\langle mean_\mu(y)\rangle$ and there is no object of value at least $\langle mean_\mu(y)\rangle$ in $\omega(l)$. So, the boundary-set of $(I, l, \langle mean_\mu(y)\rangle)$ is $\{q\}$ (Definition 3.14). Thus q is in $\mathcal{S}$ as by assumption the boundary-set of $(I, l, \langle mean_\mu(y)\rangle)$ is a subset of $\mathcal{S}$. As a result, by Lemma 4.1 the sentences $\mu(q) < mean_\mu(z)$ and $\nu(q) < mean_\nu(z)$ are equivalent. So, since $\mu(q) < \langle mean_\mu(y)\rangle = mean_\mu(z)$, $\nu(q) < mean_\nu(z) = \langle mean_\nu(y)\rangle$. Hence, as q is the last object of l, q is the biggest object of l with a ν-values at most $\langle mean_\mu(y)\rangle$ and there is no object of value at least $\langle mean_\nu(y)\rangle$ in $\omega(l)$. Thus, by Definition 3.14 the boundary-set of $(J, l, \langle mean_\nu(y)\rangle)$ is $\{q\}$. So, Claim 2 is proved.

Now we prove the first claim. Due to what we proved above, $\nu(q) < \langle mean_\nu(y)\rangle \leq mean_\nu(y)$ where q is the last object of l. Thus, ν-values of all objects of l are less than $mean_\nu(y)$. Also, we are considering the case in which μ-values of all objects are less than $mean_\mu(y)$. Hence, by Lemma 3.1, $find(I, l, mean_\mu(y))$ and $find(J, l, mean_\nu(y))$ both equal END as there is no object of μ-value at least $mean_\mu(y)$ or ν-value at least $mean_\nu(y)$.

Case 2b: μ-values of all objects of $\omega(l)$ are greater than $mean_\mu(y)$ and so they are all greater than $\langle mean_\mu(y)\rangle$.

In this case, the same argument as the previous case (by exchanging all occurrences of "greater", "bigger", "$>$", "$\geq$", and "last" with "less", "smaller", "$<$", "$\leq$", and "first", respectively, and in addition, by ignoring the very last sentence of the argument of the previous subcase) proves that the second claim of the lemma is true and that μ-values and ν-values of all objects of $\omega(l)$ are greater than $mean_\mu(y)$ and $mean_\nu(y)$, respectively. Hence, the first object p of $\omega(l)$ is the first object of $\omega(l)$ of μ-value at least $mean_\mu(y)$ and also the first object of $\omega(l)$ of ν-value at least $mean_\nu(y)$. As a result, according to Lemma 3.1, $find(I, l, mean_\mu(y))$ and $find(J, l, mean_\nu(y))$ both equal p. So, Claim 1 is correct, as well.

Case 2c: μ-values of a positive number of objects of $\omega(l)$ are less than $mean_\mu(y)$ and μ-values of a positive number of objects of $\omega(l)$ are greater than $mean_\mu(y)$.

We first consider the biggest object p of l with a μ-value at most $mean_\mu(y)$ and the smallest object of l with μ-value at least $mean_\mu(y)$. Then, we show that p and q are

consecutive objects of l and $\mu(p) < mean_\mu(y) < \mu(q)$. Afterward, using this fact we then prove Claims 2 and 1 in that order.

We first prove that $\mu(p) < mean_\mu(y) < \mu(q)$ and that p and q are consecutive. As discussed earlier, since by assumption there is no object of μ-value $\langle mean_\mu(y)\rangle$ in $\omega(l)$ and μ-values of all objects are main values, there is no object of μ-value $mean_\mu(y)$ in $\omega(l)$, as well. So $\mu(p) \neq mean_\mu(y)$ and $\mu(q) \neq mean_\mu(y)$. Hence,

$$\mu(p) < mean_\mu(y) < \mu(q) \tag{4.4}$$

as p and q are defined as objects with μ-value at most $mean_\mu(y)$ and at least $mean_\mu(y)$, respectively. Also, it follows from the definitions of p and q that they are either consecutive objects or the same objects. Hence, as by Equation 4.4 p and q do not have the same values, p and q are consecutive.

Now we prove Claim 2 by showing that the boundary-sets of both $(I, l, \langle mean_\mu(y)\rangle)$ and $(J, l, \langle mean_\nu(y)\rangle)$ are $\{p, q\}$. Since $\mu(p) < mean_\mu(y)$ (Equation 4.4), by Observation 2.30 $\mu(p) \leq \langle mean_\mu(y)\rangle$. So, as there is no object of μ-value $\langle mean_\mu(y)\rangle$ in $\omega(l)$, $\mu(p) < \langle mean_\mu(y)\rangle$. Moreover, since by Equation 4.4 $mean_\mu(y) < \mu(q)$ and by definition $\langle mean_\mu(y)\rangle \leq mean_\mu(y)$, $\langle mean_\mu(y)\rangle < \mu(q)$. Thus, $\mu(p) < \langle mean_\mu(y)\rangle < \mu(q)$. As a result, since p and q are consecutive objects, the object p (the object q) is the biggest (smallest) object of value at most (at least) $\langle mean_\mu(y)\rangle$ and consequently by Definition 3.14 $\{p, q\}$ is the boundary-set of $(I, l, \langle mean_\mu(y)\rangle)$. Hence as by assumption the boundary-set of $(I, l, \langle mean_\mu(y)\rangle)$ is a subset of $\mathcal{S}$, p and q are in $\mathcal{S}$. So, as $\mu(p) < \langle mean_\mu(y)\rangle = mean_\mu(z)$, according to Lemma 4.1, $\nu(p) < mean_\nu(z)$. Furthermore, as $mean_\mu(z) = \langle mean_\mu(y)\rangle < \mu(q)$, by Lemma 4.1, $mean_\nu(z) < \nu(q)$. Hence, $\nu(p) < \langle mean_\nu(y)\rangle < \nu(q)$ since $mean_\nu(z) = \langle mean_\nu(y)\rangle$. Thus, as p and q are consecutive, the object p (the object q) is the biggest (the smallest) object of ν-value at most (at least) $\langle mean_\nu(y)\rangle$. Consequently by Definition 3.14 $\{p, q\}$ is the boundary-set of $(J, l, \langle mean_\nu(y)\rangle)$. So Claim 2 is proved.

Now we prove Claim 1. We proved that p and q are in $\mathcal{S}$. Hence, as by Equation 4.4 $\mu(p) < mean_\mu(y)$ and $mean_\mu(y) < \mu(q)$, by Lemma 4.1, $\nu(p) < mean_\nu(y)$ and $mean_\nu(y) < \nu(q)$. Therefore, $\nu(p) < mean_\nu(y) < \nu(q)$. Also, by Equation 4.4 $\mu(p) < mean_\mu(y) < \mu(q)$. So, as p and q are consecutive, q is the smallest object of μ-value at least $mean_\mu(y)$ and at

the same time, it is the smallest object of ν-value at least $mean_\nu(y)$. Hence, by Lemma 3.1, $find(I, l, mean_\mu(y)) = q$ and $find(J, l, mean_\nu(y)) = q$. Thus, Claim 1 is also correct.

Now we have proved both claims in all cases. □

Recall that, given an eliminating proof Π for an instance I, our plan is to prove that Π is an eliminating proof for any instance J that is $Visited(I, \Pi)$-identical with I and that the output of Π on I is the same as the output of Π on J. We prove this fact in the next theorem.

Theorem 4.5 *Suppose the query tree of an instance $I = (Q, \omega, \mu)$ has at most one speedy leaf. Also, suppose Π is an eliminating proof for I and $J = (Q, \omega, \nu)$ is an instance $Visited(I, \Pi)$-identical with I. Then, the following properties hold.*

(1) Π is an eliminating proof for J.

(2) $Visited(I, \Pi) = Visited(J, \Pi)$.

(3) The output of Π on I is the same as the output of Π on J.

Proof. Using induction on i, we prove following statements for every i, $0 \leq i \leq n$.

1. The sequence $\Pi^{(i)}$ is a valid sequence of eliminating steps for J.
2. Suppose $result(I, \Pi^{(i)}) = (\mathsf{e}_I^{(i)}, \mathsf{e}_{\mathsf{s}I}^{(i)}, \Phi_I^{(i)})$ and $result(J, \Pi^{(i)}) = (\mathsf{e}_J^{(i)}, \mathsf{e}_{\mathsf{s}J}^{(i)}, \Phi_J^{(i)})$. Then,
   (a) there is an (I, Π)-descriptor $y^{(i)}$ such that $mean_\mu(y^{(i)}) = \mathsf{e}_I^{(i)}$ and $mean_\nu(y^{(i)}) = \mathsf{e}_J^{(i)}$,
   (b) there is an (I, Π)-descriptor $z^{(i)}$ such that $mean_\mu(z^{(i)}) = \mathsf{e}_{\mathsf{s}I}^{(i)}$ and $mean_\nu(z^{(i)}) = \mathsf{e}_{\mathsf{s}J}^{(i)}$, and
   (c) $\Phi_I^{(i)} = \Phi_J^{(i)}$.
3. If $i > 0$, $Visited_i(I, \Pi) = Visited_i(J, \Pi)$.

Having proved these claims for every i, $0 \leq i \leq n$, we will prove that the three claims of the lemma are true. We now first prove the claims for the base case $i = 0$. By Definition 3.13, $\Pi^{(0)}$ is valid for J and $result(I, \Pi^{(0)}) = result(J, \Pi^{(0)}) = (-\infty, -\infty, \Phi_0)$ where Φ_0 is the

empty sequence. Thus, when $i = 0$, $\mathsf{e}_I^{(i)} = \mathsf{e}\mathsf{s}_I^{(i)} = \mathsf{e}_J^{(i)} = \mathsf{e}\mathsf{s}_J^{(i)} = -\infty$ and $\Phi_I^{(i)} = \Phi_J^{(i)} = \Phi_0$. So Claims 1 and 2(c) are proved for $i = 0$. As $-\infty$ is a *Visited*(I, Π)-descriptor (Definition 4.2) and by Definition 4.3 $\mathit{mean}_\mu(-\infty) = \mathit{mean}_\nu(-\infty) = -\infty = \mathsf{e}_I^{(i)} = \mathsf{e}\mathsf{s}_I^{(i)} = \mathsf{e}_J^{(i)} = \mathsf{e}\mathsf{s}_J^{(i)}$, Claims 2(a) and 2(b) are correct, as well. Finally as $i \not> 0$, Claim 3 is also true. Consequently, all claims are correct for the base case.

Given that the claims are correct for $i - 1$, $1 \leq i \leq n$, in order to prove them for i, we consider different types of eliminating functions, one by one. Before that, we state a number of trivial observations. Since Π is an eliminating proof for I, by Lemma 3.22 $\Pi^{(i)}$ and $\Pi^{(i-1)}$ are valid for I and by the first claim of the induction hypothesis we know that $\Pi^{(i-1)}$ is valid for J. Now we consider the results $C_I^{(i-1)} = (\mathsf{e}_I^{(i-1)}, \mathsf{e}\mathsf{s}_I^{(i-1)}, \Phi_I^{(i-1)})$ and $C_J^{(i-1)} = (\mathsf{e}_J^{(i-1)}, \mathsf{e}\mathsf{s}_J^{(i-1)}, \Phi_J^{(i-1)})$ of $\Pi^{(i-1)}$ on I and on J, respectively, and also the result $C_I^{(i)} = (\mathsf{e}_I^{(i)}, \mathsf{e}\mathsf{s}_I^{(i)}, \Phi_I^{(i)})$ of $\Pi^{(i)}$ on I. As $C_I^{(i)}$ is the result of $\Pi^{(i)}$ on I, by Definition 3.13,

$$\pi_i(\mu, C_I^{(i-1)}) = C_I^{(i)} = (\mathsf{e}_I^{(i)}, \mathsf{e}\mathsf{s}_I^{(i)}, \Phi_I^{(i)}). \tag{4.5}$$

Also, as by Definition 3.15, *Visited*(I, Π) is the union of *Visited*$_j(I, \Pi)$, for j, $1 \leq j \leq n$, we can observe that

$$\mathit{Visited}_i(I, \Pi) \subseteq \mathit{Visited}(I, \Pi). \tag{4.6}$$

Eliminating functions of the first type: First we consider the case in which $\pi_i = \pi_T^-$, for a sub-union tree T of $N(Q)$ and we prove all claims one by one. Since by induction $\Pi^{(i-1)}$ is valid for J and by Definition 3.8 $\pi_i(\nu, C_J^{(i-1)})$ is always defined, by Definition 3.13 $\Pi^{(i)}$ is valid for J and so the first claim is satisfied. By Definition 3.13, $\mathit{result}(J, \Pi^{(i)}) = \pi_i(\nu, C_J^{(i-1)})$. Now we define the eliminating configuration $C_J^{(i)} = (\mathsf{e}_J^{(i)}, \mathsf{e}\mathsf{s}_J^{(i)}, \Phi_J^{(i)})$ as follows.

$$\mathit{result}(J, \Pi^{(i)}) = \pi_i(\nu, C_J^{(i-1)}) = C_J^{(i)} = (\mathsf{e}_J^{(i)}, \mathsf{e}\mathsf{s}_J^{(i)}, \Phi_J^{(i)}). \tag{4.7}$$

The following lemma will be used to prove the statements in the second induction claim.

Lemma 4.6 *For every leaf l of T, $\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}) = \mathit{find}(I, l, \mathsf{e}_I^{(i-1)})$.*

Proof. Given a leaf l of T, by Definition 3.15, the boundary-set of $(I, l, \langle \mathsf{e}_I^{(i-1)} \rangle)$ is a subset of *Visited*$_i(I, \Pi)$ because $\pi_i = \pi_T^-$. Hence, as by the induction hypothesis for Claim 2a

$\mathsf{e}_I^{(i-1)} = \mathit{mean}_\mu(y^{(i-1)})$ and by Equation 4.6, $\mathit{Visited}_i(I,\Pi) \subseteq \mathit{Visited}(I,\Pi)$, the boundary-set of $(I, l, \langle \mathit{mean}_\mu(y^{(i-1)})\rangle)$ is a subset of $\mathit{Visited}(I,\Pi)$. So, by applying Lemma 4.4 for $\mathcal{S} = \mathit{Visited}(I,\Pi)$ we can conclude that $\mathit{find}(I, l, \mathit{mean}_\mu(y^{(i-1)})) = \mathit{find}(J, l, \mathit{mean}_\nu(y^{(i-1)}))$ as by the induction hypothesis for Claim 2a $y^{(i-1)}$ is a $\mathit{Visited}(I,\Pi)$-descriptor. Also, by induction, $\mathsf{e}_I^{(i-1)} = \mathit{mean}_\mu(y^{(i-1)})$ and $\mathsf{e}_J^{(i-1)} = \mathit{mean}_\nu(y^{(i-1)})$. Thus, $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})$ equals $\mathit{find}(J, l, \mathsf{e}_J^{(i-1)})$. □

Now we prove different statements of the second claim, one by one.

Claim 2(a). Since $\mathsf{e}_I^{(i)}$ and $\mathsf{e}_J^{(i)}$ by Definition 3.8 are equal to $\mathit{hmin}(\mu, C_I^{(i-1)}, T)$ and $\mathit{hmin}(\nu, C_J^{(i-1)}, T)$, respectively, to prove the claim it will suffice to find a $\mathit{Visited}(I,\Pi)$-descriptor $y^{(i)}$ such $\mathit{mean}_\mu(y^{(i)}) = \mathit{hmin}(\mu, C_I^{(i-1)}, T)$ and $\mathit{mean}_\nu(y^{(i)}) = \mathit{hmin}(\nu, C_J^{(i-1)}, T)$. We consider two cases distinguished in Definition 3.7, separately.

The first case is when for every leaf l of T, $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}) = \text{END}$. Then, as by Lemma 4.6 $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, l, \mathsf{e}_J^{(i-1)})$, $\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}) = \text{END}$, for any leaf l of T. Thus, by Lemma 3.1 for every object o of every leaf of T, $\mu(o) < \mathsf{e}_I^{(i-1)}$ and $\nu(o) < \mathsf{e}_J^{(i-1)}$. Hence, since all leaves of T are in $N(Q)$ (because T is a sub-union tree of $N(Q)$), by Definition 3.3 all objects of all leaves of T are both $C_I^{(i-1)}$-eliminated and $C_J^{(i-1)}$-eliminated. Therefore, by Definition 3.7, both $\mathit{hmin}(\mu, C_I^{(i-1)}, T)$ and $\mathit{hmin}(\nu, C_J^{(i-1)}, T)$ are ∞. So, if we define $y^{(i)} = \infty$, by Definition 4.2 $y^{(i)}$ is a $\mathit{Visited}(I,\Pi)$-descriptor and by Definition 4.3, $\mathit{mean}_\mu(y^{(i)}) = \mathit{mean}_\nu(y^{(i)}) = \infty = \mathit{hmin}(\mu, C_I^{(i-1)}, T) = \mathit{hmin}(\nu, C_J^{(i-1)}, T)$. Thus, the claim is proved in this case.

Now we consider the other case, in which the set $\mathcal{T}$ of all leaves l of T satisfying $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}) \neq \text{END}$ is not empty. So, there is a leaf u of T such that $\mathit{find}(I, u, \mathsf{e}_I^{(i-1)}) \neq \text{END}$ and thus by Lemma 4.6 $\mathit{find}(J, u, \mathsf{e}_J^{(i-1)}) \neq \text{END}$. Consequently, by Lemma 3.1 there is an object o of u of μ-value at least $\mathsf{e}_I^{(i-1)}$ and there is an object p of u of ν-value at least $\mathsf{e}_J^{(i-1)}$. Therefore, as u is a leaf of T and so u is in $N(Q)$, by Definition 3.3 o and p are $C_I^{(i-1)}$-remaining and $C_J^{(i-1)}$-remaining, respectively. Hence, by Definition 3.7,

$$\mathit{hmin}(\mu, C_I^{(i-1)}, T) = \mu\left(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})\right), \tag{4.8}$$

where v is a leaf l of $\mathcal{T}$ minimizing $\mu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}))$.

We now claim that if we define

$$y^{(i)} = \left(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)}), 0\right), \tag{4.9}$$

$y^{(i)}$ satisfies $\mathit{mean}_\mu(y^{(i)}) = \mathit{hmin}(\mu, C_I^{(i-1)}, T)$ and $\mathit{mean}_\nu(y^{(i)}) = \mathit{hmin}(\nu, C_J^{(i-1)}, T)$. Since by Lemma 4.6 $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, l, \mathsf{e}_J^{(i-1)})$, for any leaf l of T, and $\mathcal{T}$ is the set of all leaves l of T with $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}) \neq \text{END}$, $\mathcal{T}$ is also the set of all leaves l of T with $\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}) \neq \text{END}$. Hence, as we proved above that a leaf of T exists that has a $C_J^{(i-1)}$-remaining object p, by Definition 3.7,

$$\mathit{hmin}(\nu, C_J^{(i-1)}, T) = \nu\left(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)})\right), \tag{4.10}$$

where u is a leaf l of $\mathcal{T}$ minimizing $\nu(\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}))$.

Now we prove that $\nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)})) = \nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)}))$ and then using this result we show that $y^{(i)}$ satisfies the conditions $\mathit{mean}_\mu(y^{(i)}) = \mathit{hmin}(\mu, C_I^{(i-1)}, T)$ and $\mathit{mean}_\nu(y^{(i)}) = \mathit{hmin}(\nu, C_J^{(i-1)}, T)$. Assume to the contrary that $\nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)})) \neq \nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)}))$. We know $\nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)})) \geq \nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)}))$ as u is a leaf l of the set $\mathcal{T}$ minimizing $\nu(\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}))$. Thus, as we supposed $\nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)})) \neq \nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)}))$,

$$\nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)})) > \nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)})). \tag{4.11}$$

Since v and u are in $\mathcal{T}$ and by definition $\mathcal{T} \subseteq \mathit{leaves}(T)$, v and u are leaves of T. Moreover applying Lemma 4.6 for $l = v$ and $l = u$ yields the facts that

$$\mathit{find}(J, v, \mathsf{e}_J^{(i-1)}) = \mathit{find}(I, v, \mathsf{e}_I^{(i-1)}) \tag{4.12}$$

and $\nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)})) = \nu(\mathit{find}(I, u, \mathsf{e}_I^{(i-1)}))$, respectively. Hence, we can rewrite Equation 4.11 as $\nu(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})) > \nu(\mathit{find}(I, u, \mathsf{e}_I^{(i-1)}))$. As a result, ν satisfies the comparison proposition $\mathit{find}(I, v, \mathsf{e}_I^{(i-1)}) > \mathit{find}(I, u, \mathsf{e}_I^{(i-1)})$. Also, as u and v in $\mathcal{T}$, neither $\mathit{find}(I, u, \mathsf{e}_I^{(i-1)})$ nor $\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})$ equal END. Hence, as u and v are leaves of T, by Definition 3.15 $\mathit{find}(I, u, \mathsf{e}_I^{(i-1)})$ and $\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})$ are in $\mathit{Visited}_i(I, \Pi)$ and so they are in $\mathit{Visited}(I, \Pi)$, due to Equation 4.6. Hence, as I and J are $\mathit{Visited}(I, \Pi)$-identical, by

Definition 2.15 the comparison proposition $\mathit{find}(I, v, \mathsf{e}_I^{(i-1)}) > \mathit{find}(I, u, \mathsf{e}_I^{(i-1)})$ is satisfied by μ as it is satisfied by ν. Therefore, $\mu(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})) > \mu(\mathit{find}(I, u, \mathsf{e}_I^{(i-1)}))$. Since u is in $\mathcal{T}$, this contradicts the fact that v is a leaf l of $\mathcal{T}$ minimizing $\mu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}))$. Hence the assumption that $\nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)})) \neq \nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)}))$ is wrong and thus

$$\nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)})) = \nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)})). \tag{4.13}$$

Now due to Equations 4.10, 4.13, and 4.12 we can write

$$\begin{aligned} \mathit{hmin}(\nu, C_J^{(i-1)}, T) &= \nu(\mathit{find}(J, u, \mathsf{e}_J^{(i-1)})) \\ &= \nu(\mathit{find}(J, v, \mathsf{e}_J^{(i-1)})) \\ &= \nu(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})). \end{aligned} \tag{4.14}$$

Also, since $y^{(i)} = \left(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)}), 0\right)$ (Equation 4.9) by Definition 4.3

$$\mathit{mean}_\nu(y^{(i)}) = \nu(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})) \tag{4.15}$$

and

$$\mathit{mean}_\mu(y^{(i)}) = \mu(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)})). \tag{4.16}$$

Now, by Equations 4.14 and 4.15, $\mathit{mean}_\nu(y^{(i)}) = \mathit{hmin}(\nu, C_J^{(i-1)}, T)$ and by Equations 4.8 and 4.16 we can write $\mathit{mean}_\mu(y^{(i)}) = \mu(\mathit{find}(I, v, \mathsf{e}_I^{(i-1)}))$. This result proves Claim 2(a).

Claim 2(b). We consider two cases based on Q having a speedy leaf or not. First suppose Q does not have a speedy leaf. By Equations 4.5 and 4.7 $(\mathsf{e}_I^{(i)}, \mathsf{e}_{\mathsf{s}I}^{(i)}, \Phi_I^{(i)}) = \pi_i(\mu, C_I^{(i-1)})$ and $(\mathsf{e}_J^{(i)}, \mathsf{e}_{\mathsf{s}J}^{(i)}, \Phi_J^{(i)}) = \pi_i(\nu, C_J^{(i-1)})$. So, as by assumption $\pi_i = \pi_T^-$ and Q has no speedy leaf, by Definition 3.8 $\mathsf{e}_{\mathsf{s}I}^{(i)} = \mathsf{e}_I^{(i)}$ and $\mathsf{e}_{\mathsf{s}J}^{(i)} = \mathsf{e}_J^{(i)}$. Hence as by proving Claim 2(a) we have proved $\mathit{mean}_\mu(y^{(i)}) = \mathsf{e}_I^{(i)}$ and $\mathit{mean}_\nu(y^{(i)}) = \mathsf{e}_J^{(i)}$, for the choice $z^{(i)} = y^{(i)}$ we have $\mathit{mean}_\mu(z^{(i)}) = \mathsf{e}_{\mathsf{s}I}^{(i)}$ and $\mathit{mean}_\nu(z^{(i)}) = \mathsf{e}_{\mathsf{s}J}^{(i)}$. If Q has a speedy leaf, according to Definition 3.8, $\mathsf{e}_{\mathsf{s}I}^{(i)} = \mathsf{e}_{\mathsf{s}I}^{(i-1)}$ and $\mathsf{e}_{\mathsf{s}J}^{(i)} = \mathsf{e}_{\mathsf{s}J}^{(i-1)}$ and by the induction hypothesis for Claim 2b we have $\mathit{mean}_\mu(z^{(i-1)}) = \mathsf{e}_{\mathsf{s}I}^{(i-1)}$ and $\mathit{mean}_\nu(z^{(i-1)}) = \mathsf{e}_{\mathsf{s}J}^{(i-1)}$. Hence, the choice $z^{(i)} = z^{(i-1)}$ satisfies $\mathit{mean}_\mu(z^{(i)}) = \mathsf{e}_{\mathsf{s}I}^{(i)}$ and $\mathit{mean}_\nu(z^{(i)}) = \mathsf{e}_{\mathsf{s}J}^{(i)}$.

Claim 2(c). Since by Equation 4.5 $(\mathsf{e}_I^{(i)}, \mathsf{e}_{\mathsf{s}I}^{(i)}, \Phi_I^{(i)}) = \pi_i(\mu, C_I^{(i-1)})$ and by Equation 4.7 $(\mathsf{e}_J^{(i)}, \mathsf{e}_{\mathsf{s}J}^{(i)}, \Phi_J^{(i)}) = \pi_i(\nu, C_J^{(i-1)})$ and also by assumption $\pi_i = \pi_T^-$, by Definition 3.8 $\Phi_I^{(i-1)} = \Phi_I^{(i)}$ and $\Phi_J^{(i)} = \Phi_J^{(i-1)}$. Hence, as by the induction hypothesis for Claim 2c $\Phi_J^{(i-1)} = \Phi_I^{(i-1)}$, $\Phi_J^{(i)} = \Phi_I^{(i)}$.

Claim 3. Since by assumption $\pi_i = \pi_T^-$, by Definition 3.15,

$$\mathit{Visited}_i(I, \Pi) = \mathcal{S}_{I,l} \cup \bigcup_{l \in \mathit{leaves}(T)} \{\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})\} - \{\text{END}\}$$

and

$$\mathit{Visited}_i(J, \Pi) = \mathcal{S}_{J,l} \cup \bigcup_{l \in \mathit{leaves}(T)} \{\mathit{find}(J, l, \mathsf{e}_J^{(i-1)})\} - \{\text{END}\}$$

where $\mathcal{S}_{I,l}$ and $\mathcal{S}_{J,l}$ are boundary-sets of $(I, l, \langle \mathsf{e}_I^{(i-1)} \rangle)$ and $(J, l, \langle \mathsf{e}_J^{(i-1)} \rangle)$, respectively, for every leaf l of T. According to Lemma 4.6, $\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}) = \mathit{find}(I, l, \mathsf{e}_I^{(i-1)})$, for any leaf l of T. So, to prove $\mathit{Visited}_i(I, \Pi) = \mathit{Visited}_j(I, \Pi)$, it suffices to prove that boundary-sets of $(I, l, \langle \mathsf{e}_I^{(i-1)} \rangle)$ and $(J, l, \langle \mathsf{e}_J^{(i-1)} \rangle)$ are the same, for any leaf l of T. Given a leaf l of T, by Definition 3.15, the boundary-set of $(I, l, \langle \mathsf{e}_I^{(i-1)} \rangle)$ is subset of $\mathit{Visited}_i(I, \Pi)$ and so it is a subset of $\mathit{Visited}(I, \Pi)$ as by Equation 4.6, $\mathit{Visited}_i(I, \Pi) \subseteq \mathit{Visited}(I, \Pi)$. Furthermore, by the induction hypothesis for Claim 2a $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$. Thus, the boundary-set of $(I, l, \langle \mathit{mean}_\mu(y^{(i-1)}) \rangle)$ is subset of $\mathit{Visited}(I, \Pi)$. Hence, by applying Lemma 4.4 for $\mathcal{S} = \mathit{Visited}(I, \Pi)$, boundary-sets of $(I, l, \langle \mathit{mean}_\mu(y^{(i-1)}) \rangle)$ and $(J, l, \langle \mathit{mean}_\nu(y^{(i-1)}) \rangle)$ are the same. Also, by the induction hypothesis for Claim 2a $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ and $\mathit{mean}_\nu(y^{(i-1)}) = \mathsf{e}_J^{(i-1)}$. So, boundary-sets of $(I, l, \langle \mathsf{e}_I^{(i-1)} \rangle)$ and $(J, l, \langle \mathsf{e}_J^{(i-1)} \rangle)$ are the same. As a result, $\mathit{Visited}_i(I, \Pi) = \mathit{Visited}_i(J, \Pi)$.

Before proving the cases in which π_i is of the second or of the third type, since the definition of the output of the eliminating configuration resulting from each of these two types is based on the function Φ_{sl}, we first prove that Φ_{sl} generates the same output when it is applied to I or to J.

Lemma 4.7 *Suppose Q has no speedy leaf or if it does have one, $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)})$. Then, $\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)})$.*

Proof. We prove the lemma by considering two possibilities based on Q having a speedy leaf or not. By Definition 3.9, if Q has no speedy leaf,

$$\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_I^{(i-1)}$$

and

$$\Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)}) = \Phi_J^{(i-1)}.$$

So as by the induction hypothesis for Claim 2c $\Phi_I^{(i-1)} = \Phi_J^{(i-1)}$, $\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)})$.

To prove the lemma in the case in which there is a speedy leaf, we first prove that $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}I}^{(i-1)}) = \mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}J}^{(i-1)})$ in the two cases $\mathsf{e}_{\mathsf{s}I}^{(i-1)} \in \{-\infty, \infty\}$ and $\mathsf{e}_{\mathsf{s}I}^{(i-1)} \notin \{-\infty, \infty\}$, separately, and then we complete the proof of the lemma using this fact.

First consider the case $\mathsf{e}_{\mathsf{s}I}^{(i-1)} \in \{-\infty, \infty\}$. By the induction hypothesis for Claim 2b $z^{(i-1)}$ is a $\mathit{Visited}(I, \Pi)$-descriptor satisfying $\mathit{mean}_\mu(z^{(i-1)}) = \mathsf{e}_{\mathsf{s}I}^{(i-1)}$. Hence, $z^{(i-1)}$ is not of the form (o, j) since otherwise by Definition 4.3 $\langle \mathit{mean}_\mu(z^{(i-1)}) \rangle$ was the value of the object o while $\mathit{mean}_\mu(z^{(i-1)}) = \mathsf{e}_{\mathsf{s}I}^{(i-1)}$ and we supposed $\mathsf{e}_{\mathsf{s}I}^{(i-1)} \in \{-\infty, \infty\}$. Therefore, by Definition 4.2 $z^{(i-1)}$ is $-\infty$ or ∞. In the case $z^{(i-1)} = \infty$, as by the induction hypothesis for Claim 2b $\mathit{mean}_\mu(z^{(i-1)}) = \mathsf{e}_{\mathsf{s}I}^{(i-1)}$ and $\mathit{mean}_\nu(z^{(i-1)}) = \mathsf{e}_{\mathsf{s}J}^{(i-1)}$, by Definition 4.3 $\mathsf{e}_{\mathsf{s}I}^{(i-1)} = \mathit{mean}_\mu(\infty) = \infty$ and $\mathsf{e}_{\mathsf{s}J}^{(i-1)} = \mathit{mean}_\nu(\infty) = \infty$. So, as μ-values and ν-values of all objects of $\mathit{sl}(Q)$ are less than $\infty = \mathsf{e}_{\mathsf{s}I}^{(i-1)} = \mathsf{e}_{\mathsf{s}J}^{(i-1)}$, by Lemma 3.1 $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}I}^{(i-1)})$ and $\mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}J}^{(i-1)})$ both equal END. In the case $z^{(i-1)} = -\infty$ similar to the above argument WE can write $\mathsf{e}_{\mathsf{s}I}^{(i-1)} = \mathit{mean}_\mu(-\infty) = -\infty$ and $\mathsf{e}_{\mathsf{s}J}^{(i-1)} = \mathit{mean}_\nu(-\infty) = -\infty$. Hence, as μ-values and ν-values of all objects of $\mathit{sl}(Q)$ are at least $-\infty = \mathsf{e}_{\mathsf{s}I}^{(i-1)} = \mathsf{e}_{\mathsf{s}J}^{(i-1)}$, by Lemma 3.1 $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}I}^{(i-1)})$ and $\mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}J}^{(i-1)})$ both are equal to the first object of the speedy leaf. So, in either case, $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}I}^{(i-1)}) = \mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_{\mathsf{s}J}^{(i-1)})$.

Now we consider the other case, that is, $\mathsf{e}_{\mathsf{s}I}^{(i-1)} \notin \{-\infty, \infty\}$. By the induction hypothesis for Claim 2b $z^{(i-1)}$ is a $\mathit{Visited}(I, \Pi)$-descriptor satisfying $\mathit{mean}_\mu(z^{(i-1)}) = \mathsf{e}_{\mathsf{s}I}^{(i-1)}$. Also, as $\mathsf{e}_{\mathsf{s}I}^{(i-1)} \notin \{-\infty, \infty\}$, by Lemma 3.24 the boundary-set of $(I, \mathit{sl}(Q), \langle \mathsf{e}_{\mathsf{s}I}^{(i-1)} \rangle)$ is a subset of $\mathit{Visited}(I, \Pi)$. Hence, the boundary-set of $(I, \mathit{sl}(Q), \langle \mathit{mean}_\mu(z^{(i-1)}) \rangle)$ is a subset of $\mathit{Visited}(I, \Pi)$. Consequently, as by the induction hypothesis for Claim 2b $z^{(i-1)}$ is a $\mathit{Visited}(I, \Pi)$-descriptor, applying Lemma 4.4 for $\mathcal{S} = \mathit{Visited}(I, \Pi)$, we can conclude that

$\mathit{find}(I, \mathit{sl}(Q), \mathit{mean}_\mu(z^{(i-1)})) = \mathit{find}(J, \mathit{sl}(Q), \mathit{mean}_\nu(z^{(i-1)}))$. Thus, as by the induction hypothesis for Claim 2b $\mathit{mean}_\mu(z^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ and $\mathit{mean}_\nu(z^{(i-1)}) = \mathsf{e}_J^{(i-1)}$, in all cases we have proved

$$\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)}). \tag{4.17}$$

Now we complete the proof using the above equation. By Lemma 3.4, Equation 4.17 yields the fact that

$$\mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{pos}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)}). \tag{4.18}$$

Also, as we are considering the case in which Q has a speedy leaf, by the assumption of the lemma $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)})$ and so again by Lemma 3.4

$$\mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{pos}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)}). \tag{4.19}$$

Now we consider two cases based on the equation $\mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$ being true or not. First suppose it is correct. Then,

$$\begin{aligned}
\mathit{pos}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)}) &= \mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) && \text{by Equation 4.18} \\
&= \mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) && \text{by assumption} \\
&= \mathit{pos}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)}) && \text{by Equation 4.19.}
\end{aligned}$$

As by assumption $\mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$, by Definition 3.9,

$$\Phi_{\mathit{sl}}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_I^{(i-1)}$$

and as we proved $\mathit{pos}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)}) = \mathit{pos}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)})$, by Definition 3.9,

$$\Phi_{\mathit{sl}}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)}) = \Phi_J^{(i-1)}.$$

So as by the induction hypothesis for Claim 2c $\Phi_I^{(i-1)} = \Phi_J^{(i-1)}$, $\Phi_{\mathit{sl}}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_{\mathit{sl}}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)})$.

Now we consider the other case, in which $\mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) \neq \mathit{pos}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$.

Then by Definition 3.9,

$$\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_I^{(i-1)} \odot \Big(sl(Q), pos(I, sl(Q), \mathsf{e}\mathsf{s}_I^{(i-1)}), pos(I, sl(Q), \mathsf{e}_I^{(i-1)}) - 1\Big). \tag{4.20}$$

Now, since $pos(I, sl(Q), \mathsf{e}\mathsf{s}_I^{(i-1)}) \neq pos(I, sl(Q), \mathsf{e}_I^{(i-1)})$, because of Equations 4.18 and 4.19

$$pos(J, sl(Q), \mathsf{e}\mathsf{s}_J^{(i-1)}) \neq pos(J, sl(Q), \mathsf{e}_J^{(i-1)})$$

and thus by Definition 3.9,

$$\begin{aligned}
&\Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)}) \\
&= \Phi_J^{(i-1)} \odot \Big(sl(Q), pos(J, sl(Q), \mathsf{e}\mathsf{s}_J^{(i-1)}), pos(J, sl(Q), \mathsf{e}_J^{(i-1)}) - 1\Big) \\
&= \Phi_I^{(i-1)} \odot \Big(sl(Q), pos(J, sl(Q), \mathsf{e}\mathsf{s}_J^{(i-1)}), pos(J, sl(Q), \mathsf{e}_J^{(i-1)}) - 1\Big) && \text{by the induction hypothesis for Claim 2c} \\
&= \Phi_I^{(i-1)} \odot \Big(sl(Q), pos(I, sl(Q), \mathsf{e}\mathsf{s}_I^{(i-1)}), pos(J, sl(Q), \mathsf{e}_J^{(i-1)}) - 1\Big) && \text{by Equation 4.18} \\
&= \Phi_I^{(i-1)} \odot \Big(sl(Q), pos(I, sl(Q), \mathsf{e}\mathsf{s}_I^{(i-1)}), pos(I, sl(Q), \mathsf{e}_I^{(i-1)}) - 1\Big) && \text{by Equation 4.19} \\
&= \Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) && \text{by Equation 4.20.}
\end{aligned}$$

□

Eliminating functions of the second type: Next we suppose $\pi_i = \pi_T^+$, for a sub-intersection tree T of Q, and we prove the induction claims for i.

First we prove Claim 1 by showing that every leaf of T has an object of ν-value $\mathsf{e}_J^{(i-1)}$ and that $\mathsf{e}_J^{(i-1)}$ is a main value. Because by assumption Π is an eliminating proof for I, by Lemma 3.22 $\Pi^{(i)}$ is valid for I and hence by Definition 3.13 $\pi_i(\mu, C_I^{(i-1)})$ is defined. So, as by definition $C_I^{(i-1)} = (\mathsf{e}_I^{(i-1)}, \mathsf{e}\mathsf{s}_I^{(i-1)}, \Phi_I^{(i-1)})$, by Definition 3.10 $\omega(l)$ has an object of μ-value $\mathsf{e}_I^{(i-1)}$, for any leaf l of T. Thus, by Lemma 3.2 $find(I, l, \mathsf{e}_I^{(i-1)})$ is an object of

μ-value $\mathsf{e}_I^{(i-1)}$. Therefore, by the induction hypothesis for Claim 2a,

$$\mu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})) = \mathsf{e}_I^{(i-1)} = \mathit{mean}_\mu(y^{(i-1)}), \tag{4.21}$$

for any leaf l of T. Furthermore, since by Definition 3.15 $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})$ is in $\mathit{Visited}_i(I, \Pi)$, and by Equation 4.6 $\mathit{Visited}_i(I, \Pi) \subseteq \mathit{Visited}(I, \Pi)$, $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})$ is in $\mathit{Visited}(I, \Pi)$. So, as we proved $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})$ is an object and by Equation 4.21 $\mu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})) = \mathit{mean}_\mu(y^{(i-1)})$, by Lemma 4.1 for $\mathcal{S} = \mathit{Visited}(I, \Pi)$, $\nu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})) = \mathit{mean}_\nu(y^{(i-1)})$, for any leaf l of T. Hence, since by the induction hypothesis for Claim 2a $\mathit{mean}_\nu(y^{(i-1)}) = \mathsf{e}_J^{(i-1)}$

$$\nu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})) = \mathsf{e}_J^{(i-1)}, \tag{4.22}$$

for any leaf l of T. As a result every leaf of T has an object of ν-value $\mathsf{e}_J^{(i-1)}$. Thus by Definition 2.5 $\mathsf{e}_J^{(i-1)}$ is in $\mathbb{V}$ and hence $\mathsf{e}_J^{(i-1)}$ is a main value. So, by Definition 3.10, $\pi_T^+(\mu, C_J^{(i-1)})$ is defined. Also, it follows from Claim 1 for $i-1$ that $\Pi^{(i-1)}$ is valid for J. Therefore, by Definition 3.13 $\Pi^{(i)}$ is valid for J and hence the first claim is true. Now we define the eliminating configuration $C_J^{(i)}$ to be

$$\mathit{result}(J, \Pi^{(i)}) = \pi_i(\nu, C_J^{(i-1)}) = C_J^{(i)} = (\mathsf{e}_J^{(i)}, \mathsf{e}_{\mathsf{s}J}^{(i)}, \Phi_J^{(i)}). \tag{4.23}$$

Now we prove that $y^{(i-1)} = (o, 0)$ for an object of o in $\mathit{Visited}(I, \Pi)$ satisfying $\mu(o) = \mathsf{e}_I^{(i-1)}$ and $\nu(o) = \mathsf{e}_J^{(i-1)}$ and then we show that the choice $y^{(i)} = z^{(i)} = (o, 1)$ proves Claims 2(a) and 2(b). By the induction hypothesis for Claim 2a, $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ and as we proved in the previous paragraph, every leaf l of T has an object of μ-value $\mathsf{e}_I^{(i-1)}$. Hence, $\mathit{mean}_\mu(y^{(i-1)})$ is the value of an object and so it is not equal to $-\infty$ or ∞ and it is not a skirted value. Thus, it follows from Definition 4.3 that $y^{(i-1)}$ is not equal to $-\infty$ or ∞ and is not of the form $(o, 1)$. Hence by Definition 4.2, $y^{(i-1)}$ equals $(o, 0)$, for an object o in $\mathit{Visited}(I, \Pi)$ since by the induction hypothesis for Claim 2a $y^{(i-1)}$ is a $\mathit{Visited}(I, \Pi)$-descriptor. So, $\mathit{mean}_\mu(y^{(i-1)}) = \mu(o)$ and $\mathit{mean}_\nu(y^{(i-1)}) = \nu(o)$ while by the induction hypothesis for Claim 2a $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ and $\mathit{mean}_\nu(y^{(i-1)}) = \mathsf{e}_J^{(i-1)}$. Therefore, $\mu(o) = \mathsf{e}_I^{(i-1)}$ and $\nu(o) = \mathsf{e}_J^{(i-1)}$. Also, by Definition 4.2 o is an object of $\mathit{Visited}(I, \Pi)$ as $y = (o, 0)$ is a $\mathit{Visited}(I, \Pi)$-descriptor.

Now we prove Claims 2(a) and 2(b). As $o \in \mathit{Visited}(I,\Pi)$, by Definition 4.2 $(o,1)$ is a $\mathit{Visited}(I,\Pi)$-descriptor and by Definition 4.3 $\mathit{mean}_\mu((o,1)) = \mu(o)_+ = \mathsf{e}_I^{(i-1)}{}_+$ and $\mathit{mean}_\nu((o,1)) = \nu(o)_+ = \mathsf{e}_J^{(i-1)}{}_+$. Since by Equation 4.5 $\pi_i(\mu, C_I^{(i-1)}) = (\mathsf{e}_I^{(i)}, \mathsf{e_s}_I^{(i)}, \Phi_I^{(i)})$ and by assumption $\pi_i = \pi_T^+$, by Definition 3.10, $\mathsf{e}_I^{(i-1)}{}_+ = \mathsf{e}_I^{(i)} = \mathsf{e_s}_I^{(i)}$. Also, as by Equation 4.23 $\pi_i(\nu, C_J^{(i-1)}) = (\mathsf{e}_J^{(i)}, \mathsf{e_s}_J^{(i)}, \Phi_J^{(i)})$, by Definition 3.10, $\mathsf{e}_J^{(i-1)}{}_+ = \mathsf{e}_J^{(i)} = \mathsf{e_s}_J^{(i)}$. Hence, if we define $y^{(i)} = z^{(i)} = (o,1)$, then $y^{(i)}$ and $z^{(i)}$ are $\mathit{Visited}(I,\Pi)$-descriptors satisfying $\mathit{mean}_\mu(y^{(i)}) = \mathsf{e}_I^{(i-1)}{}_+ = \mathsf{e}_I^{(i)}$, $\mathit{mean}_\nu(y^{(i)}) = \mathsf{e}_J^{(i-1)}{}_+ = \mathsf{e}_J^{(i)}$, $\mathit{mean}_\mu(z^{(i)}) = \mathsf{e}_I^{(i-1)}{}_+ = \mathsf{e_s}_I^{(i)}$, and $\mathit{mean}_\nu(z^{(i)}) = \mathsf{e}_J^{(i-1)}{}_+ = \mathsf{e_s}_J^{(i)}$.

Next we prove Claim 2(c). We will use Lemma 4.7 to prove that $\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)})$ and then we show that $\mathit{pos}(I, l_T, \mathsf{e}_I^{(i-1)}) = \mathit{pos}(J, l_T, \mathsf{e}_J^{(i-1)})$ where l_T is defined in Definition 3.10. Finally using these two results and Definition 3.10 we will show the correctness of Claim 2(c).

To prove that $\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)})$, we first show that if Q has a speedy leaf then $\mathit{find}(I, \mathit{sl}(Q), \mathit{mean}_\mu(y^{(i-1)})) = \mathit{find}(J, \mathit{sl}(Q), \mathit{mean}_\nu(y^{(i-1)}))$ and after that we apply Lemma 4.7. By the induction hypothesis for Claim 2a $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ and $\mathit{mean}_\nu(y^{(i-1)}) = \mathsf{e}_J^{(i-1)}$. Furthermore, if Q has a speedy leaf, since by assumption $\pi_i = \pi_T^+$, by Definition 3.15 the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$ is a subset of $\mathit{Visited}_i(I,\Pi)$ and hence it is a subset of $\mathit{Visited}(I,\Pi)$ as by Equation 4.6 $\mathit{Visited}_i(I,\Pi) \subseteq \mathit{Visited}(I,\Pi)$. Also, as we proved, $\mathsf{e}_I^{(i-1)}$ is a main value and so $\mathsf{e}_I^{(i-1)} = \langle \mathsf{e}_I^{(i-1)} \rangle$. Therefore, the boundary-set of $(I, \mathit{sl}(Q), \langle \mathsf{e}_I^{(i-1)} \rangle)$ is a subset of $\mathit{Visited}(I,\Pi)$. Thus, as $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ the boundary-set of $(I, \mathit{sl}(Q), \langle \mathit{mean}_\mu(y^{(i-1)}) \rangle)$ is a subset of $\mathit{Visited}(I,\Pi)$. As a result, by Lemma 4.4,

$$\mathit{find}(I, \mathit{sl}(Q), \mathit{mean}_\mu(y^{(i-1)})) = \mathit{find}(J, \mathit{sl}(Q), \mathit{mean}_\nu(y^{(i-1)})). \tag{4.24}$$

Consequently, as by the induction hypothesis for Claim 2a $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ and $\mathit{mean}_\nu(y^{(i-1)}) = \mathsf{e}_J^{(i-1)}$, we can conclude that $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$ is equal to $\mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)})$ if Q has a speedy leaf. So, by Lemma 4.7

$$\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)}). \tag{4.25}$$

Now we prove that $\mathit{pos}(I, l_T, \mathsf{e}_I^{(i-1)}) = \mathit{pos}(J, l_T, \mathsf{e}_J^{(i-1)})$ by first showing that for any

leaf l of T, $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, l, \mathsf{e}_J^{(i-1)})$ and then by applying Lemma 3.4. Since by Equation 4.22 $\nu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})) = \mathsf{e}_J^{(i-1)}$, l has an object of ν-value $\mathsf{e}_J^{(i-1)}$, for any leaf l of T. Thus, by Lemma 3.2 $\nu(\mathit{find}(J, l, \mathsf{e}_J^{(i-1)})) = \mathsf{e}_J^{(i-1)}$, for any leaf l of T. As a result, $\nu(\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})) = \nu(\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}))$. Therefore, as l does not have two distinct objects of the same ν-value,

$$\mathit{find}(J, l, \mathsf{e}_J^{(i-1)}) = \mathit{find}(I, l, \mathsf{e}_I^{(i-1)}), \tag{4.26}$$

for any leaf l of T. So, as by Definition 3.10 for arbitrary leaf l_T of T, Equation 4.26 holds for $l = l_T$ and thus by Lemma 3.4

$$\mathit{pos}(J, l_T, \mathsf{e}_J^{(i-1)}) = \mathit{pos}(I, l_T, \mathsf{e}_I^{(i-1)}). \tag{4.27}$$

Now having proved Equations 4.25 and 4.27, we can use Definition 3.10 to write

$$\begin{aligned}
\Phi_I^{(i)} &= \Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) \odot (l_T, \mathit{pos}(I, l_T, \mathsf{e}_I^{(i-1)}), \mathit{pos}(I, l_T, \mathsf{e}_I^{(i-1)})) \\
&= \Phi_{sl}(C_J^{(i-1)}, \mu, \mathsf{e}_J^{(i-1)}) \odot (l_T, \mathit{pos}(J, l_T, \mathsf{e}_J^{(i-1)}), \mathit{pos}(J, l_T, \mathsf{e}_J^{(i-1)})) \\
&= \Phi_J^{(i)}.
\end{aligned}$$

It now remains to prove Claim 3, that is, to show that $\mathit{Visited}_i(I, \Pi) = \mathit{Visited}_i(J, \Pi)$. As by assumption $\pi_i = \pi_T^+$, according to Definition 3.15,

$$\mathit{Visited}_i(I, \Pi) = \bigcup_{l \in \mathit{leaves}(T)} \{\mathit{find}(I, l, \mathsf{e}_I^{(i-1)})\} \cup \mathcal{S}$$

and

$$\mathit{Visited}_i(J, \Pi) = \bigcup_{l \in \mathit{leaves}(T)} \{\mathit{find}(J, l, \mathsf{e}_J^{(i-1)})\} \cup \mathcal{T}$$

where $\mathcal{S}$ and $\mathcal{T}$ are empty if Q has no speedy leaf and otherwise they are boundary-sets of $(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$ and $(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)})$, respectively. Since by Equation 4.26 $\mathit{find}(I, l, \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, l, \mathsf{e}_J^{(i-1)})$, for every leaf l of T, it suffices to prove that $\mathcal{S} = \mathcal{T}$, that is, to prove that if Q has a speedy leaf, boundary-sets of $(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$ and $(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)})$ are the same.

Since $\pi_i = \pi_T^+$, by Definition 3.15, the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$ is a subset of $\mathit{Visited}_i(I, \Pi)$ and hence it is a subset of $\mathit{Visited}(I, \Pi)$ as by Equation 4.6 $\mathit{Visited}_i(I, \Pi) \subseteq \mathit{Visited}(I, \Pi)$. Also, as we proved that $\pi_T^+(\mu, C_I^{(i-1)})$ is defined, by Definition 3.10, $\mathsf{e}_I^{(i-1)}$ is a

main value and hence $\langle \mathsf{e}_I^{(i-1)} \rangle = \mathsf{e}_I^{(i-1)}$. Moreover, by the induction hypothesis for Claim 2a $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$ and hence $\mathsf{e}_I^{(i-1)} = \langle \mathsf{e}_I^{(i-1)} \rangle = \langle \mathit{mean}_\mu(y^{(i-1)}) \rangle$. So, the boundary-set of $(I, \mathit{sl}(Q), \langle \mathit{mean}_\mu(y^{(i-1)}) \rangle)$ is a subset of $\mathit{Visited}(I, \Pi)$. Consequently, applying Lemma 4.4 for $\mathcal{S} = \mathit{Visited}(I, \Pi)$ we can conclude that the boundary-sets of $(I, \mathit{sl}(Q), \langle \mathit{mean}_\mu(y^{(i-1)}) \rangle)$ and $(J, \mathit{sl}(Q), \langle \mathit{mean}_\nu(y^{(i-1)}) \rangle)$ are the same. Also exactly the same argument that is used above to prove that $\mathsf{e}_I^{(i-1)} = \langle \mathit{mean}_\mu(y^{(i-1)}) \rangle$ can be implied (by replacing occurrences of "I" and "μ" with "J" and "ν", respectively) to show that $\mathsf{e}_J^{(i-1)} = \langle \mathit{mean}_\nu(y^{(i-1)}) \rangle$. Hence, the boundary-sets of $(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)})$ and $(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)})$ are the same. So, $\mathit{Visited}_i(I, \Pi) = \mathit{Visited}_i(J, \Pi)$, as explained.

Eliminating functions of the third type : Finally, we consider the case in which $\pi_i = \pi^\infty$ and we prove the claims one by one. We use the next lemma to prove that $\mathsf{e}_J^{(i)} = \infty$ and then we conclude that $\Pi^{(i)}$ is valid for J.

Lemma 4.8 *Given a* $\mathit{Visited}(I, \Pi)$*-descriptor* y*, if* $\mathit{mean}_\mu(y) = \infty$ *then* $\mathit{mean}_\nu(y) = \infty$.

Proof. We first show that $y = \infty$ and then we conclude that $\mathit{mean}_\nu(y) = \infty$. If y is of the form (o, j), for an object o and an integer j, then by Definition 4.3 $\mathit{mean}_\mu(y) = \mu(o)$ or $\mathit{mean}_\mu(y) = \mu(o)_+$ and in both cases we have $\langle \mathit{mean}_\mu(y) \rangle = \mu(o)$. So, as $\langle \mathit{mean}_\mu(y) \rangle = \langle \infty \rangle = \infty$ is not the μ-value of any object (Definition 2.5), y is not of the form (o, j). Also, $y \neq -\infty$ as otherwise by Definition 4.3 $\mathit{mean}_\mu(y) = -\infty$. So by Definition 4.2, $y = \infty$ and hence by Definition 4.3 $\mathit{mean}_\nu(y) = \infty$. □

We now prove the first claim. Since Π_i is valid for I, by Definition 3.13 $\pi_i(\mu, C_I^{(i-1)})$ is defined and hence by Definition 3.11,

$$\mathsf{e}_I^{(i-1)} = \infty. \tag{4.28}$$

Thus, $\mathit{mean}_\mu(y^{(i-1)}) = \infty$ as by the induction hypothesis for Claim 2a $\mathit{mean}_\mu(y^{(i-1)}) = \mathsf{e}_I^{(i-1)}$. Hence by Lemma 4.8 $\mathit{mean}_\nu(y^{(i-1)}) = \infty$. So, by the induction hypothesis for Claim 2a $\mathit{mean}_\nu(y^{(i-1)}) = \mathsf{e}_J^{(i-1)}$,

$$\mathsf{e}_J^{(i-1)} = \infty. \tag{4.29}$$

Therefore, by Definition 3.11 $\pi_i(\nu, C_J^{(i-1)})$ is defined and hence as by the induction hypothesis for Claim 1 $\Pi^{(i-1)}$ is valid for J, by Definition 3.13 $\Pi^{(i)}$ is valid for J. So, Claim 1 is correct.

We now prove Claims 2(a) and 2(b). In the proof of these two claims and also the proof of the claim 2(c), $(\mathsf{e}_J^{(i)}, \mathsf{es}_J^{(i)}, \Phi_J^{(i)})$ is defined to be $\mathit{result}(J, \Pi^{(i)})$. Since by Definition 3.13 $\mathit{result}(I, \Pi^{(i)}) = \pi_i(\mu, C_I^{(i-1)})$ and $\mathit{result}(J, \Pi^{(i)}) = \pi_i(\nu, C_J^{(i-1)})$, by Definition 3.11

$$\mathit{result}(I, \Pi^{(i)}) = \pi_i(\mu, C_I^{(i-1)}) = (\infty, \infty, \Phi_{sl}(C_I^{(i-1)}, \mu, \infty))$$

and

$$\mathit{result}(J, \Pi^{(i)}) = \pi_i(\nu, C_J^{(i-1)}) = (\infty, \infty, \Phi_{sl}(C_J^{(i-1)}, \nu, \infty)).$$

Thus, since by definition $\mathit{result}(J, \Pi^{(i-1)}) = (\mathsf{e}_J^{(i)}, \mathsf{es}_J^{(i)}, \Phi_J^{(i)})$ and also $\mathit{result}(I, \Pi^{(i-1)}) = (\mathsf{e}_I^{(i)}, \mathsf{es}_I^{(i)}, \Phi_I^{(i)})$,

$$\mathsf{e}_I^{(i)} = \mathsf{es}_I^{(i)} = \mathsf{e}_J^{(i)} = \mathsf{es}_J^{(i)} = \infty, \tag{4.30}$$

$$\Phi_I^{(i)} = \Phi_{sl}(C_I^{(i-1)}, \mu, \infty), \tag{4.31}$$

and

$$\Phi_J^{(i)} = \Phi_{sl}(C_J^{(i-1)}, \nu, \infty). \tag{4.32}$$

Now, if we define $y^{(i)} = z^{(i)} = \infty$, $y^{(i)}$ and $z^{(i)}$ are $\mathit{Visited}(I, \Pi)$-descriptors (Definition 4.2) and by Definition 4.3 $\mathit{mean}_\mu(y^{(i)}) = \mathit{mean}_\nu(z^{(i)}) = \infty = \mathsf{e}_I^{(i)} = \mathsf{es}_I^{(i)} = \mathsf{e}_J^{(i)} = \mathsf{es}_J^{(i)}$. Hence Claims 2(a) and 2(b) are correct for i.

We now prove Claim 2(c) for i. If Q has a speedy leaf, μ-values and ν-values of all objects of $\mathit{sl}(Q)$ are less than ∞ and hence by Lemma 3.1 both $\mathit{find}(I, \mathit{sl}(Q), \infty)$ and $\mathit{find}(J, \mathit{sl}(Q), \infty)$ are equal to END. Thus, as by Equations 4.28 and 4.29 $\mathsf{e}_I^{(i-1)} = \mathsf{e}_J^{(i-1)} = \infty$, if Q has a speedy leaf then $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_I^{(i-1)}) = \mathit{find}(J, \mathit{sl}(Q), \mathsf{e}_J^{(i-1)}) = \text{END}$. Hence, by Lemma 4.7, $\Phi_{sl}(C_I^{(i-1)}, \mu, \mathsf{e}_I^{(i-1)}) = \Phi_{sl}(C_J^{(i-1)}, \nu, \mathsf{e}_J^{(i-1)})$. Therefore because of Equations 4.28 and 4.29, $\Phi_{sl}(C_I^{(i-1)}, \mu, \infty) = \Phi_{sl}(C_J^{(i-1)}, \nu, \infty)$. So, due to Equations 4.31 and 4.32, $\Phi_I^{(i)} = \Phi_J^{(i)}$.

Finally, the third claim is true as π_i is of the third type and so by Definition 3.15 $\mathit{Visited}_i(I, \Pi)$ and $\mathit{Visited}_i(J, \Pi)$ are both equal to the empty set.

Now we prove all claims of the theorem using what we showed by induction. Since

$\Pi = \Pi^{(n)}$ is an eliminating step for I, by Definition 3.13, $\mathsf{e}_I^{(n)} = \mathsf{es}_I^{(n)} = \infty$. Thus, by Claims 2(a) and 2(b) $mean_\mu(y^{(n)}) = mean_\mu(z^{(n)}) = \infty$ and hence by Lemma 4.8 $mean_\nu(y^{(n)}) = mean_\nu(z^{(n)}) = \infty$. So, by Claims 2(a) and 2(b) for n, $\mathsf{e}_J^{(n)} = \mathsf{es}_J^{(n)} = \infty$. Therefore, since we proved $\Pi = \Pi^{(n)}$ is valid for J, Π is an eliminating proof for J (Definition 3.13). The third claim of the theorem also follows immediately by Claim 2(c) for $i = n$. Also, we proved $Visited_i(I, \Pi) = Visited_i(J, \Pi)$, for every i, $1 \leq i \leq n$. Consequently, $Visited(I, \Pi) = \bigcup_{i=1}^{n} Visited_i(I, \Pi) = \bigcup_{i=1}^{n} Visited_i(J, \Pi) = Visited(J, \Pi)$. Hence, all claims of the theorem are correct. □

Corollary 4.9 *Suppose the query tree of an instance I has at most one speedy leaf. Then, for every eliminating proof Π of I, there is a proof of I that visits no object outside $Visited(I, \Pi)$.*

Proof. We claim that the set $\mathcal{P}$ of all comparison propositions between objects in $Visited(I, \Pi)$ satisfied by I is a proof for I. To prove this claim, by Lemma 2.9 it suffices to show that the expansion Σ of the output Φ of Π on I is a solution to every instance that is $Visited(I, \Pi)$-identical with I. So consider an arbitrary instance J that is $Visited(I, \Pi)$-identical with I. According to Theorem 4.5, Π is an eliminating proof for J and Φ is the output of Π on J as it is the output of Π on I. Consequently, by Definition 3.13 the result of Π on J is $C = (\infty, \infty, \Phi)$ since Π is an eliminating proof for J. So, by Definition 3.3 all objects are C-eliminated. Also, by Lemma 3.23 C is valid for J. Thus, according to Lemma 3.9, Σ is a solution to J as Σ is the expansion of the output of C. Therefore, Σ is a solution to any arbitrary instance J that is $Visited(I, \Pi)$ identical with I. Hence, by Lemma 2.9 $\mathcal{P}$ is a proof for I while by the definition of $\mathcal{P}$, $\mathcal{P}$ visits no object outside $Visited(I, \Pi)$. □

4.2 Building an Eliminating Proof

In this part we prove that, given a proof $\mathcal{P}$ for an instance $I = (Q, \omega, \mu)$, one can construct an eliminating proof Π for I such that $Visited(I, \Pi) \subseteq Visited(\mathcal{P})$. Equivalently, we want to prove that the set of objects visited by a proof has enough members to be a superset

of the set of objects visited by some eliminating proof. Now we explain the ideas used to prove such a fact. A proof $\mathcal{P}$ by definition has a certified solution Σ which is a solution for every instance satisfying $\mathcal{P}$. This means that one should be able to convince an observer that Σ is a solution to an instance J satisfying $\mathcal{P}$ using only values of objects visited by $\mathcal{P}$ in J. To prove such a fact to the observer, due to the first condition in Definition 2.10, one must show that, given an object o in Σ, the value of o is in the value set of the root in J and this should be accomplished using only values of objects visited by $\mathcal{P}$. Hence, by Lemma 2.22 using values of objects visited by $\mathcal{P}$ one should be able to prove that there is a sub-intersection tree in which every leaf has an object with the same value as o. When o is not a hidden object (Definition 2.24), as we prove more formally in the next lemma, this idea yields the fact that a sub-intersection tree should exist in which every leaf has a visited object with the same value as o.

Definition 4.4 *Consider a proof $\mathcal{P}$ for an instance $I = (Q, \omega, \mu)$, a value b, and a sub-intersection tree T of Q such that every leaf of T has a $\mathcal{P}$-visited object of value b. Then, T is a* $\mathcal{P}$-witness for b in I.

Lemma 4.10 *For every proof $\mathcal{P}$ of an instance $I = (Q, \omega, \mu)$ and for every value a in the value set of the root that is not the value of any hidden object, there exists a $\mathcal{P}$-witness for a in I.*

Proof. We assume to the contrary that there is no $\mathcal{P}$-witness for a. We create an instance J satisfying $\mathcal{P}$ such that, given a solution Σ certified by $\mathcal{P}$, Σ is not a solution to J and then we show that this contradicts Definition 2.16.

To create the instance J, consider the set $\mathcal{S}$ of all leaves l of Q such that $\omega(l)$ does not have an object of value a visited by $\mathcal{P}$. Since there is no $\mathcal{P}$-witness for a, there is no sub-intersection tree T of Q such that $\mathit{leaves}(T) \cap \mathcal{S} = \emptyset$. So, according to Lemma 2.21, there is a sub-union tree U of Q such that $\mathit{leaves}(U) \subseteq \mathcal{S}$, that is, no leaf of U contains an object of value a visited by $\mathcal{P}$. Suppose $\mathcal{T} = \{o_1, o_2, \ldots, o_m\}$ is the set of all objects of value a in the sequences associated with leaves of U. Due to our selection of U, objects of $\mathcal{T}$ are all skipped by $\mathcal{P}$. Now suppose ϕ is a numbering function defining μ (which exists by Observation 2.24). We now define $\nu = (\mu \times_\phi 2)[o_1, o_2, \ldots, o_m \rightarrowtail (a \times_\phi 2) +_\phi 1]$ and $J = (Q, \omega, \nu)$. Now let us first explain how the ν-values of objects of μ-value a are

evaluated. By Definition 2.20 $a \times_\phi 2 = \phi(2\phi^{-1}(a))$ and so $\phi^{-1}(a \times_\phi 2) = 2\phi^{-1}(a)$. Now, given an object p of I, if p is in $\mathcal{T}$,

$$\begin{array}{lll} \nu(p) & = (a \times_\phi 2) +_\phi 1 & \text{by the definition of } \nu \\ & = \phi(\phi^{-1}(a \times_\phi 2) + 1) & \text{by Definition 2.20} \\ & = \phi(2\phi^{-1}(a) + 1) & \text{by Definition 2.20;} \end{array}$$

otherwise,

$$\begin{array}{lll} \nu(p) & = (\mu \times_\phi 2)(p) & \text{by the definition of } \nu \\ & = \mu(p) \times_\phi 2 & \text{by Definition 2.21} \\ & = \phi(2\phi^{-1}(\mu(p))) & \text{by Definition 2.20.} \end{array}$$

As a result, given an object p of I, if p is in $\mathcal{T}$, $\phi^{-1}(\nu(p)) = 2\phi^{-1}(a) + 1$ and so $\phi^{-1}(\nu(p))$ is odd; otherwise $\phi^{-1}(\nu(p)) = 2\phi^{-1}(\mu(p))$ and thus $\phi^{-1}(\nu(p))$ is even.

Considering a solution Σ certified by $\mathcal{P}$ (which exists by Definition 2.17 it), we now prove that Σ is not a solution to J. Since $\mathcal{P}$ is a proof for I, I satisfies $\mathcal{P}$ and thus by Definition 2.16 Σ is a solution to I. Hence, as by assumption a is in the value set of the root in I, by Definition 2.10 Σ contains an object o of μ-value a. As by assumption a is not the value of any hidden object, the object o is not hidden and hence by Definition 2.24 the object o is visited by $\mathcal{P}$. Therefore, o is not in $\mathcal{T}$ and so $\nu(o) = \mu(o) \times_\phi 2 = a \times_\phi 2$. To prove that Σ is not a solution to J it suffices to show that $\nu(o)$ is not in the value set of the root in J. To prove this fact, by Lemma 2.22 it is sufficient to show that no leaf of U has an object of ν-value $\nu(o) = a \times_\phi 2$. Suppose to the contrary that a leaf l of U has an object q of ν-value $a \times_\phi 2$. Then, $\phi^{-1}(\nu(q)) = \phi^{-1}(a \times_\phi 2) = 2\phi^{-1}(a)$ is even. Consequently, as we proved above, q is not in $\mathcal{T}$ and so as we showed above, $\phi^{-1}(\nu(q)) = 2\phi^{-1}(\mu(q))$. So, as we proved $\phi^{-1}(\nu(q)) = 2\phi^{-1}(a)$, $2\phi^{-1}(\mu(q)) = 2\phi^{-1}(a)$ and thus as by Definition 2.19 ϕ is one-to-one, $\mu(q) = a$. We then obtain a contradiction: by the definition of $\mathcal{T}$, every object in every leaf of U of μ-value a is in $\mathcal{T}$ while we proved $\mu(q) = a$ and q is not in $\mathcal{T}$. So, no leaf of U has an object of ν-value $a \times_\phi 2$ and hence Σ is not a solution to J.

Now one can see a contradiction as follows. As objects $o_1, \ldots, o_m$ are in $\mathcal{T}$ and hence as mentioned they are $\mathcal{P}$-skipped, due to construction of J, by Lemma 2.26 J is an instance satisfying $\mathcal{P}$. Also, by assumption Σ is certified by $\mathcal{P}$. Thus by Definition 2.16 Σ is a solution to J. But, we proved that Σ is not a solution to J. This contradiction yields the

fact that our earlier assumption that there is no $\mathcal{P}$-witness for a was wrong. Therefore, the lemma is correct. □

As the next step towards proving that the set of objects visited by a proof $\mathcal{P}$ has enough members to be the superset of the set of objects visited by an eliminating proof, in the next lemma we show that if there is any speedy leaf, objects of $sl(Q)$ not appearing in all solutions certified by $\mathcal{P}$ are visited by $\mathcal{P}$.

Lemma 4.11 *Consider the instance $I = (Q, \omega, \mu)$ with a speedy leaf and a proof $\mathcal{P}$ of I. If o is an object in $\omega(sl(Q))$ skipped by $\mathcal{P}$, o is in any solution certified by $\mathcal{P}$.*

Proof. We construct an instance $J = (Q, \omega, \nu)$ satisfying $\mathcal{P}$ such that $\nu(o)$ is in the value set of the root but, except o, J does not have any object of ν-value $\nu(o)$ and then we conclude that o appears in any solution certified by $\mathcal{P}$. Suppose ϕ is a numbering function defining μ (Observation 2.24) and consider the value function $\nu = (\mu \times_\phi 2)[o \rightarrowtail (a \times_\phi 2) +_\phi 1]$. Since o is skipped by $\mathcal{P}$, by Lemma 2.26 $J = (Q, \omega, \nu)$ is an instance satisfying $\mathcal{P}$. Consequently, by Definition 2.16, given a solution Σ certified by $\mathcal{P}$, Σ is a solution to J. As o is an object in $\omega(sl(Q))$, $\nu(o)$ is in the value set of the speedy leaf and hence by Observation 2.32 $\nu(o)$ is in the value set of the root. Thus, there is an object of ν-value $\nu(o)$ in Σ. Now, given an object p such that $p \neq o$, by definition of ν, $\nu(p) = (\mu \times_\phi 2)(p) = \mu(p) \times_\phi 2 = \phi(\phi^{-1}(2\mu(p)))$ while $\nu(o) = (a \times_\phi 2) +_\phi 1 = \phi(2\phi^{-1}(a) + 1)$. Hence, $\nu(o)$ is not ϕ-even while the ν-values of all objects other than o are ϕ-even. Therefore, o is the only object of ν-value $\nu(o)$ and so as we proved there is an object of ν-value $\nu(o)$ in Σ, o is in Σ. □

We now study the next idea used to prove that the set of objects visited by a proof $\mathcal{P}$ of an instance I is enough big to be a superset of an eliminating proof for I. If we change values of objects skipped by $\mathcal{P}$ then it can proved that the resulting disordered instance J satisfies $\mathcal{P}$. So, if this modification is such that J remains an instance but has no solution in common with I, then it follows from Definition 2.16 that there is no solution certified by $\mathcal{P}$ and this contradicts Definition 2.17. Consequently, informally speaking, the set of objects visited by a proof can not be too small because otherwise, the set of objects skipped by $\mathcal{P}$ will have enough members so that by changing values of objects skipped by $\mathcal{P}$ one can create an instance J having no solution in common with I. Also, using Observation 2.5

one can prove that J is an instance if and only if J satisfies $\mathcal{K}(I)$. Therefore, if we create a disordered instance J from I by changing values of some of $\mathcal{P}$-skipped objects of I so that J satisfies $\mathcal{K}(I)$ then J has a solution in common with I.

Definition 4.5 *Given a set $\mathcal{P}$ of comparison propositions, an instance $I = (Q, \omega, \mu)$, and a value function ν, ν is a* $(\mu, \mathcal{P})$-similar value function *if ν satisfies $\mathcal{K}(I)$ and for every object o of I visited by $\mathcal{P}$, $\nu(o) = \mu(o)$.*

Given a proof $\mathcal{P}$ for an instance $I = (Q, \omega, \mu)$ and a solution Σ certified by $\mathcal{P}$, an observer who only know values of $\mathcal{P}$-visited objects should be able to confirm that Σ is a solution to I, as explained at the beginning of the section. But, given a $(\mu, \mathcal{P})$-similar value function ν, informally speaking, such an observer can not distinguish between instances I and $J = (Q, \omega, \nu)$ because $\mathcal{P}$-visited objects have the same values in these two instances. So, given a value a and a node v such that a is in the value set of v in J but a in not in the value set of v in I, the observer can not deny or accept that a is in the value set of v in I. But in order to confirm that Σ is a solution to I, the observer should be able to confirm that the value of every object of Σ is in the value set of the root, due to Definition 2.10. We now, given a node v, in the next definition explore the set of values whose membership in the value set of v the observer can not deny.

Definition 4.6 *Given an instance $I = (Q, \omega, \mu)$, a value $a \in \mathbb{V}$, and a node v of Q, if there exists a $(\mu, \mathcal{P})$-similar value function ν such that a is in the value set of v in (Q, ω, ν), we say a* potentially belongs to the value set of v with respect to $(I, \mathcal{P})$. *We denote the set of all values that are potentially in the value set of a node v with respect to $(I, \mathcal{P})$, for some I and $\mathcal{P}$, by Potential$(I, \mathcal{P}, v)$.*

Given a proof $\mathcal{P}$ for an instance $I = (Q, \omega, \mu)$, by Observation 2.5 I satisfies $\mathcal{K}(I)$ and thus by Definition 4.5 μ is $(\mu, \mathcal{P})$-similar. Thus, the next observation follows from Definition 4.6 for $\nu = \mu$.

Observation 4.12 *Given an instance $I = (Q, \omega, \mu)$, a proof $\mathcal{P}$, and a node v of Q, the value set of v is a subset of Potential$(I, \mathcal{P}, v)$.*

Now we first in Lemma 4.13 present an equivalent definition for a value being potentially in the value set of a leaf and then in Lemmas 4.14 and 4.15 explain how the sets of values being potentially in value sets of internal nodes can be evaluated, recursively.

Lemma 4.13 *Given an instance $I = (Q, \omega, \mu)$, a proof $\mathcal{P}$ for I, a leaf l of Q, and a value b that is not in the value set of l, the value b is in Potential$(I, \mathcal{P}, l)$ if and only if there exists a skipped object o of l such that the value function $\mu[o \rightarrowtail b]$ satisfies $\mathcal{K}(I)$.*

Proof. We first prove the "if" part of the lemma and then we provide a proof for the "only if" part. If such an object o exists, then as o is skipped, for every visited object p, $p \neq o$ and thus by definition $\mu[o \rightarrowtail b](p) = \mu(p)$. Hence, as $\mu[o \rightarrowtail b]$ satisfies $\mathcal{K}(I)$, $\mu[o \rightarrowtail b]$ is $(\mu, \mathcal{P})$-similar. Also, as by definition $\mu[o \rightarrowtail b](o) = b$, b is in the value set of l in $(Q, \omega, \mu[o \rightarrowtail b])$. Therefore, by definition b is potentially in the value set of l with respect to $(I, \mathcal{P})$.

Now we suppose that b is potentially in the value set of l with respect to $(I, \mathcal{P})$ and we prove the existence of such an object o. As $b \in$ *Potential*$(I, \mathcal{P}, l)$, by definition there exists a $(\mu, \mathcal{P})$-similar value function ν such that b is in the value set of l in (Q, ω, ν). So, there is an object p of l such that $\nu(p) = b$. As ν satisfies $\mathcal{K}(I)$, it is ordered (Observation 2.5). Suppose a_1 and a_2 are the μ-value of the biggest visited object of l smaller than p ($-\infty$ if it does not exist) and the μ-value of the smallest visited object of l bigger than p (∞ if it does not exist), respectively. Since ν is $(\mu, \mathcal{P})$-similar, the μ-value and the ν-value of an object visited by $\mathcal{P}$ are the same. Hence, a_1 (a_2, respectively) is either $-\infty$ (∞, respectively), or the ν-value and μ-value of an object smaller than (greater than) p. Therefore, since μ and ν are ordered, $a_1 < \mu(p) < a_2$ and $a_1 < \nu(p) < a_2$. So, as $\nu(p) = b$, $a_1 < b < a_2$. Now, suppose $\mathcal{S}$ is the set of all objects with μ-values less than a_2 and greater than a_1 in $\omega(l)$. Then, due to our selection of a_1 and a_2, as p is skipped, all objects of $\mathcal{S}$ are skipped. Also, as we proved $a_1 < \mu(p) < a_2$, p is in $\mathcal{S}$ and hence $\mathcal{S}$ is not empty.

Now we prove that there exists an object q of $\mathcal{S}$ such that the values of all objects of $\omega(l)$ before q are less than b and the values of all objects of $\omega(l)$ after q are greater than b. Then, we show the choice $o = q$ proves the lemma. If $\mathcal{S}$ has any object with a μ-value at most b, we set q to be the biggest object whose μ-value is at most b; otherwise we set q to be the smallest object of $\mathcal{S}$. In either case, one can observe that values of all objects

of $\mathcal{S}$ before (after) q are less than (greater than) b. Also, the μ-value of every object of $\omega(l)$ which is not in $\mathcal{S}$ and is before (after) q is at most a_1 (at least a_2) and we proved $a_1 < b < a_2$. So, q satisfies the required property.

Now we show that the choice $o = q$ proves the lemma. The value function $\mu[o \rightarrowtail b]$ is constructed from the ordered value function μ by just changing the value of $o = q$. Hence, for every two consecutive objects o_1 and o_2 of l other than q, $\nu(o_1) < \nu(o_2)$ as by Definition 2.5 $\mu(o_1) < \mu(o_2)$. Also, for every object o_1 of $\omega(l)$ before q we proved $\mu(o_1) < b$ and so $\nu(o_1) = \mu(o_1) < b = \nu(q)$. Furthermore, for every object o_2 of $\omega(l)$ after q, we showed that $\mu(o_2) > b$ and thus $\nu(o_2) = \mu(o_2) > b = \nu(q)$. As a result, for all choices of two consecutive objects o_1 and o_2 of l, $\nu(o_1) < \nu(o_2)$. Consequently, by Definition 2.5 $\mu[o \rightarrowtail b]$ is ordered and thus it satisfies $\mathcal{K}(I)$. Furthermore, q is skipped by $\mathcal{P}$ since q is in $\mathcal{S}$. So, q is the object we are looking for. □

To explain how the set of values potentially being in the value set of an internal node v is recursively evaluated, we consider two cases depending on whether v is an intersection node or is a union node.

Lemma 4.14 *Consider an instance $I = (Q, \omega, \mu)$, a proof $\mathcal{P}$ for I, and an intersection node v of Q with children $u_1, \ldots, u_k$. Then, Potential$(I, \mathcal{P}, v) = \bigcap_{i=1}^{k}$ Potential$(I, \mathcal{P}, u_i)$.*

Proof. First we prove that Potential$(I, \mathcal{P}, v) \subseteq \bigcap_{i=1}^{k}$ Potential$(I, \mathcal{P}, u_i)$ and then we show that $\bigcap_{i=1}^{k}$ Potential$(I, \mathcal{P}, u_i) \subseteq$ Potential$(I, \mathcal{P}, v)$. Consider a value a in Potential$(I, \mathcal{P}, v)$. By definition, there exists a $(\mu, \mathcal{P})$-similar value function ν satisfying $\mathcal{K}(I)$ such that a is in the value set of v in $J = (Q, \omega, \nu)$. Since a is in the value set of v in J, it is in the value set of u_i in J, for every child u_i of v (Definition 2.9). Hence, due to the existence of the value function ν, a is in Potential$(I, \mathcal{P}, u_i)$, for every child u_i of v. Therefore, a is in $\bigcap_{i=1}^{k}$ Potential$(I, \mathcal{P}, u_i)$.

Now we prove that $\bigcap_{i=1}^{k}$ Potential$(I, \mathcal{P}, u_i) \subseteq$ Potential$(I, \mathcal{P}, v)$. Consider a value a that is in Potential$(I, \mathcal{P}, u_i)$, for every child u_i of v. Then, for every i, $1 \leq i \leq k$, there exists a $(\mu, \mathcal{P})$-similar value function ν_i such that a is in the value set of v in (Q, ω, ν_i). We now construct the new value function ξ by combining the value functions $\nu_1, \ldots, \nu_k$, and μ and then, prove that ξ is a $(\mu, \mathcal{P})$-similar value function and a is in the value set of v in (Q, ω, ξ). For every leaf l of Q and every object o of $\omega(l)$, $\xi(o)$ is defined as follows.

If l is not a descendant of v, we define $\xi(o) = \mu(o)$; otherwise we consider a child u_i of v such that l is a descendant of u_i and we define $\xi(o) = \nu_i(o)$. Now, given an object o visited by $\mathcal{P}$, as ν_i is $(\mu, \mathcal{P})$-similar, $\nu_i(o) = \mu(o)$, for every i, $1 \leq i \leq k$. Also, for every object o visited by $\mathcal{P}$, due to the construction of ξ, either $\xi(o)$ equals $\mu(o)$ or it equals $\nu_i(o)$, for an i, $1 \leq i \leq k$. Thus, in any case, $\xi(o) = \mu(o)$, for every object o visited by $\mathcal{P}$. Now we prove that ξ satisfies $\mathcal{K}(I)$. By Definition 2.14, for every comparison proposition c in $\mathcal{K}(I)$, there is a leaf l of Q such that c is between two objects of $\omega(l)$. Also, given a leaf l of Q, due to the construction of ξ, there exists $\rho \in \{\mu, \nu_1, \ldots, \nu_k\}$ such that for every object o in $\omega(l)$, $\xi(o) = \rho(o)$. Consequently as all members of $\{\mu, \nu_1, \ldots, \nu_k\}$ are $(\mu, \mathcal{P})$-similar and so they all satisfy $\mathcal{K}(I)$, ρ satisfies $\mathcal{K}(I)$. Hence, c is satisfied by ρ and so it is satisfied by ξ. Thus, ξ satisfies $\mathcal{K}(I)$. Therefore, ξ is $(\mu, \mathcal{P})$-similar.

Finally we show that a is in the value set of v in (Q, ω, ξ). For every child u_i of v, as ν_i-values and ξ-values of all objects of $Q[u_i]$ are the same, by Lemma 2.2 value sets of u_i in (Q, ω, ν_i) and in (Q, ω, ξ) are the same. So, since a is in the value set of u_i in (Q, ω, ν_i) , it is in the value set of u_i in (Q, ω, ξ), for every child u_i of v. Therefore, a is in the value set of v in (Q, ω, ξ). Hence, as we proved above that ξ is $(\mu, \mathcal{P})$-similar, a is in $\mathit{Potential}(I, \mathcal{P}, v)$. As a result, $\bigcap_{i=1}^{k} \mathit{Potential}(I, \mathcal{P}, u_i) \subseteq \mathit{Potential}(I, \mathcal{P}, v)$. □

Lemma 4.15 *Consider an instance $I = (Q, \omega, \mu)$, a proof $\mathcal{P}$ for I, and a union node v of Q with children $u_1, \ldots, u_k$. Then, $\mathit{Potential}(I, \mathcal{P}, v) = \bigcup_{i=1}^{k} \mathit{Potential}(I, \mathcal{P}, u_i)$.*

Proof. We prove a value a is in $\mathit{Potential}(I, \mathcal{P}, v)$ if and only if a is in $\mathit{Potential}(I, \mathcal{P}, u_i)$, for a child u_i of v. A value a is in $\mathit{Potential}(I, \mathcal{P}, v)$ if and only if there exists a $(\mu, \mathcal{P})$-similar value function ν such that a is in the value set of v in $J = (Q, \omega, \nu)$, that is, a is in the value set of u_i in J, for a child u_i of v. So, a is in $\mathit{Potential}(I, \mathcal{P}, v)$ if and only if there exists a child u_i of v such that a is in $\mathit{Potential}(I, \mathcal{P}, u_i)$. Thus, $\mathit{Potential}(I, \mathcal{P}, u_i)$ equals $\bigcup_{i=1}^{k} \mathit{Potential}(I, \mathcal{P}, u_i)$. □

Given a proof $\mathcal{P}$ for an instance (Q, ω, μ), due to Lemma 4.13, if a value a is not in $\mathit{Potential}(I, \mathcal{P}, l)$ then for every object o of l such that $\mu[o \rightarrowtail a]$ satisfies $\mathcal{K}(I)$, o is visited by $\mathcal{P}$. We now prove that this result yields the fact that when a value a is not in $\mathit{Potential}(I, \mathcal{P}, l)$, the members of the boundary-set of (I, l, a) are visited by $\mathcal{P}$. To see why

this can be useful, recall from Definition 3.15 that in order to build an eliminating proof Π for an instance I such that $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$, it is needed that for some values b and leaves l, the boundary-set of (I, l, b) is a subset of $\mathit{Visited}(\mathcal{P})$.

Lemma 4.16 *Consider a proof $\mathcal{P}$ for an instance $I = (Q, \omega, \mu)$ and a value b such that $b \notin \mathit{Potential}(I, \mathcal{P}, l)$. Then, the boundary-set of (I, l, b) is a subset of $\mathit{Visited}(\mathcal{P})$.*

Proof. Suppose $\mathcal{S}$ is the set of objects of $\omega(l)$ with values at least (at most) b. By Definition 3.14 it suffices to prove that if $\mathcal{S} \neq \emptyset$ then the smallest (biggest) object o of $\mathcal{S}$ is visited by $\mathcal{P}$. Consider the situation in which this is not true. Since $b \notin \mathit{Potential}(I, \mathcal{P}, l)$, by Observation 4.12 b is not in the value set of l and hence, there is no object of value b in $\omega(l)$. Thus, values of all objects before o are less than b and values of all objects after o are more than b. So, the value function $\nu = \mu[o \rightarrowtail b]$ satisfies $\mathcal{K}(I)$. Therefore, as we supposed o is skipped by $\mathcal{P}$, ν is $(\mu, \mathcal{P})$-similar. Also, as by definition of ν $\nu(o) = b$, b is in the value set of l in (Q, ω, ν). Hence, b is in $\mathit{Potential}(I, \mathcal{P}, l)$, contradicting the assumption of the lemma. □

Suppose we want to construct an eliminating proof Π for an instance I with the same set of visited objects as a proof $\mathcal{P}$, and we need to use an eliminating step π^-, for a sub-union tree T of $N(Q)$, as the ith step of Π. Then, as Definition 3.15 shows, it is necessary that members of the boundary-set of (I, l, e) are visited by $\mathcal{P}$, for every leaf l of T, where e is the e-value of $\Pi^{(i-1)}$ on I. Motivated by this fact, we now show that given a value a that is not in the value set of the root, there is a sub-union tree T of $N(Q)$ such that members of the boundary-set of (I, l, a) are visited by $\mathcal{P}$, for every leaf l of T. To prove this fact, we first in Lemma 4.17 prove that, given a value a that is not in the value set of the root, a is not potentially in the value set of the root in $N(I)$ and then in Lemma 4.18 we show that this result yields the fact that there is a sub-union tree T of $N(Q)$ such that a is not potentially in the value set of any leaf of T. Then we can conclude from Lemma 4.16 that the members of the boundary-set of (I, l, a) are visited by $\mathcal{P}$, for every leaf l of T.

Lemma 4.17 *Suppose $\mathcal{P}$ is a proof for an intersection-normalized instance $I = (Q, \omega, \mu)$. Then, every member of $\mathit{Potential}(N(I), \mathcal{P}, \mathit{root}(N(Q)))$ is in the value set of the root in I.*

Proof. The lemma is almost just a corollary of Lemma 4.10. Given a member a of $\mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$, we create an instance J in which the value of every object o visited by $\mathcal{P}$ is the same as the value of o in I and a is in the value set of the root in $N(J)$. Then we use Lemma 4.10 to find a $\mathcal{P}$-witness for a. Afterward, we show that the existence of this $\mathcal{P}$-witness proves the membership of a in the value set of the root in $N(I)$ and hence in I.

Considering a member a of $\mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$, we now construct the instance J and prove $a \in \mathit{RootValueSet}(J)$. By Definition 4.6, there exists a $(\mu, \mathcal{P})$-similar value function ν such that a is in the value set of the root in $(N(Q), \omega, \nu)$. Now we define the value function ξ for (Q, ω) as follows: For every object o of $N(I)$ we define $\xi(o) = \nu(o)$ and for every object o of I that is not in $N(I)$ we define $\xi(o) = \mu(o)$. Since $\nu(o) = \xi(o)$, for every object o of $N(I)$, by Lemma 2.2 value sets of the root in $(N(Q), \omega, \nu)$ and in $(N(Q), \omega, \xi)$ are the same. So, a is in the value set of the root in $(N(Q), \omega, \xi)$ since a is in the value set of the root in $(N(Q), \omega, \nu)$. Consequently, by Observation 2.32 a is in the value set of the root in $J = (Q, \omega, \xi)$.

Based on a being the ξ-value of a hidden object or not, we now consider two cases. If a is the ξ-value of a hidden object o, o is an object of a speedy leaf, according to Lemma 2.34. So, $\mu(o) = \xi(o) = a$ and hence, due to Observation 2.32, a is in the value set of the root in I since it is the value set of a speedy leaf. Therefore, the lemma is correct in this case.

Now we prove the lemma in the case in which a is not the ξ-value of any hidden object. We first prove that $\mathcal{P}$ is a proof for J and then we conclude that there is a $\mathcal{P}$-witness for a. To prove that $\mathcal{P}$ is a proof for J, by Observation 2.8 it suffices to show that J is an instance $\mathit{Visited}(\mathcal{P})$-identical with I. Also, the correctness of the claim that J is an instance $\mathit{Visited}(\mathcal{P})$-identical is proved if we show that, given an object o in $\mathit{Visited}(\mathcal{P})$, $\mu(o) = \xi(o)$. We now prove this claim. Suppose o is a $\mathcal{P}$-visited object in $\omega(l)$, for a leaf l of Q. Then, due to the construction of ξ, if l is a speedy leaf $\xi(o) = \mu(o)$; otherwise $\xi(o) = \nu(o)$ and since o is visited and ν is $(\mu, \mathcal{P})$-similar, $\nu(o) = \mu(o)$. Hence in either case, $\xi(o) = \mu(o)$. Therefore, the claim that $\mathcal{P}$ is a proof for J is correct.

Finally we prove that there is a $\mathcal{P}$-witness for a in J and we use this $\mathcal{P}$-witness to show the correctness of the lemma. Since we proved that a is in the value set of the root in J and a is not the ξ-value of any hidden object, the existence of an $\mathcal{P}$-witness T for a in J

follows from Lemma 4.10. By Definition 4.4 T is a sub-intersection tree of Q such that for every leaf l of T, $\omega(l)$ contains a $\mathcal{P}$-visited object o_l of ξ-value a. Given a leaf l of T, since o_l is visited by $\mathcal{P}$, by the same argument as above $\mu(o_l) = \xi(o_l)$ and hence $\mu(o_l) = a$. So, T is a sub-intersection tree of Q such that every leaf of T has an object of μ-value a. Therefore, a is in the value set of the root in I, according to Lemma 2.22. □

Lemma 4.18 *Consider an intersection-normalized instance $I = (Q, \omega, \mu)$, a proof $\mathcal{P}$ of I, a value b and a node v of Q such that $b \notin \mathit{Potential}(I, \mathcal{P}, v)$. Then, there exists a sub-union tree T of $Q[v]$ such that $b \notin \bigcup_{l \in \mathit{leaves}(T)} \mathit{Potential}(I, \mathcal{P}, l)$.*

Proof. Defining $\mathcal{S}$ to be the set of all leaves l of Q such that $b \notin \mathit{Potential}(I, \mathcal{P}, l)$, if we show that for every sub-intersection tree U of $Q[v]$ we have $\mathit{leaves}(U) \cap \mathcal{S} \neq \emptyset$, according to Lemma 2.21, we can conclude that there exists a sub-union tree T of $Q[v]$ such that $\mathit{leaves}(T) \subseteq \mathcal{S}$ and thus, $b \notin \bigcup_{l \in \mathit{leaves}(T)} \mathit{Potential}(I, \mathcal{P}, l)$. So in the rest of the proof we prove that, given a sub-intersection tree U of $Q[v]$, there is a leaf l of U such that $l \in \mathcal{S}$, that is, $b \notin \mathit{Potential}(I, \mathcal{P}, l)$.

To prove the above property for U we suppose to the contrary that for every leaf l of U, $b \in \mathit{Potential}(I, \mathcal{P}, l)$. Now by induction on the height of nodes of U we prove that $b \in \mathit{Potential}(I, \mathcal{P}, u)$, for every node u of U. If u is a leaf the proof is trivial. Now suppose u is an internal node and the claim is true for all children of u. First suppose u is union node. Then, defining w as the root of $\top_{\ominus}(U)$, w is a child of u and a node of U. Hence, by induction $b \in \mathit{Potential}(I, \mathcal{P}, w)$ and so $b \in \mathit{Potential}(I, \mathcal{P}, u)$, due to Lemma 4.15. Next suppose u is an intersection node. Then, for every child w of u, by Definition 2.18 w is in U and thus by induction $b \in \mathit{Potential}(I, \mathcal{P}, w)$. Therefore, $b \in \mathit{Potential}(I, \mathcal{P}, u)$, due to Lemma 4.14. Consequently, the claim is correct for every node u of $Q[v]$ and thus it is true for v, that is, $b \in \mathit{Potential}(I, \mathcal{P}, v)$. This contradicts the assumption of the lemma. So the lemma is correct, as argued. □

To facilitate use of the ideas explained in Section 2.4.2, given a proof $\mathcal{P}$ of an instance $I = (Q, \omega, \mu)$, before starting constructing an eliminating proof for I with the same set of visited objects as $\mathcal{P}$, we consider a numbering function ϕ defining μ and we will construct an eliminating function for $J = (Q, \omega, \mu \times_{\phi} 2)$ rather than for I. Then, we will argue

that since I and J are $Objects(I)$-identical instances, they have the same sets of proofs and eliminating proofs and thus our approach is correct. It can be proved that values of objects in J are all ϕ-even. So, in the next lemmas we will suppose that there is a numbering function ϕ such that values of all objects are ϕ-even. This assumption enables us to change the value of a $\mathcal{P}$-skipped object o of a leaf and obtain a new instance satisfying $\mathcal{P}$, as the next lemma shows.

Lemma 4.19 *Consider an intersection-normalized instance $I = (Q, \omega, \mu)$, a leaf l of Q, and a numbering function ϕ such that values of all objects in $\omega(l)$ are ϕ-even. Also suppose $\mathcal{P}$ is a proof for I and o is an object of l skipped by $\mathcal{P}$. If $\nu = \mu[o \rightarrowtail \mu(o) +_\phi 1]$, (Q, ω, ν) is an instance satisfying $\mathcal{P}$.*

Proof. We first prove that $J = (Q, \omega, \nu)$ is an instance and then we show that J satisfies $\mathcal{P}$. By Definition 2.6 J is an instance if ν is ordered. Hence, due to Definition 2.5, we must prove that, given two consecutive objects p and q of a leaf of Q, $\nu(p) < \nu(q)$. As I is an instance, μ is ordered and so $\mu(p) < \mu(q)$. Hence, if neither p nor q equals o, as by definition of ν, $\nu(p) = \mu(p)$ and $\nu(q) = \mu(q)$, the inequality $\nu(p) < \nu(q)$ holds. Now, in either of the two cases $o = p$ and $o = q$ we prove that $\nu(q) < \mu(q)$. Since $\mu(p) < \mu(q)$, by Lemma 2.23 $\phi^{-1}(\mu(p)) < \phi^{-1}(\mu(q))$. Also, as p and q belong to the same leaf and one of them is o and also o is an object of l, p and q are objects of l. Hence, by assumption $\mu(p)$ and $\mu(q)$ are ϕ-even, that is, $\phi^{-1}(\mu(p))$ and $\phi^{-1}(\mu(q))$ are even integers. So as we proved $\phi^{-1}(\mu(p)) < \phi^{-1}(\mu(q))$, $\phi^{-1}(\mu(p)) < \phi^{-1}(\mu(p)) + 1 < \phi^{-1}(\mu(q)) < \phi^{-1}(\mu(q)) + 1$ and thus by Definition 2.19,

$$\begin{array}{lll} \phi(\phi^{-1}(\mu(p))) & = \mu(p) & < \\ \phi(\phi^{-1}(\mu(p)) + 1) & = \mu(p) +_\phi 1 & < \\ \phi(\phi^{-1}(\mu(q))) & = \mu(q) & < \\ \phi(\phi^{-1}(\mu(q)) + 1) & = \mu(q) +_\phi 1. & \end{array}$$

Now, for each of the cases $o = p$ and $o = q$, by the definition of μ, $\nu(p)$ is either $\mu(p)$ or $\mu(p) +_\phi 1$ and $\nu(q)$ is either $\mu(q)$ or $\mu(q) +_\phi 1$. Hence, in any case by the above inequality, $\nu(p) < \nu(q)$. Therefore, ν is ordered and so J is an instance.

Finally, it remains to prove that J satisfies $\mathcal{P}$. By the definition of ν, o is the only object whose μ-value and ν-value are not the same. So, as o is skipped, for every object

p participating in a comparison proposition in $\mathcal{P}$, $\mu(p) = \nu(p)$. Thus, as μ satisfies $\mathcal{P}$, ν satisfies $\mathcal{P}$. □

Given a proof $\mathcal{P}$ for an instance $I = (Q, \omega, \mu)$, we now consider the situation in which we are constructing an eliminating proof Π for I not visiting any objects other than objects visited by $\mathcal{P}$ and we want to use an eliminating step of the form π_T^+ as the ith step of Π, for some T. Then, as Definition 3.15 shows, we must show that if Q has a speedy leaf, the boundary-set of $(I, sl(Q), \mathsf{e})$ is a subset of *Visited*$(\mathcal{P})$ where e is the e-value of the result of $\Pi^{(i-1)}$ on I. We only consider the situation in which $\mathsf{e} \in$ *Potential*$(N(I), \mathcal{P}, \text{root}(N(Q)))$ because otherwise, as we will see, an eliminating step of the first type will be used rather than an eliminating step of the second type. We prove that when $\mathsf{e} \in$ *Potential*$(N(I), \mathcal{P}, \text{root}(N(Q)))$, if Q has a speedy leaf, the boundary-set of $(I, sl(Q), \mathsf{e})$ is a subset of *Visited*$(\mathcal{P})$. If $\mathsf{e} \notin$ *Potential*$(I, \mathcal{P}, sl(Q))$ the claim is true, due to Lemma 4.16. So we just explore the case $\mathsf{e} \in$ *Potential*$(I, \mathcal{P}, sl(Q))$. In this case, we prove that a $\mathcal{P}$-visited object o of value e is in $\omega(sl(Q))$. Having proved this fact, when constructing Π we will easily prove that the boundary-set of $(I, sl(Q), \mathsf{e})$ is $\{o\}$ and so it is a subset of *Visited*$(\mathcal{P})$. We first discuss the special case in which e is in the value sets of the root in $N(I)$ and the speedy leaf and then we consider the general case.

Lemma 4.20 *Consider an intersection-normalized instance $I = (Q, \omega, \mu)$ with one speedy leaf and a numbering function ϕ defining μ such that the μ-value of every object of the speedy leaf is ϕ-even. Also, consider a proof $\mathcal{P}$ for I and suppose a ϕ-even value b is in both the value sets of the root in $N(I)$ and the speedy leaf. Then, there exists an object of value b visited by $\mathcal{P}$ in $\omega(sl(Q))$.*

Proof. The existence of an object o of value b in $\omega(sl(Q))$ follows from the assumption that b is in the value set of the speedy leaf. So we prove that o is visited by $\mathcal{P}$. Assume to the contrary that o is $\mathcal{P}$-skipped. We consider a solution Σ certified by $\mathcal{P}$ (by Definition 2.17 it exists) and construct an instance satisfying $\mathcal{P}$ such that Σ is not a solution to that new instance, contradicting Definition 2.16.

So, let us construct the instance we talked about. Since o is skipped by $\mathcal{P}$, o is in Σ, according to Lemma 4.11. Thus, there is no object of μ-value b other than o in Σ. Now we define $\nu = \mu[o \rightarrowtail b +_\phi 1]$ and $J = (Q, \omega, \nu)$. Then, as $b = \mu(o)$ and by assumption

o is $\mathcal{P}$-skipped, by Lemma 4.19 J is an instance satisfying $\mathcal{P}$ and so by Definition 2.16 Σ is a solution to J. We now prove that b is in the value set of the root in J and then we use this fact to show that there is an object of μ-value b other than o in Σ. The object o is an object of the speedy leaf and hence it is not an object of $N(J)$. Also, by definition for every object p of J other than o the equation $\mu(p) = \nu(p)$ holds. Therefore, for every object p of $N(J)$, $\mu(p) = \nu(p)$. Consequently, by Lemma 2.2 value sets of the root in $N(J)$ and in $N(I)$ are the same and thus b is in the value set of the root in $N(J)$ as by assumption it is in the value set of the root in $N(I)$. As a result, by Observation 2.32 b is in the value set of the root in J. Hence, there is an object q of ν-value b in Σ. Since $\nu(o) = b +_\phi 1$, $q \neq o$. Therefore, by definition of ν, $\mu(q) = \nu(q)$ and so $\mu(q) = b$. Thus, Σ has two distinct objects o and q of the same μ-value b, contradicting the fact that Σ is a solution to I (Definition 2.10). So, the assumption that o is skipped is wrong and hence the lemma is correct. □

Lemma 4.21 *Consider an intersection-normalized instance $I = (Q, \omega, \mu)$ with one speedy leaf and a numbering function ϕ defining μ such that the μ-value of every object of the speedy leaf is ϕ-even. Also, consider a proof $\mathcal{P}$ for I and suppose a ϕ-even value b is in both $Potential(N(I), \mathcal{P}, root(N(Q)))$ and $Potential(I, \mathcal{P}, sl(Q))$. Then, there exists an object of value b visited by $\mathcal{P}$ in $\omega(sl(Q))$.*

Proof. We prove the claim by considering different cases depending on b being in the value set of $sl(Q)$ and in the value set of the root in $N(I)$ or not. In the following discussion, Σ is a solution certified by $\mathcal{P}$ (Definition 2.17). The proof of the case in which b is in both the value set of $sl(Q)$ and the value set of the root in $N(I)$ follows from Lemma 4.20.

Now we consider the case in which b is in the value set of the root in $N(I)$ but it is not in the value set of $sl(Q)$. We prove that this case does not happen. Since b is in the value set of the root in $N(I)$, by Observation 2.32 b is in the value set of the root in I. Hence, as Σ is a solution to I, there is an object o of value b in Σ. The object o does not belong to the speedy leaf as b is not in the value set of the speedy leaf. As $b \in Potential(I, \mathcal{P}, sl(Q))$, according to Lemma 4.13, there exists a skipped object p in $\omega(sl(Q))$ such that $\nu = \mu[p \rightarrowtail b]$ satisfies $\mathcal{K}(I)$. Therefore, as the values of each $\mathcal{P}$-visited object in $J = (Q, \omega, \nu)$ and in I are the same and I satisfies $\mathcal{P}$, J also satisfies $\mathcal{P}$. Hence, by

Definition 2.16 Σ is a solution to J. As p is $\mathcal{P}$-skipped, p is in Σ, according to Lemma 4.11. Also, since p is in $\omega(sl(Q))$, $o \neq p$ and so by definition of ν, $\nu(o) = \mu(o) = b$. Also, by definition of ν, $\nu(p) = b$. Hence, o and p are distinct objects of ν-value b and both appear in the solution Σ of J, contradicting Definition 2.10. So this case does not happen at all.

The last possibility is that b is not in the value set of the root in $N(I)$. In this case, we first show that b is the value of an object o of the speedy leaf and then we prove that b is visited by $\mathcal{P}$. Since $b \in \mathit{Potential}(N(I), \mathcal{P}, \text{root}(N(Q)))$, Lemma 4.17 yields the fact that b is in the value set of the root. Hence, since by assumption b is not in the value set of the root in $N(I)$, it follows from Observation 2.32 that b is in the value of the speedy leaf. So b is the value of an object o of $sl(Q)$.

We next prove that o is visited by $\mathcal{P}$. Suppose this is not true. Then, the object o is in Σ, according to Lemma 4.11. As b is in $\mathit{Potential}(N(I), \mathcal{P}, \text{root}(N(Q)))$, there exists a $(\mu, \mathcal{P})$-similar value function ν for $(N(Q), \omega)$ such that b is in the value set of $(N(Q), \omega, \nu)$. We define the value function ξ as follows: Given an object p of a leaf l of Q, we define $\xi(p) = \nu(p)$ if l is a leaf of $N(Q)$ and we define $\xi(p) = \mu(p)$ if l is a speedy leaf. Since ν is $(\mu, \mathcal{P})$-similar, for every object p visited by $\mathcal{P}$, $\nu(p) = \mu(p)$. Thus, as $\xi(p)$ is either $\mu(p)$ or $\nu(p)$, $\xi(p) = \mu(p)$, for every object p visited by $\mathcal{P}$. Hence, ξ satisfies $\mathcal{P}$, as μ does. Also, ξ satisfies $\mathcal{K}(I)$ because for every two objects o_1 and o_2 belonging to the same leaf, we have either $\xi(o_1) = \mu(o_1)$ and $\xi(o_2) = \mu(o_2)$ or $\xi(o_1) = \nu(o_1)$ and $\xi(o_2) = \nu(o_2)$ and μ and ν are ordered and satisfy $\mathcal{K}(I)$. So, $J = (Q, \omega, \xi)$ is an instance satisfying $\mathcal{P}$ and thus by Definition 2.16 Σ is a solution to J. Also, as for every object p of $N(J)$, p is not in $\omega(sl(Q))$, by definition of ξ, $\xi(p) = \nu(p)$ and hence by Observation 2.2 b is in the value set of the root in $N(J)$ as b is in the value set of the root in $(N(Q), \omega, \nu)$. Furthermore, as o is in $\omega(sl(Q))$, $\xi(o) = \mu(o) = b$ and so b is in the value the speedy leaf in J. Moreover, as every object of the speedy leaf has the same ξ-value and μ-value, ξ-values of all objects of the speedy leaf are ϕ-even. Hence, by Lemma 4.20 an object q of ξ-value b visited by $\mathcal{P}$ is in $\omega(sl(Q))$. Since q is in $sl(Q)$, $\mu(q) = \xi(q) = b$. Furthermore, $q \neq o$ because q is $\mathcal{P}$-visited while by assumption o is not. So, $\omega(sl(Q))$ contains two distinct objects o and q of the same ξ-value b and this contradicts Definition 2.5. Hence, the assumption that o is not visited by $\mathcal{P}$ is wrong and so the lemma is true in this case, as well. □

Our approach is to consider an empty sequence and each time to add one eliminating step to this sequence until we have a complete eliminating proof. So, each time, having a valid sequence Π_1 of eliminating steps, we build a valid sequence Π_2 of eliminating steps of the form $\Pi_2 = \Pi_1 \odot \pi$. Now we explain how the result of Π_2 on a given instance I can be evaluated. If $\Pi_2 = \pi_1, \ldots, \pi_n$, since $\Pi_2 = \Pi_1 \odot \pi$ for $\Pi_1 = \pi_1, \ldots, \pi_{n-1}$ and $\pi_n = \pi$, the next observation follows immediately from Definition 3.13.

Observation 4.22 *Consider a sequence Π_1 of eliminating steps for an instance I and an eliminating step π. If Π_1 is valid for I and $\pi(\mu, C)$ is defined, where $C = \mathit{result}(I, \Pi_1)$, then $\Pi_2 = \Pi_1 \odot \pi$ is valid for I and $\mathit{result}(I, \Pi_2) = \pi(\mu, C)$.*

Now the next lemma shows how the set of objects visited by $\Pi_1 \odot \pi$ is evaluated.

Lemma 4.23 *Consider a valid sequence Π_1 of eliminating steps for an intersection-normalized instance I and an eliminating step π such that $\Pi_2 = \Pi_1 \odot \pi$ is also a valid sequence of eliminating steps for I. The set $\mathit{Visited}(I, \Pi_2)$ can be evaluated as $\mathit{Visited}(I, \Pi_1) \cup \mathit{Visited}_n(I, \Pi_2)$ where n is the length of the sequence Π_2.*

Proof. We use Lemma 3.25 to prove the lemma. By Definition 2.2 the length of the sequence Π_1 is one less than the length of $\Pi_2 = \Pi_1 \odot \pi$, that is, $n - 1$. Suppose $\Pi_1 = \pi_1, \ldots, \pi_{n-1}$ and so $\Pi_2 = \pi_1, \ldots, \pi_{n-1}, \pi$. Then, $\Pi_2^{(i)} = \pi_1, \ldots, \pi_i = \Pi_1^{(i)}$ and thus by Definition 3.25 $\mathit{Visited}_i(I, \Pi_1) = \mathit{Visited}_i(I, \Pi_2)$, for every i, $1 \leq i \leq n - 1$. Hence, by Definition 3.15

$$\begin{aligned}
\mathit{Visited}(I, \Pi_2) &= \textstyle\bigcup_{i=1}^{n} \mathit{Visited}_i(I, \Pi_2) \\
&= \mathit{Visited}_n(I, \Pi_2) \cup \textstyle\bigcup_{i=1}^{n-1} \mathit{Visited}_i(I, \Pi_2) \\
&= \mathit{Visited}_n(I, \Pi_2) \cup \textstyle\bigcup_{i=1}^{n-1} \mathit{Visited}_i(I, \Pi_1) \\
&= \mathit{Visited}_n(I, \Pi_2) \cup \mathit{Visited}(I, \Pi_1).
\end{aligned}$$

□

Given a proof $\mathcal{P}$ of an instance I, the problem is to construct an eliminating proof Π for I visiting no object outside $\mathit{Visited}(\mathcal{P})$. Equivalently, we want to create a valid sequence Π of eliminating steps such that the e-value and the $\mathsf{e_s}$-value of the result of Π on I are ∞ and also $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$. We set our primary goal as constructing Π so that

the e-value of its result is more than values of all $\mathcal{P}$-visited objects of $N(I)$. Suppose a_1, $\ldots$, a_n is the sequence of the values of all $\mathcal{P}$-visited objects of $N(I)$ in order and we have constructed a sequence Π_1 of eliminating steps such that the e-value of its result is greater than a_i but less than a_{i+1}, for some i, and $\mathit{Visited}(I, \Pi_1) \subseteq \mathit{Visited}(\mathcal{P})$. Given a value b, $a_i < b \leq a_{i+1}$, such that $b \notin \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$, in the next lemma we illustrate how we can create a valid sequence Π_2 of eliminating steps such that the e-value of its result is greater than b and $\mathit{Visited}(I, \Pi_2) \subseteq \mathit{Visited}(\mathcal{P})$. Then in Lemma 4.26 we show how we can use this result to create a valid sequence Π_2 of eliminating steps such that the e-value of its result is more than a_{i+1}. In this way, we can inductively reach our primary goal that we described above.

Lemma 4.24 *Consider an intersection-normalized instance $I = (Q, \omega, \mu)$, a proof $\mathcal{P}$ for I, a member b of $\mathbb{V}^+$, and a sequence Π of eliminating steps for I such that the following properties hold.*

1. *The sequence Π is valid.*
2. *$\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$.*
3. *$b < \infty$.*
4. *$b \notin \mathit{Potential}(N(I), \mathcal{P}, \mathit{root}(N(Q)))$.*
5. *For every object o visited by $\mathcal{P}$ of $N(I)$, if $\mu(o) < b$ then $\mu(o) < \mathsf{e}$ where e is the e-value of the result of Π on I.*

Then, there exists a valid sequence Π' of eliminating steps such that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ and the e-value of $\mathit{result}(I, \Pi')$ is greater than b.

Proof. We first define the sequence Π' and then we show that it satisfies the properties required in the lemma. If $\mathsf{e} > b$ then $\Pi' = \Pi$ satisfies the required properties and hence the lemma is trivial in this case. So we suppose $b \geq \mathsf{e}$. Since by assumption $b \notin \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$, by Lemma 4.18 $N(Q)$ has a sub-union tree T such that $b \notin \mathit{Potential}(I, \mathcal{P}, l)$, for any leaf l of T. We now define $\Pi' = \Pi \odot \pi_T^-$. Defining $\mathit{result}(I, \Pi) = C = (\mathsf{e}, \mathsf{e_s}, \Phi)$, since by assumption Π is valid, by Lemma 3.23 C is valid.

Hence, by Definition 3.8 $\pi_T^-(\mu, C)$ is defined. Thus, since Π is valid and $\pi_T^-(\mu, C)$ is defined, by Observation 4.22 Π' is valid.

We first prove that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ and then we show that the e-value of the result of Π' on I is greater than b. By Lemma 4.23 $\mathit{Visited}(I, \Pi') = \mathit{Visited}(I, \Pi) \cup \mathit{Visited}_n(I, \Pi')$ where n is the length of the sequence Π'. Hence, as by assumption, $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$, in order to prove $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ we just need to prove $\mathit{Visited}_n(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$. Since $\pi_n = \pi_T^-$, by Definition 3.15, $\mathit{Visited}_n(I, \Pi')$ equals the set consisting of $\mathit{find}(I, l, \mathsf{e})$ (if it does not equal END) and members of the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$, for all leaves l of T. Hence, we now show that the aforementioned objects are visited by $\mathcal{P}$.

We first prove that, given a leaf l of T, the set of objects of l with values less than (greater than) e is the same as the set of objects with values less than (greater than) b and we conclude that $\mathit{find}(I, l, \mathsf{e})$ (if it does not equal END) and members of the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$ are in the boundary-set of (I, l, b). Also, since $b \notin \mathit{Potential}(N(I), \mathcal{P}, l)$, the boundary-set of (I, l, b) is a subset of $\mathit{Visited}(\mathcal{P})$, due to Lemma 4.16. Hence, by proving the aforementioned claims, we prove that the members of $\mathit{Visited}_n(I, \Pi')$ are in $\mathit{Visited}(\mathcal{P})$ and so $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$. Suppose $\mathcal{S}_l^{\prec}$ and $\mathcal{S}_l^{\succ}$ are the set of objects of $\omega(l)$ of value less than b and the set of objects of $\omega(l)$ of values greater than b, respectively. As l is a leaf of T, by assumption $b \notin \mathit{Potential}(I, \mathcal{P}, l)$ and thus b is not in the value set of l, according to Observation 4.12. Hence, there is no object of value b in $\omega(l)$ and so every object of $\omega(l)$ is either in $\mathcal{S}_l^{\prec}$ or in $\mathcal{S}_l^{\succ}$.

Claim 4.25 *Given a leaf l of T, $\mathcal{S}_l^{\prec}$ is the set of objects of l with values less than e and $\mathcal{S}_l^{\succ}$ is the set of objects of l with values greater than e. Also, there is no object of value e in $\omega(l)$.*

Proof. Since we proved that every object of l is either in $\mathcal{S}_l^{\prec}$ or in $\mathcal{S}_l^{\succ}$, it suffices to show that values of all objects of $\mathcal{S}_l^{\prec}$ are less than e and values of all objects of $\mathcal{S}_l^{\succ}$ are greater than e. The correctness of the claim that the value of each object of $\mathcal{S}_l^{\succ}$ is greater than e follows from the facts that by the definition of $\mathcal{S}_l^{\succ}$, values of all objects of $\mathcal{S}_l^{\succ}$ are all greater than b and by assumption $b \geq \mathsf{e}$. Now we prove that values of all objects of $\mathcal{S}_l^{\prec}$ are less than e.

Suppose to the contrary that the value of an object of $\mathcal{S}_l^{\prec}$ is not less than e. Therefore, $\mathcal{S}_l^{\prec}$ is not empty and the value of the biggest object o of $\mathcal{S}_l^{\prec}$ is at least e. We prove first that o is visited by $\mathcal{P}$ and then we obtain a contradiction. Since the object o is the biggest object of $\mathcal{S}_l^{\prec}$ and $\mathcal{S}_l^{\prec}$ is the set of all objects of value less than b, o is the biggest object of $\omega(l)$ of value less than b. Thus, as we proved $\omega(l)$ has no object of value b, o is the biggest object of $\omega(l)$ of value at most b and so by Definition 3.14 o is in the boundary-set of (I, l, b). Also, as l is a leaf of T, $b \notin \mathit{Potential}(I, \mathcal{P}, l)$ and hence by Lemma 4.16 the boundary-set of (I, l, b) is subset of $\mathit{Visited}(\mathcal{P})$. Therefore, o is in $\mathit{Visited}(\mathcal{P})$. Moreover, by assumption $\mu(o) \geq \mathsf{e}$ and since $o \in \mathcal{S}_l^{\prec}$, $\mu(o) < b$. Consequently, there is a $\mathcal{P}$-visited object with a value at least e and less than b, contradicting the assumption of the lemma. Hence, $\mathcal{S}_l^{\prec}$ has no object of value at least e. □

Before proving that $\mathit{find}(I, l, \mathsf{e})$ (if it does not equal END) and members of the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$ are in the boundary-set of (I, l, b), for a given leaf l of T, we explain which objects of $\mathcal{S}_l^{\prec}$ and $\mathcal{S}_l^{\succ}$ are in the boundary-set of (I, l, b). As we proved, there is no object of value b in $\omega(l)$ and thus it follows from the definition of $\mathcal{S}_l^{\prec}$ (of $\mathcal{S}_l^{\succ}$) that when $\mathcal{S}_l^{\prec}$ ($\mathcal{S}_l^{\succ}$, respectively) is not empty, the biggest object of $\mathcal{S}_l^{\prec}$ (the smallest object of $\mathcal{S}_l^{\succ}$) is the biggest (the smallest) object of value at least (at most) b in l. Hence, by Definition 3.14 the biggest object of $\mathcal{S}_l^{\prec}$ (if $\mathcal{S}_l^{\prec} \neq \emptyset$) and the smallest object of $\mathcal{S}_l^{\succ}$ (if $\mathcal{S}_l^{\succ} \neq \emptyset$) are in the boundary-set of (I, l, b).

Now we show that, given a leaf l of T, if $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$ then $\mathit{find}(I, l, \mathsf{e})$ is the smallest object of $\mathcal{S}_l^{\succ}$ and is in the boundary-set of (I, l, b). If $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$, by Lemma 3.1 there is an object with a value at least e in $\omega(l)$ and $\mathit{find}(I, l, \mathsf{e})$ is the smallest object of l with a value at least e. Hence, as by Claim 4.25 l has no object of value e, when $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$, $\mathit{find}(I, l, \mathsf{e})$ is the smallest object of l with a value greater than e, that is, the smallest object of $\mathcal{S}_l^{\succ}$. So, as we proved above that the smallest object of $\mathcal{S}_l^{\succ}$ is in the boundary-set of (I, l, b), when $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$, $\mathit{find}(I, l, \mathsf{e})$ is in the boundary-set of (I, l, b).

Next we prove that, given a leaf l of T, the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$ is a subset of the boundary-set of (I, l, b). We consider two cases depending on whether l has an object of value $\langle \mathsf{e} \rangle$ or not. If l has an object o of value $\langle \mathsf{e} \rangle$, o is the greatest (the smallest) object of l with a value at most (at least) $\langle \mathsf{e} \rangle$ and thus by Definition 3.14 the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$

equals $\{o\}$. We now prove that o is in the boundary-set of (I, l, b). Since by Claim 4.25 there is no object of value e in l and by assumption there is an object of value $\langle \mathsf{e} \rangle$ in l, $\mathsf{e} \neq \langle \mathsf{e} \rangle$ and so $\mathsf{e} = \langle \mathsf{e} \rangle_+$. Thus, $\mu(o) = \langle \mathsf{e} \rangle < \langle \mathsf{e} \rangle_+ = \mathsf{e}$. Moreover, for every object p of l bigger than o, $\mu(p) > \mu(o) = \langle \mathsf{e} \rangle$ and hence by Observation 2.29 $\mu(p) > \langle \mathsf{e} \rangle_+ = \mathsf{e}$. So, o is the biggest object of l with a value less than e. Therefore, o is the biggest object of $\mathcal{S}_l^{\prec}$ (Claim 4.25) and thus as we proved that the biggest object of $\mathcal{S}_l^{\prec}$ is in the boundary-set of (I, l, b), o is in the boundary-set of (I, l, b).

Now we consider the other case, in which l does not have an object of value $\langle \mathsf{e} \rangle$ and we prove that the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$ is a subset of the boundary-set of (I, l, b). Since $\langle \mathsf{e} \rangle \leq \mathsf{e}$, given an object of l such that $\mu(o) < \langle \mathsf{e} \rangle$, $\mu(o) < \mathsf{e}$ and so by Claim 4.25 $o \in \mathcal{S}_l^{\prec}$. Also, given an object o of $\mathcal{S}_l^{\prec}$, by Claim 4.25 $\mu(o) < \mathsf{e}$ and so by Observation 2.30, $\mu(o) \leq \langle \mathsf{e} \rangle$. Hence, as by assumption l does not have any object of value $\langle \mathsf{e} \rangle$ the value of every object of $\mathcal{S}_l^{\prec}$ is less than $\langle \mathsf{e} \rangle$. Therefore, $\mathcal{S}_l^{\prec}$ is the set of all objects of l with value less than $\langle \mathsf{e} \rangle$. As a result, since l does not have any object of value $\langle \mathsf{e} \rangle$, $\mathcal{S}_l^{\prec}$ is the set of all objects of l with value at most $\langle \mathsf{e} \rangle$. Consequently, as $\mathcal{S}_l^{\prec}$ and $\mathcal{S}_l^{\succ}$ partition the set of objects of l, $\mathcal{S}_l^{\succ}$ is the set of all objects of l with values greater than $\langle \mathsf{e} \rangle$, or equivalently, the set of all objects of l with values at least $\langle \mathsf{e} \rangle$. Therefore, it follows from Definition 3.14 that the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$ is the set of the biggest object of $\mathcal{S}_l^{\prec}$ (if $\mathcal{S}_l^{\prec} \neq \emptyset$) and the smallest object of $\mathcal{S}_l^{\succ}$ (if $\mathcal{S}_l^{\succ} \neq \emptyset$) and we proved any of these two objects that exists is in the boundary-set of (I, l, b). Hence, the boundary-set of $(I, l, \langle \mathsf{e} \rangle)$ is a subset of the boundary-set of (I, l, b). Also, we proved above that $\mathit{find}(I, l, \mathsf{e})$ is END or is in the boundary-set of (I, l, b). So, the claim that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ is true, as explained before.

Now, defining e' as the result of Π' on I, it remains to show that $\mathsf{e}' > b$. Since by Observation 4.22 $\mathit{result}(I, \Pi') = \pi_T^-(\mu, C)$, e' is the e-value of $\pi_T^-(\mu, C)$. Hence, by Definition 3.8, $\mathsf{e}' = \mathit{hmin}(\mu, C, T)$ and so all we need to prove is that $\mathit{hmin}(\mu, C, T) > b$. If no leaf of T has a C-remaining object, by Definition 3.7 $\mathit{hmin}(\mu, C, T) = \infty$. So, as by assumption $b < \infty$ the claim is true in this case. Now consider the case in which there is a C-remaining object and hence by Definition 3.7 $\mathit{hmin}(\mu, C, T)$ is the minimum of $\mu(\mathit{find}(I, l, \mathsf{e}))$ for all leaves l of T with $\mathit{find}(I, l, \mathsf{e}) \neq$ END. We proved above that when $\mathit{find}(I, l, \mathsf{e}) \neq$ END then $\mathit{find}(I, l, \mathsf{e})$ is the smallest object of $\mathcal{S}_l^{\succ}$, for every leaf l of T. Also, by definition values of all objects of $\mathcal{S}_l^{\succ}$ are greater than b, for every leaf l of T. Hence,

when $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$, $\mu(\mathit{find}(I, l, \mathsf{e})) > b$, for all leaves l of T. Thus, in this case $\mathit{hmin}(\mu, C, T) > b$ as $\mathit{hmin}(\mu, C, T)$ is the minimum of $\mu(\mathit{find}(I, l, \mathsf{e}))$ for all leaves l of T with $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$. So, $\mathsf{e}' > b$. □

Using the following lemma we inductively construct a valid sequence of eliminating steps such that its result is an eliminating configuration with an e-value more than the value of every object visited by $\mathcal{P}$ and also, $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$.

Lemma 4.26 *Suppose the following conditions hold.*

1. $I = (Q, \omega, \mu)$ *is an intersection-normalized instance with at most one speedy leaf.*
2. ϕ *is a numbering function defining* μ *such that the value of each object of* I *is* ϕ*-even and* ϕ*-positive.*
3. $\mathcal{P}$ *is a proof for* I *and* $a_1, \ldots, a_m$ *is the sequence of values of all objects of* $N(I)$ *visited by* $\mathcal{P}$ *in increasing order. We also define* $a_0 = -\infty$.
4. Π *is a valid sequence of eliminating steps of* I *such that* $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$. *Also we suppose* $\mathit{result}(I, \Pi) = (\mathsf{e}, \mathsf{e_s}, \Phi)$.
5. i *is an integer such that* $0 \leq i \leq m - 1$ *and if* $i \neq 0$ *then* $\mathsf{e} > a_i$.

Then, there exists a valid sequence Π' *of eliminating steps of* I *such that* $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ *and the* e*-value of* $\mathit{result}(I, \Pi')$ *is greater than* a_{i+1}.

Proof. We first in the case in which $a_{i+1} \notin \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$ prove the lemma by simply applying Lemma 4.24. Afterward, we discuss the lemma in the case $a_{i+1} \in \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$. In this case, we first construct a valid sequence Π_1 of eliminating steps visiting no $\mathcal{P}$-skipped object such that, defining e_1 as the e-value of Π_1 on I, $\mathsf{e}_1 \geq a_{i+1}$. Then as we will argue, if $\mathsf{e}_1 > a_{i+1}$, the choice $\Pi' = \Pi_1$ simply proves the lemma. So we suppose $\mathsf{e}_1 = a_{i+1}$ and based on a_{i+1} being the value of an object of a speedy leaf or not we will have two different arguments.

Before starting considering the two cases based on a_{i+1} being in $\mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$ or not, we mention that as by assumption Π is valid and $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$, when $a_{i+1} < \mathsf{e}$ the choice $\Pi' = \Pi$ proves the lemma and hence in the proof we suppose $a_{i+1} \geq \mathsf{e}$.

To prove the lemma in the case $a_{i+1} \notin \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$ we show that we can apply Lemma 4.24 for Π and $b = a_{i+1}$ to construct the sequence Π' required by the lemma. For this purpose, we prove the five conditions required by Lemma 4.24, one after another. The first two conditions follow from the assumption of the lemma. Also, since a_{i+1} is the value of an object, $b = a_{i+1} < \infty$ and so the third condition is also satisfied. The fourth condition follows from the fact that we are considering the case $b = a_{i+1} \notin \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$. Now we prove the fifth condition. Due to the definition of the sequence $a_1, \ldots, a_m$, given a $\mathcal{P}$-visited object o with a value less than $b = a_{i+1}$, $\mu(o) \in \{a_1, \ldots, a_i\}$ and so $\mu(o) \leq a_i$. Also, given such an object o, $i \geq 1$ (since $\mu(o) \in \{a_1, \ldots, a_i\}$) and thus it follows from the assumption of the lemma that $a_i < \mathsf{e}$. Consequently, given a $\mathcal{P}$-visited object o with a value less than $b = a_{i+1}$, $\mu(o) < \mathsf{e}$. Therefore, all the five conditions hold and so we can apply Lemma 4.24 for Π and $b = a_{i+1}$ and conclude the correctness of the lemma.

In the rest of the proof we consider the case $a_{i+1} \in \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$. Since by assumption a_{i+1} is the value of an object and hence it is ϕ-positive, by Definition 2.22 $\phi^{-1}(a_{i+1}) \geq 1$ and thus by Definition 2.19 $\phi(\phi^{-1}(a_{i+1}) - 1)$ is defined. Now, defining $b = \phi(\phi^{-1}(a_{i+1}) - 1)$, we first show that we can apply Lemma 4.24 for Π and b to have a new valid sequence Π_1 of eliminating steps such that the e-value of the result of Π_1 is greater than b. Then we prove that the e-value of the result of Π_1 is at least a_{i+1}, as described at the beginning of the proof. So, let us prove the correctness of the five conditions of Lemma 4.24, one by one. As before, the first two conditions follow from the assumption of the lemma. Now we discuss the third condition. Since $b = \phi(\phi^{-1}(a_{i+1}) - 1)$,

$$\phi^{-1}(b) = \phi^{-1}(a_{i+1}) - 1 < \phi^{-1}(a_{i+1}). \tag{4.33}$$

Therefore, by Definition 2.19, $\phi(\phi^{-1}(b)) < \phi(\phi^{-1}(a_{i+1}))$ and thus $b < a_{i+1} \leq \infty$. Hence, the third condition is true. Also, since by assumption a_{i+1} is the value of a visited object, by assumption a_{i+1} is ϕ-even and hence $\phi^{-1}(a_{i+1})$ is even. Therefore, due to Equation 4.33, $\phi^{-1}(b)$ is not even and so b is not ϕ-even. Consequently, I does not have an object of value b since values of all objects of I are ϕ-even. As a result, b is not in the value set of the root of I and thus b in not in $\mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$, according to Lemma 4.17. So, the fourth condition is also satisfied. Now we consider the fifth condition. As we proved in the

previous case, for every object o visited by $\mathcal{P}$ with a value less than a_{i+1}, $\mu(o) \in \{a_1, \ldots, a_i\}$ and $i > 0$ and thus $\mu(o) \leq a_i < \mathsf{e}$. Hence, as by assumption $b < a_{i+1}$, for every $\mathcal{P}$-visited object o with a value less than b, $\mu(o) < \mathsf{e}$. Therefore all the five conditions are true. So we can apply Lemma 4.24 and conclude that there is a valid sequence of eliminating steps Π_1 such that $\mathit{Visited}(I, \Pi_1) \subseteq \mathit{Visited}(\mathcal{P})$ and, defining $\mathit{result}(I, \Pi_1) = C_1 = (\mathsf{e}_1, \mathsf{e}_{\mathsf{s}1}, \Phi_1)$,

$$\mathsf{e}_1 > b. \tag{4.34}$$

Now we prove that $\mathsf{e}_1 \geq a_{i+1}$. To prove this fact, we first show that if $\mathsf{e}_1 \notin \{-\infty, \infty\}$ then $\langle \mathsf{e}_1 \rangle$ is ϕ-even. Then we will argue that since $\mathsf{e}_1 > b = \phi(\phi^{-1}(a_{i+1}) - 1)$, $\mathsf{e}_1 \geq a_{i+1}$. By Lemma 3.26 e_1 either is in $\{-\infty, \infty\}$ or $\langle \mathsf{e}_1 \rangle$ is the value of an object of I. Also, since by Equation 4.34 $\mathsf{e}_1 > b$, $\mathsf{e}_1 \neq -\infty$. In addition, if $\mathsf{e}_1 = \infty$, then the correctness of the claim $\mathsf{e}_1 \geq a_{i+1}$ is trivial. Now, consider the case in which $\mathsf{e}_1 \notin \{-\infty, \infty\}$ and so $\langle \mathsf{e}_1 \rangle$ is the value of an object. Then, as by assumption the values of all objects are ϕ-even, $\langle \mathsf{e}_1 \rangle$ is ϕ-even. So, as we proved before that b is not ϕ-even, $b \neq \langle \mathsf{e}_1 \rangle$. Thus, either $b > \langle \mathsf{e}_1 \rangle$ or $b < \langle \mathsf{e}_1 \rangle$. In the first case, $b \not\leq \langle \mathsf{e}_1 \rangle$ and thus by Observation 2.30 $b \not< \mathsf{e}_1$, contradicting Equation 4.34. So, $b < \langle \mathsf{e}_1 \rangle$. Also, as $\langle \mathsf{e}_1 \rangle$ is ϕ-even, by Definition 2.22, it is defined by ϕ. Hence, by Lemma 2.23 $\phi^{-1}(b) < \phi^{-1}(\langle \mathsf{e}_1 \rangle)$ and thus because of Equation 4.33 $\phi^{-1}(a_{i+1}) - 1 < \phi^{-1}(\langle \mathsf{e}_1 \rangle)$. Therefore, as the result of function ϕ^{-1} is integer, $\phi^{-1}(a_{i+1}) \leq \phi^{-1}(\langle \mathsf{e}_1 \rangle)$ and so by Definition 2.19, $a_{i+1} = \phi(\phi^{-1}(a_{i+1})) \leq \phi(\phi^{-1}(\langle \mathsf{e}_1 \rangle)) = \langle \mathsf{e}_1 \rangle \leq \mathsf{e}_1$. So the claim $a_{i+1} \leq \mathsf{e}_1$ is proved.

Next we show that when $\mathsf{e}_1 > a_{i+1}$ the lemma is trivial. If $\mathsf{e}_1 > a_{i+1}$, as Π_1 is valid and $\mathit{Visited}(I, \Pi_1) \subseteq \mathit{Visited}(\mathcal{P})$, the choice $\Pi' = \Pi_1$ proves the lemma. So we suppose this is not the case and hence $\mathsf{e}_1 = a_{i+1}$.

Here we consider two possibilities depending on whether there is any object of value a_{i+1} in any speedy leaf or not. First suppose that $\mathit{sl}(Q)$ exists and contains an object of value a_{i+1}. Then, a_{i+1} is in the value set of $\mathit{sl}(Q)$ and so by Observation 4.12 a_{i+1} is in $\mathit{Potential}(I, \mathcal{P}, \mathit{sl}(Q))$. Also, by assumption a_{i+1} is in $\mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$. Hence, the speedy leaf has an object p visited by $\mathcal{P}$ of value a_{i+1}, according to Lemma 4.21. We now prove that, defining T as the sub-intersection of Q with only one leaf $\mathit{sl}(Q)$, the choice $\Pi' = \Pi_1 \odot \pi_T^+$ proves the lemma. As by definition C_1 is the result of Π_1 and Π_1 is valid, by Lemma 3.23 C_1 is valid. Also, $\mathit{sl}(Q)$ is the only leaf of T and $\mathit{sl}(Q)$ has an object

of value e_1. Therefore, e_1 is a main value and by Definition 3.10 $\pi_T^+(\mu, C_1)$ is defined. Also, Π_1 is valid. Hence, by Observation 4.22 Π' is a valid sequence of eliminating steps and $\mathit{result}(I, \Pi') = \pi_T^+(\mu, C_1)$. Suppose $\pi_T^+(\mu, C_1) = (\mathsf{e}', \mathsf{e_s}', \Phi')$.

We now first prove that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ and then we show that $\mathsf{e}' > a_{i+1}$. Defining n as the length of Π', by Lemma 4.23 $\mathit{Visited}(I, \Pi') = \mathit{Visited}(I, \Pi_1) \cup \mathit{Visited}_n(I, \Pi')$. Also, since $\pi_n = \pi_T^+$ and $\mathit{sl}(Q)$ is the only leaf of T, by Definition 3.15 $\mathit{Visited}_n(I, \Pi') = \mathcal{T} \cup \mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_1)$ where $\mathcal{T}$ is the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_1)$. As a result, $\mathit{Visited}(I, \Pi') = \mathit{Visited}(I, \Pi_1) \cup \mathcal{T} \cup \mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_1)$. Thus, as $\mathit{Visited}(I, \Pi_1) \subseteq \mathit{Visited}(\mathcal{P})$, in order to prove that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ it suffices to show that $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_1)$ and the members of the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_1)$ are $\mathcal{P}$-visited objects. As the speedy leaf has an object of value e_1, by Lemma 3.2 $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_1)$ is an object of value e_1 of the speedy leaf and hence as p (defined in the previous paragraph) is an object of the speedy leaf of value e_1, $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}_1) = p$. Also, since $\mu(p) = \mathsf{e}_1$, p the biggest (the smallest) object of value at most (at least) e_1 and hence the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_1)$ has only one member p. Moreover, as stated in the previous paragraph, p is visited by $\mathcal{P}$. So, as explained, we may conclude that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$. The correctness of the claim $\mathsf{e}' > a_{i+1}$ follows form the fact that by Definition 3.10 the e-value of $\pi_T^+(\mu, C_1)$ equals e_1+ and hence since by assumption $\mathsf{e}_1 = a_{i+1}$, $\mathsf{e}' = \mathsf{e}_1+ > \mathsf{e}_1 = a_{i+1}$. So Π' satisfies all requirements.

Now, we consider the other possibility, in which there is no object of value a_{i+1} in any speedy leaf. First we prove that there is a $\mathcal{P}$-witness T for e_1 and then we show that the choice $\Pi' = \Pi_1 \odot \pi_T^+$ proves the lemma. Since $\mathit{sl}(Q)$ has no object of value a_{i+1} , a_{i+1} is not in the value set of any speedy leaf. Also, as by assumption $a_{i+1} \in \mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$, Lemma 4.17, a_{i+1} is in the value set of the root in I. So, it follows from Observation 2.32 that a_{i+1} is in the value set of the root in $N(I)$. Since $a_{i+1} = \mathsf{e}_1$ is not the value of any object of any speedy leaf, by Lemma 2.34 it is not the value of any hidden object. Hence, as we proved e_1 is in the value set of the root in I, by Lemma 4.10 there exists a $\mathcal{P}$-witness for e_1, that is, there is a sub-intersection tree T such that every leaf of T contains a $\mathcal{P}$-visited object of value $a_{i+1} = \mathsf{e}_1$ (Definition 4.4).

Now we prove that $\Pi' = \Pi_1 \odot \pi_T^+$ satisfies the required properties in the lemma. We first prove that Π' is valid for I, then we prove that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$, and finally

we show that the e-value of the result of Π' on I is greater than a_{i+1}. As argued before, C_1 is valid and $\mathsf{e}_1 = a_{i+1}$ is a main value. Also, as mentioned, for every leaf l of T, $\omega(l)$ has an object of value e_1. Consequently, by Definition 3.10 $\pi^+_T(\mu, C_1)$ is defined. So, as Π_1 is valid, by Observation 4.22 Π' is also valid and its result on I is $\pi^+_T(\mu, C_1)$. Suppose $\pi^+_T(\mu, C_1) = (\mathsf{e}', \mathsf{e_s}', \Phi')$. By Definition 3.10 $\mathsf{e}' = \mathsf{e}_1+ > \mathsf{e}_1 = a_{i+1}$ and hence it suffices to show that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$.

Now we complete the proof of the lemma in this case by showing that $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$. By Lemma 4.23 $\mathit{Visited}(I, \Pi') = \mathit{Visited}(I, \Pi_1) \cup \mathit{Visited}_n(I, \Pi')$ where n is the length of Π'. Also, since $\pi_n = \pi^+_T$, by Definition 3.15 $\mathit{Visited}_n(I, \Pi') = \mathcal{S} \cup \bigcup_{l \in leaves(T)} \{\mathit{find}(I, l, \mathsf{e}_1)\}$ where $\mathcal{S} = \emptyset$ if Q has no speedy leaf and $\mathcal{S}$ equals the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_1)$, otherwise. As mentioned before, $\mathit{Visited}(I, \Pi_1)$ is a subset of $\mathit{Visited}(\mathcal{P})$. Also, as T is a $\mathcal{P}$-witness for e_1, by Definition 4.4 every leaf l of T has a $\mathcal{P}$-visited object of value e_1 and thus by Lemma 3.2 $\mathit{find}(I, l, \mathsf{e}_1)$ is the object of l of value e_1, for every leaf l of T. Hence, for every leaf l of T, $\mathit{find}(I, l, \mathsf{e}_1)$ is visited by $\mathcal{P}$. Thus, it remains to prove that $\mathcal{S} \subseteq \mathit{Visited}(\mathcal{P})$. If the query tree has no speedy leaf, this claim is true as $\mathcal{S} = \emptyset$. So, suppose Q has a speedy leaf and thus $\mathcal{S}$ is the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_1)$. Recall that a_{i+1} is the value of a visited object and thus by assumption it is a ϕ-even value. Also, by assumption a_{i+1} is in $\mathit{Potential}(N(I), \mathcal{P}, \mathrm{root}(N(Q)))$ and there is no object of value a_{i+1} in any speedy leaf. As a result, if a speedy leaf exists, $a_{i+1} = \mathsf{e}_1$ is not in $\mathit{Potential}(I, \mathcal{P}, \mathit{sl}(Q))$, according to Lemma 4.21 for $b = a_{i+1}$. So, by Lemma 4.16 the boundary-set of $(I, \mathit{sl}(Q), \mathsf{e}_1)$ is a subset of $\mathit{Visited}(\mathcal{P})$. Hence, in any case $\mathcal{S} \subseteq \mathit{Visited}(\mathcal{P})$. Therefore, $\mathit{Visited}(I, \Pi') \subseteq \mathit{Visited}(\mathcal{P})$ and thus Π' satisfies all properties required by the lemma. Consequently, the lemma is correct in this case, as well. □

Suppose using Lemma 4.26 we have constructed a valid sequence Π of eliminating steps of an instance I such that the e-value of the result of Π on I is greater than the value of every object of $N(I)$ visited by a proof $\mathcal{P}$ and $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$. We use the next lemma to construct an eliminating step π such that $\Pi \odot \pi$ is a valid sequence of eliminating steps visiting no object outside $\mathit{Visited}(\mathcal{P})$ such that the e-value the result of $\Pi \odot \pi$ is ∞.

Lemma 4.27 *Consider an intersection-normalized instance $I = (Q, \omega, \mu)$ and a proof $\mathcal{P}$ for I. Every sub-intersection tree T of Q with at least two leaves has a leaf l such that the*

biggest object of $\omega(l)$ is visited by $\mathcal{P}$.

Proof. Assume to the contrary that there exists a sub-intersection tree T violating the lemma. Thus, given a leaf l of T, the last object o_l of l is skipped by $\mathcal{P}$. By changing the value of the last object of each leaf of T we build two instances satisfying $\mathcal{P}$ but having no solution in common and then we show that this contradicts the fact that every proof has a certified solution.

Let us first construct the instances. Consider an integer i greater than $\phi^{-1}(\mu(p))$, for any object p of I, and suppose $\mathit{leaves}(T) = \{l_1, \ldots, l_k\}$. We define $\nu = \mu[o_{l_1}, o_{l_2}, \cdots, o_{l_k} \rightarrowtail \phi(i)]$ and $\xi = \nu[o_{l_k} \rightarrowtail \phi(i+1)]$. We first prove that $J = (Q, \omega, \nu)$ and $K = (Q, \omega, \xi)$ satisfy $\mathcal{P}$ and are instances and then we prove that these two instances do not have any solution in common. Since by assumption all objects $o_{l_1}, \ldots, o_{l_k}$ are skipped by $\mathcal{P}$, for every object o visited by $\mathcal{P}$, $\mu(o) = \nu(o) = \xi(o)$. Hence, as $\mathcal{P}$ is satisfied by μ, $\mathcal{P}$ is also satisfied by ν and by ξ, that is, by J and by K. In the rest of the proof we will use the fact that as $i + 1 > i > \phi^{-1}(\mu(p))$, by Definition 2.19 $\phi(i+1) > \phi(i) > \phi(\phi^{-1}(\mu(p))) = \mu(p)$, for every object p of I.

We now prove that ν and ξ are ordered. For this purpose, by Definition 2.5 we must prove that, given two consecutive objects p and q of a leaf l of Q, the inequalities $\nu(p) < \nu(q)$ and $\xi(p) < \xi(q)$ hold. The object p is not the last object of l and hence $\xi(p) = \nu(p) = \mu(p)$. If $l \notin \mathit{leaves}(T)$ or q is not the last object of l, $\xi(q) = \nu(q) = \mu(q)$ and hence as $\mu(p) < \mu(q)$ (because μ is ordered), the inequalities $\nu(p) < \nu(q)$ and $\xi(p) < \xi(q)$ hold. So, suppose $l \in \mathit{leaves}(T)$ and $q = o_l$. As we proved, $\phi(i+1) > \phi(i) > \mu(p)$. Hence, as $\nu(q)$ and $\xi(q)$ each one equals $\phi(i)$ or $\phi(i+1)$, and we showed $\mu(p) = \nu(p) = \xi(p)$, the inequalities $\nu(p) < \nu(q)$ and $\xi(p) < \xi(q)$ hold. Thus, ν and ξ are ordered and hence J and K are instances.

Having proved that J and K are instances satisfying $\mathcal{P}$, we now prove that a solution Σ of J is not a solution of K. Assume to the contrary that Σ is a solution to both J and K. Since by the definition of ν, $\phi(i)$ the ν-value of the object o_l of l, for every leaf l of the sub-intersection tree T of Q, $\phi(i)$ is in the value set of the root in J, due to Lemma 2.22. So, by Definition 2.10 an object o of ν-value $\phi(i)$ is in Σ. But, as we proved, $\phi(i)$ is greater than the μ-value of any object of I. Therefore, as $\nu(o) = \phi(i)$, $\mu(o) \neq \nu(o)$ and hence by the definition of ν, o is in the set $\mathcal{S} = \{o_{l_1}, o_{l_2}, \ldots, o_{l_k}\}$. Consequently, the ξ-value of o is

either $\phi(i)$ or $\phi(i+1)$. Hence, as o is in Σ and by assumption Σ is a solution to K, by Definition 2.10 a value a in $\{\phi(i), \phi(i+1)\}$ is in the value set of the root in K.

Now we complete the proof of the claim that Σ is not a solution to both J and K by showing a contradiction resulting from the membership of a in the value set of the root in K. Since $\phi(i)$ and $\phi(i+1)$ are greater than the μ-value of any object of I, for every object of ξ-value $\phi(i)$ or $\phi(i+1)$, $\xi(p) \neq \mu(p)$ and thus by the definition of ξ and ν, p is among the objects $o_{l_1}, \ldots, o_{l_k}$, that is, p is in $\mathcal{S}$. Also there is exactly one object of ξ-value $\phi(i+1)$ in $\mathcal{S}$ and by the definition of $\mathcal{S}$, $|\mathcal{S}| = |\mathit{leaves}(T)| > 1$. So, there is at least one object of ξ-value $\phi(i)$ and exactly one object of ξ-value $\phi(i+1)$ in $\mathcal{S}$. Hence, as $a \in \{\phi(i), \phi(i+1)\}$ and we showed all objects of value $\phi(i)$ or $\phi(i+1)$ are in $\mathcal{S}$, the set of objects of value a is a proper subset of $\mathcal{S}$. Moreover, $\mathcal{S}$ is the set of the last objects of leaves of T and so in this way there is a one to one correspondence between leaves of T and objects of $\mathcal{S}$. So, the set $\mathcal{T}$ of leaves having an object of ξ-value a is a proper subset of $\mathit{leaves}(T)$. Since a is in the value set of the root in K, there is a sub-intersection tree U such that every leaf of U has an object of ξ-value a, according to Lemma 2.22. So, $\mathit{leaves}(U) \subseteq \mathcal{T} \subsetneq \mathit{leaves}(T)$ and thus $\mathit{leaves}(U) \subseteq \mathit{leaves}(T)$ but $\mathit{leaves}(U) \neq \mathit{leaves}(T)$. This contradicts Lemma 2.18. Therefore Σ is not a solution to both J and K where Σ was selected as an arbitrary solution to both J and K. As a result J and K do not have any solution in common. But as J and K are instances satisfying $\mathcal{P}$, by Definition 2.16 a solution certified by $\mathcal{P}$ is a solution to both J and K. Hence, $\mathcal{P}$ has no certified solution and so by Definition 2.17 $\mathcal{P}$ is not a proof for (Q, ω), contradicting the assumption of the lemma. Therefore, the assumption that T violates the lemma is false. Consequently, the lemma is correct. □

Now we can prove that, given a proof $\mathcal{P}$ for an instance I, there is an eliminating proof for I visiting no object other than those visited by $\mathcal{P}$. Recall the definition of a query tree being normalized from Definition 2.3.

Theorem 4.28 *Consider a normalized instance $I = (Q, \omega, \mu)$ such that Q has more than one node and has at most one speedy leaf. Given a proof $\mathcal{P}$ for I, there exists an eliminating proof Π for I such that $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$.*

Proof. Considering a numbering function ϕ defining I, we first create an instance J such that J is $\mathit{Objects}(I)$-identical with I and the value of every object of J is ϕ-even

and ϕ-positive. Then we use Lemma 4.26 to inductively create a valid sequence Π of eliminating steps visiting no $\mathcal{P}$-skipped object such that the e-value of the result of Π on J is greater than the value of every $\mathcal{P}$-visited object in $N(J)$. Then, we apply Lemma 4.27 for sub-intersection trees of $N(Q)$ to constructing a step π such that the e-value of $\Pi \odot \pi$ is ∞. After that, we complete the eliminating proof by adding the step π^∞ to $\Pi \odot \pi$. Finally, we use Theorem 4.5 to prove that $(\Pi \odot \pi) \odot \pi^\infty$ is an eliminating proof for I and $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$.

Defining $\nu = (\mu +_\phi 1) \times_\phi 2$ and $J = (Q, \omega, \nu)$, we now prove that the values of all objects of J are ϕ-even and ϕ-positive, J is an instance, and $\mathcal{P}$ is a proof for J. By Definitions 2.21 and 2.20, given an object o, $(\mu +_\phi 1)(o) = \mu(o) +_\phi 1 = \phi(\phi^{-1}(\mu(o)) + 1)$ and thus $\phi^{-1}((\mu +_\phi 1)(o)) = \phi^{-1}(\mu(o)) + 1$. Consequently, since $\nu = (\mu +_\phi 1) \times_\phi 2$, by Definitions 2.21 and 2.20, $\nu(o) = (\mu +_\phi 1)(o) \times_\phi 2 = \phi(2\phi^{-1}((\mu +_\phi 1)(o))) = \phi(2(\phi^{-1}(\mu(o)) + 1)) = \phi(2\phi^{-1}(\mu(o)) + 2)$ and so $\phi^{-1}(\nu(o)) = 2\phi^{-1}(\mu(o)) + 2$, for every object o of I. Therefore, as by Definition 2.19, for every value a defined by ϕ, $\phi^{-1}(a)$ is a non-negative integer, $\phi^{-1}(\nu(o))$ is an even positive integer and hence by Definition 2.22 $\nu(o)$ is ϕ-even and ϕ-positive, for every object o of J.

Now let us prove that J is an instance and $\mathcal{P}$ is a proof for J. The two disordered instances $I = (Q, \omega, \mu)$ and $(Q, \omega, \mu +_\phi 1)$ and the two disordered instances $(Q, \omega, \mu +_\phi 1)$ and $J = (Q, \omega, (\mu +_\phi 1) \times_\phi 2)$ are $\mathit{Objects}(I)$-identical, due to Observation 2.25. Therefore, by Definition 2.15 for every comparison proposition c, c is satisfied by I if and only if c is satisfied by $(Q, \omega, \mu +_\phi 1)$ and also c is satisfied by $(Q, \omega, \mu +_\phi 1)$ if and only if c is satisfied by J. So, J satisfies $\mathcal{K}(I)$ because by assumption I is an instance and hence by Observation 2.5 I satisfies $\mathcal{K}(I)$. As a result, by Observation 2.5 J is an instance. Also, J satisfies $\mathcal{P}$ since by assumption $\mathcal{P}$ is a proof for I and so I satisfies $\mathcal{P}$. Thus, $\mathcal{P}$ if a proof for J, as well.

Now we construct an eliminating proof such that the e-value of its result is greater than values of all $\mathcal{P}$-visited objects. Suppose $a_1, \ldots, a_m$ is the sequence of ν-values of all objects of $N(J)$ visited by $\mathcal{P}$ in increasing order and $a_0 = -\infty$. We claim that for every i, $0 \leq i \leq m$, there exists a valid sequence Π_i of eliminating steps for J such that $\mathit{Visited}(J, \Pi_i) \subseteq \mathit{Visited}(\mathcal{P})$ and if $\mathit{result}(J, \Pi_i) = C_i = (\mathsf{e}_i, \mathsf{e}_{\mathsf{s}i}, \Phi_i)$ then $\mathsf{e}_i > a_i$ or $i = 0$. Defining Π_0 as the empty sequence of eliminating steps, by Definition 3.13 Π_0 is valid and by Definition 3.15 $\mathit{Visited}(I, \Pi_0) = \emptyset \subseteq \mathit{Visited}(\mathcal{P})$. Hence, the claim is true for $i = 0$. Also

if the claim is true for some i, the claim is true also for $i+1$, due to Lemma 4.26. So, Π_m is a sequence of eliminating steps such that $\mathit{Visited}(J, \Pi_m) \subseteq \mathit{Visited}(\mathcal{P})$, $\mathit{result}(J, \Pi_m) = C_m = (\mathsf{e}_m, \mathsf{e}_{\mathsf{s}m}, \Phi_m)$, and if $m > 0$ then $\mathsf{e}_m > a_m$, that is, e_m is greater than than the ν-value of every object of J visited by $\mathcal{P}$.

In order to use Lemma 4.27 to complete the eliminating proof, given a sub-intersection tree T of $N(Q)$, we first prove that T is a sub-intersection tree of Q and after that we show that T has at least two leaves. If Q has no speedy leaf, by the definition of the function N (Section 2.5) $N(Q) = Q$ and thus trivially T is a sub-intersection tree of Q. So, suppose there is a speedy leaf and hence the root is a union node. Then, $\top_{\ominus}(T)$ is a sub-intersection tree of $N(Q)[v]$, for a child v of the root. Also, since $N(Q)$ is constructed from Q by just deleting the speedy leaf, the complete subtree of Q rooted at v is the same as the complete subtree of $N(Q)$ rooted at v and thus $Q[v] = N(Q)[v]$. Therefore, $\top_{\ominus}(T)$ is a sub-intersection tree of $Q[v]$ while by the definition of $\top_{\ominus}$, $\top_{\ominus}(T)$ is constructed from T by just deleting the root. Hence, T is constructed by adding the root to a sub-intersection tree of $Q[v]$ where v is a child of the root and the root is a union node. So, T is a sub-intersection tree of Q.

Now we prove that every sub-intersection tree T of $N(Q)$ has at least two leaves by first showing that T has two children of one internal node. Since by assumption Q is normalized with at least two nodes, the root of Q is an internal node with at least two children. If the root is an intersection node, by the definition of the function N, $N(Q) = Q$ and so as by Definition 2.18 T contains the root, T also contains all children of the root which are at least two. We consider two cases based on the root of Q being a union node or an intersection node. First suppose the root is a union node. Then, every child l of Q that is a leaf is a speedy leaf and so l is not in $N(Q)$. Also, since we proved the root of Q has at least two children and by assumption at most one of these children is a speedy leaf, the root has at least one internal node v as as a child and so v is in $N(Q)$ as it is not a speedy leaf. Therefore, the root in $N(Q)$ has at least one child. Thus, by Definition 2.18 T contains one child u of the root in $N(Q)$. As the node u is in $N(Q)$, u is not a speedy leaf and so as by assumption the root is a union node and u is child of the root, u is an internal node. Hence, as the root is a union node, u is an intersection node. Also, since Q is normalized u has at least two children. As T contains the intersection node u, by Definition 2.18 T

contains all children of u which are at least two. So in any case we proved that there at least two children v_1 and v_2 of a single node are in T. It follows from Definition 2.18 that when T contains an internal node v, it contains a child of v and thus, recursively applying this definition, we may conclude that T contains a descendant of v that is a leaf. Hence, T contains one descendant of each of v_1 and v_2 that is a leaf. Also, as v_1 and v_2 are two children of the same node, they do not have any descendant in common. Therefore, T has two distinct leaves.

Now we add additional eliminating steps to Π_m, as we explained. Suppose $\mathcal{S}$ is the set of leaves l of $N(Q)$ such that the last object of $\omega(l)$ is an object visited by $\mathcal{P}$. For every sub-intersection tree U of $N(Q)$, as we proved U is a sub-intersection tree of Q and has at least two leaves. Hence, by Lemma 4.27 for every sub-intersection tree U of $N(Q)$, $\mathit{leaves}(U) \cap \mathcal{S} \neq \emptyset$. So, there exists a sub-union tree T of $N(Q)$ such that $\mathit{leaves}(T) \subseteq \mathcal{S}$, according to Lemma 2.21. Now we consider the sequence $\Pi_{m+1} = \Pi_m \odot \pi_T^-$. We prove that Π_{m+1} is valid, the e-value of its result on J is ∞, and $\mathit{Visited}(J, \Pi_{m+1}) \subseteq \mathit{Visited}(\mathcal{P})$. So, let us prove these claims, one by one. Since Π_m is valid, by Lemma 3.23 the result C_m of Π_m is valid. Hence, by Definition 3.8 $\pi_T^-(\nu, C_m)$ is defined. Thus, as Π_m is valid, by Observation 4.22 Π_{m+1} is also valid and $\mathit{result}(J, \Pi_{m+1}) = \pi_T^-(\nu, C_m)$. Suppose $\pi_T^-(\nu, C_m) = C_{m+1} = (\mathsf{e}_{m+1}, \mathsf{es}_{m+1}, \Phi_{m+1})$. We now prove that $\mathsf{e}_{m+1} = \infty$. Since $\mathit{leaves}(T) \subseteq \mathcal{S}$, by definition of $\mathcal{S}$ the last object of every leaf of T is $\mathcal{P}$-visited. Also, every object of every leaf of T is an object of $N(J)$ as T is a sub-union tree of $N(Q)$. Hence, as e_m is greater than the values of all objects of $N(J)$ visited by $\mathcal{P}$, the value of the last object of each leaf of T is less than e_m. Thus, for every leaf l of T, the value of every object of $\omega(l)$ is less than e_m. Therefore, as T is a subtree of $N(Q)$ and so all of its objects are in $N(J)$, by Definition 3.3 all objects of leaves of T are C_m-eliminated. So, by Definition 3.7, $\mathit{hmin}(\mu, C_m, T) = \infty$ and hence by Definition 3.8, $\mathsf{e}_{m+1} = \infty$.

Now we prove that $\mathit{Visited}(J, \Pi_{m+1}) \subseteq \mathit{Visited}(J, \Pi)$. Defining n as the length of Π_{m+1}, by Lemma 4.23 $\mathit{Visited}(J, \Pi_{m+1}) = \mathit{Visited}(J, \Pi_m) \cup \mathit{Visited}_n(J, \Pi_{m+1})$. Since we proved that $\mathit{Visited}(J, \Pi_m) \subseteq \mathit{Visited}(\mathcal{P})$, it suffices to show that $\mathit{Visited}_n(J, \Pi_{m+1}) \subseteq \mathit{Visited}(\mathcal{P})$. As the nth step in Π_{m+1} is π_T^-, by Definition 3.15 $\mathit{Visited}_n(J, \Pi_{m+1}) = \bigcup_{l \in \mathit{leaves}(T)} \{\mathit{find}(J, l, \mathsf{e}_m)\} \cup \mathcal{T}_l - \{\text{END}\}$ where $\mathcal{T}_l$ is the boundary-set of $(J, l, \langle \mathsf{e}_m \rangle)$, for every leaf l of T. We proved above that values of all objects of l are less than e_m, for

every leaf l of T. Thus by Lemma 3.1 $\mathit{find}(J, l, \mathsf{e}_m) = \text{END}$, for every leaf l of T. Now to prove $\mathit{Visited}(J, \Pi_{m+1}) \subseteq \mathit{Visited}(J, \Pi)$ it remains to show that, given a leaf l of T, $\mathcal{T}_l \subseteq \mathit{Visited}(\mathcal{P})$. To prove this, we show that $\mathcal{T}_l = \{o\}$ where o is the last object of l. Having proved this fact, as by the definition of T the last object of every leaf of T is $\mathcal{P}$-visited, we will have proved that $\mathcal{T}_l \subseteq \mathit{Visited}(\mathcal{P})$ and thus $\mathit{Visited}(J, \Pi_{m+1}) \subseteq \mathit{Visited}(\mathcal{P})$. Recall that $\mathcal{T}_l$ is the boundary-set of $(J, l, \langle \mathsf{e}_m \rangle)$. We consider the two cases $\mu(o) = \langle \mathsf{e}_m \rangle$ and $\mu(o) \neq \langle \mathsf{e}_m \rangle$. In the first case o is the biggest (the smallest) object of l with a value at most (at least) $\langle \mathsf{e}_m \rangle$ and thus the boundary-set of $(J, l, \langle \mathsf{e}_m \rangle)$ is $\{o\}$. Now consider the case $\mu(o) \neq \langle \mathsf{e}_m \rangle$. Since o is the last object of a leaf of T, as we showed, $\mu(o) < \mathsf{e}_m$ and hence by Observation 2.30 $\mu(o) \leq \langle \mathsf{e}_m \rangle$. So, as we are considering the case in which $\mu(o) \neq \langle \mathsf{e}_m \rangle$, $\mu(o) < \langle \mathsf{e}_m \rangle$. Thus, as o is the last object of l, l does not have any object of value at least $\langle \mathsf{e}_m \rangle$ and o is the biggest object of l with a value at most $\langle \mathsf{e}_m \rangle$. So, the boundary-set of $(J, l, \langle \mathsf{e}_m \rangle)$ is $\{o\}$. Therefore in any case $\mathcal{T}_l = \{o\}$ and as discussed, this result yields the fact that $\mathit{Visited}(J, \Pi_{m+1}) \subseteq \mathit{Visited}(J, \Pi)$.

Next, defining $\Pi_{m+2} = \Pi_{m+1} \odot \pi^\infty$, we prove that Π_{m+2} is an eliminating proof for J, visiting no object outside $\mathit{Visited}(\mathcal{P})$. Since we proved Π_{m+1} is valid, by Definition 3.23 the result C_{m+1} of Π_{m+1} on J is a valid eliminating configuration. Furthermore, we showed that the e-value of the result C_{m+1} of Π_{m+1} on J is ∞. So, by Definition 3.11 $\pi^\infty(\mu, C_{m+1})$ is defined and $\pi^\infty(\mu, C_{m+1}) = (\infty, \infty, \Phi_{m+2})$, for some Φ_{m+2}. Hence, by Observation 4.22 Π_{m+2} is valid and its result on J is $\pi^\infty(\mu, C_{m+1}) = (\infty, \infty, \Phi_{m+2})$. So, by the Definition 3.13 Π_{m+2} is an eliminating proof for J. Also, as $n+1$ is the length of Π_{m+2}, by Lemma 4.23 $\mathit{Visited}(J, \Pi_{m+2}) = \mathit{Visited}(J, \Pi_{m+1}) \cup \mathit{Visited}_{n+1}(J, \Pi_{m+2})$. Since the $n+1$st step of Π_{m+2} is π^∞, by Definition 3.15 $\mathit{Visited}_{n+1}(J, \Pi_{m+2}) = \emptyset$ and thus $\mathit{Visited}(J, \Pi_{m+2}) = \mathit{Visited}(J, \Pi_{m+1})$. Therefore, as we proved $\mathit{Visited}(J, \Pi_{m+1}) \subseteq \mathit{Visited}(\mathcal{P})$, we may conclude that $\mathit{Visited}(J, \Pi_{m+2}) \subseteq \mathit{Visited}(\mathcal{P})$. Thus Π_{m+2} is an eliminating proof for J not visiting any $\mathcal{P}$-skipped object and so as we showed I and J are $\mathit{Objects}(I)$-identical, by Theorem 4.5 Π_{m+2} is an eliminating proof for I not visiting any object outside $\mathit{Visited}(\mathcal{P})$. □

In this chapter we studied the relation between proofs and eliminating proofs of an instance and we proved that given a proof $\mathcal{P}$ of an instance one can build an eliminating proof visiting no object skipped by $\mathcal{P}$. In the next chapter we describe our algorithm and

in Chapter 6 we analyze the running time of the algorithm based on the sets of objects visited by eliminating proofs of the given instance as the input. Then, due to the result proved in Theorem 4.28 we will be able to compare the running time of the algorithm with gap-costs of proofs of the instance and so with the difficulty of the instance.

Chapter 5

The Algorithm

In this chapter we present an algorithm for query trees with at most one speedy leaf. For instances with more than one speedy leaf, this algorithm should be combined with the algorithm designed by Erik D. Demaine et al. [DLOM00]. Compared to our lower bound, the running time has an additional factor that is roughly logarithmic in the size of the query tree.

5.1 Overview

We first present basic definitions and explain general ideas of the algorithm. Throughout the chapter we suppose that $I = (Q, \omega, \mu)$ is a normalized instance given as the input to the algorithm and that Q has at most one speedy leaf. Also, to avoid considering additional special cases in proofs, we define $\mu(\text{BEG}) = -\infty$ and $\mu(\text{END}) = \infty$. The value of a variable v of the algorithm at a specific time t during the execution of the algorithm is denoted by $\mathsf{v}^{(t)}$. In this representation the superscript "(t)" is called a *time stamp*. We will express as "invariants" a number of provable facts about contents of the variables during running the algorithm but they are not used in formal arguments as proved facts before their correctness is shown. Moreover, for every variable v that we define for the algorithm, we express some conditions that should be satisfied by what is stored in v and, at any time during execution of the algorithm, we will say v is well-valued if those conditions hold. For a procedure in the code of the algorithm, we may define a set of preconditions,

a set of changing variables, and a set of postconditions. In such cases, we will prove that if before calling the procedure its preconditions hold, the procedure, during its execution, does not modify any variable other than those that are defined as changing variables of the procedure and after executing the procedure its postconditions hold. During the execution of the algorithm, whenever the algorithm calls a procedure we say the procedure call is *legal* if preconditions of the procedure hold before calling the procedure.

The algorithm is presented in Figures 5.1 through 5.6. The main idea of the algorithm is to consider a valid eliminating configuration and to change it gradually until it is transformed to a valid eliminating configuration in which all objects are eliminated. Then, the expansion of the output of the eliminating configuration is a solution to the problem, according to Lemma 3.9, and so the algorithm has solved the problem.

Now we explain how the eliminating configuration is stored and how it is changed. Variables e, $\mathsf{e_s}$, and Φ will be used to store the e-value, the $\mathsf{e_s}$-value, and the output of this eliminating configuration and at any time C is defined as this eliminating configuration, that is, $\mathsf{C} = (\mathsf{e}, \mathsf{e_s}, \Phi)$.

Invariant 1 *C is a valid eliminating configuration.*

At the beginning, the algorithm sets $\mathsf{e} = \mathsf{e_s} = -\infty$, and Φ equal to the empty sequence. If the algorithm reaches a point in which all objects are eliminated, then the algorithm has finished its job as explained. If at some point e becomes equal to ∞, by Definition 3.3 all objects of non-speedy leaves are eliminated. Then, if there is a speedy leaf, as we will see, the algorithm can easily eliminate objects of the speedy leaf in a simple way and get a valid eliminating configuration in which all objects are eliminated. So, the algorithm aims to increase e until e becomes as large as ∞. In the meantime, $\mathsf{e_s}$ and Φ change so that C remains a valid eliminating configuration.

In order to increase e, the algorithm first evaluates the membership of e in the value set of the root. This will help the algorithm to add to the output an object of value e if necessary and then to update e. By Lemma 2.22, to evaluate the membership of e in the value set of the root, it suffices to find a sub-intersection tree in which every leaf has an object of value e or to find a sub-union tree in which no leaf has an object of value e. In order to know whether a leaf has an object of value e or not, one should find the smallest

object of value e of the leaf and see if its value is e. So the algorithm tries to find the smallest object of value at least e in $\omega(l)$, that is to compute $find(I, l, \mathtt{e})$ (Lemma 3.1), for different leaves l. To do this, for every leaf l, the algorithm uses a variable fi$[l]$ to remember the biggest object of l of value less than e that has already been compared with e. If the value of the last object of l is compared with e and is less than e then fi$[l]$ is set equal to END; otherwise fi$[l]$ is set equal to the object of l following the biggest object of l whose value is known to be less than e.

Definition 5.1 The head array *fi is an array such that* fi$[l]$ *is an object of* $\omega(l)$ *or it equals* END*, for every leaf* l *of* Q*. Every object of* $\omega(l)$ *with a value less than* $\mu($fi$[l])$ *is* scanned *and the rest of objects of* l *are* waiting objects*, for every leaf* l *of* Q*. Also, for a time* t*, by a* t-scanned object *(a* t-waiting objects*) we mean an object that is scanned (is waiting) at the time* t*.*

Throughout the chapter, we suppose that the elements of fi satisfy the above definition, that is, fi$[l]$ is either END or an object of l, for every leaf l. Now we explain how we ensure that this is a correct assumption. As we will see, at the beginning fi$[l]$ is initialized to the first objects of l, for every leaf l. Also there are only three lines in the algorithm modifying the elements of fi after initialization. For each of these lines, we will prove that when the line modifies fi$[l]$, for a leaf l, the line sets fi$[l]$ equal to END or an object of l. So, the assumption is correct.

The algorithm each time selects a leaf l and tries to find a prefix of the sequence of waiting objects of l in which all objects have values less than e and then modifies fi$[l]$ so that objects of this prefix become scanned. By repeating this operation for a specific leaf l, the algorithm can compute $find(I, l, \mathtt{e})$, as we will show. The next lemma shows how the algorithm can identify the sequence of waiting objects of a given leaf.

Lemma 5.1 *Given a leaf* l*, if* l *has any waiting object,* fi$[l]$ *is the smallest waiting object of* l*.*

Proof. We consider two cases fi$[l]$ = END and fi$[l] \neq$ END. In the first case, for every object o of l, $\mu(o) < \infty = \mu(\text{END}) = \mu(fi[l])$ and so by Definition 5.1 o is not a waiting object. So in this case, there is no waiting object and hence the lemma is correct.

Now suppose $\mathfrak{fi}[l] \neq \text{END}$ and thus by Definition 5.1, $\mathfrak{fi}[l]$ is an object. Then, since μ is ordered, for every object o of l that is before $\mathfrak{fi}[l]$ in $\omega(l)$, $\mu(o) < \mu(\mathfrak{fi}[l])$ and hence by Definition 5.1 o is not a waiting object. Also, as $\mu(\mathfrak{fi}[l]) \leq \mu(\mathfrak{fi}[l])$, by definition $\mathfrak{fi}[l]$ is waiting. Thus, $\mathfrak{fi}[l]$ is the smallest waiting object of l. □

An important problem is how to divide the processing time among different leaves l to process $\omega(l)$ and change $\mathfrak{fi}[l]$ so that more objects with values less than e in $\omega(l)$ become scanned. We will define two recursive procedures, called visit procedures, namely Visit$^-$, the negative visit algorithm, and Visit$^+$, the positive visit algorithm. Every visit procedure has a parameter that is a node of the query tree. At any moment that Visit$^-(v)$ or Visit$^+(v)$ is being executed, for a node v, we say v is *being negative-visited* or *being positive-visited*, respectively, or just is *being visited*. When a leaf l is being visited, a prefix of the sequence of waiting objects of $\omega(l)$ of values less than e is selected its objects become scanned. When an internal node is being visited, a number of its children are selected and are visited recursively.

Procedures Visit$^+$ and Visit$^-$ have different policies in selecting the next child of the vertex being visited to be visited and these two procedures are optimized for when e is not in the value set of the root in $N(I)$ and when e is in the value set of the root in I, respectively. Since by Observation 2.32 the value set of the root in I is a subset of the value set of the root in $N(I)$, at any time at least one of these two situations is the case. So, informally, at any time one of these two procedures works optimally. Hence, as the algorithm can not decide which one is the optimal one at a specific time, the algorithm alternates between calling these two procedures with the root as parameter. This alternation will be so that these two procedures roughly take the same amount of the time and thus running the non-optimal one increases the running time of the whole algorithm by only a constant factor.

5.2 Updating the Eliminating Configuration

In this section we explain more about our ways of updating the eliminating configuration by the algorithm so that C remains valid and e is increased. We first explain the idea of evaluation of $\mathit{find}(I, l, \mathsf{e})$ for different leaves l of Q. Then, we introduce two functions

that can be evaluated using the result of the expression $\mathit{find}(I, l, \mathsf{e})$, for different leaves l of Q. These functions will be so that, having evaluated them, the algorithm will be able to update C and increase e. Then, in the next section we will discuss how to evaluate the two functions that we talked about, more extensively. As mentioned, each time that the algorithm visits a leaf l, it sets $\mathfrak{fi}[l]$ equal to a bigger object and in this way makes more objects of l scanned. However, as the algorithm aims to find the smallest object of value at least e, the algorithm never makes an object of value at least e scanned.

Invariant 2 *The value of every scanned object is less than* e.

Since the definition of being scanned only depends on contents of the array $\mathfrak{fi}$, the following observation follows from the definition of Invariant 2.

Observation 5.2 *The correctness of Invariant 2 depends only on contents of variables* $\mathfrak{fi}$ *and* e.

When Invariant 2 holds and all objects with values less than e are scanned, the algorithm has evaluated $\mathit{find}(I, l, \mathsf{e})$ as the next lemma shows.

Definition 5.2 *When all objects of a leaf* l *with values less than* e *are scanned, we say* l *is* satiated.

Lemma 5.3 *Suppose Invariant 2 holds and* l *is a satiated leaf. Then,* $\mathfrak{fi}[l] = \mathit{find}(I, l, \mathsf{e})$.

Proof. We consider two cases based on l having an object of value at least e or not. If l does not have such an object, all objects of l have values less than e and so, as by assumption l is satiated, by Definition 5.2 objects of l are all scanned. We now prove that $\mathfrak{fi}[l] = \textsc{end}$. Assume to the contrary that this is not true and hence by Definition 5.1 $\mathfrak{fi}[l]$ is an object of l. Then, since the value of $\mathfrak{fi}[l]$ is not less than $\mu(\mathfrak{fi}[l])$, $\mathfrak{fi}[l]$ is not scanned while we proved all object of l are scanned, a contradiction. Thus, the claim $\mathfrak{fi}[l] = \textsc{end}$ is true. In addition, as by assumption there is no object of value at least e in $\omega(l)$, by Lemma 3.1 $\mathit{find}(I, l, \mathsf{e}) = \textsc{end}$. Therefore, $\mathit{find}(I, l, \mathsf{e}) = \mathfrak{fi}[l]$.

Now consider the other case, in which there is an object of value at least e in $\omega(l)$. Then, by Lemma 3.1 $\mathit{find}(I, l, \mathsf{e})$ is the smallest object of l with a value at least e. So, as

every object before $\mathit{find}(I, l, \mathsf{e})$ has a value less than e and l is satiated, every object before $\mathit{find}(I, l, \mathsf{e})$ is scanned, according to Definition 5.2. Furthermore, since by assumption Invariant 2 holds and $\mathit{find}(I, l, \mathsf{e})$ has a value at least e, by Invariant 2 $\mathit{find}(I, l, \mathsf{e})$ is not scanned. Thus, $\mathit{find}(I, l, \mathsf{e})$ is the smallest waiting object of l. Therefore, by Lemma 5.1 $\mathfrak{fi}[l] = \mathit{find}(I, l, \mathsf{e})$. □

The next lemma shows how the algorithm can know if a leaf l is satiated.

Lemma 5.4 *A leaf l is satiated if and only if $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}$.*

Proof. We first prove the "if" part of the claim and then we prove the "only if" part. When $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}$, every object o of value less than e has a value less than $\mathfrak{fi}[l]$ and thus by Definition 5.1 o is scanned. Hence, in this case by Definition 5.2 l is satiated.

To prove the "only if" part we consider the two cases $\mathfrak{fi}[l] = \text{END}$ and $\mathfrak{fi}[l] \neq \text{END}$. In the first case, as we defined, $\mu(\mathfrak{fi}[l]) = \mu(\text{END}) = \infty$ and by Observation 2.28 $\infty \geq \mathsf{e}$. So, $\mu(\mathfrak{fi}[l]) \leq \mathsf{e}$. Now suppose $\mathfrak{fi}[l] \neq \text{END}$. Then, since $\mu(\mathfrak{fi}[l]) \not< \mu(\mathfrak{fi}[l])$, by Definition 5.1 $\mathfrak{fi}[l]$ is not a scanned object. Thus, as l is satiated, by Definition 5.2 $\mu(\mathfrak{fi}[l])$ is not less than e. Consequently, $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}$. □

We now explain more precisely how the algorithm can use satiated leaves to increase e. We consider two possibilities that may overlap: when e is not in the value set of the root in $N(I)$ and when e is in the value set of the root in I. First we discuss the case in which e is not in the value set of the root in $N(I)$. The algorithm makes leaves of Q satiated such that if e is not in the value set of the root in $N(I)$, a sub-union tree T of $N(Q)$ is found such that no leaf of T has an object of value e and all leaves of T are satiated. Then, as we prove in the next lemma, the algorithm can easily evaluate $\mathit{hmin}(\mu, \mathsf{C}, T)$ by finding the minimum of $\mu(\mathfrak{fi}[l])$ over all leaves l of T. Then, having evaluated $\mathit{hmin}(\mu, \mathsf{C}, T)$, the algorithm can compute $\pi^-_T(\mu, \mathsf{C})$, as described in Definition 3.8. After that, the algorithm sets C equal to $\pi^-_T(\mu, \mathsf{C})$ and in this way e is increased as we will see.

Definition 5.3 *Given a node v of $N(Q)$ and a sub-union tree T of $N(Q)[v]$, min-wait(T) is defined as $\min_{l \in \mathit{leaves}(T)} \mu(\mathfrak{fi}[l])$.*

Lemma 5.5 *Suppose Invariant 2 holds. Also, suppose T is a sub-union tree of $N(Q)[v]$, for a node v of $N(Q)$, such that every leaf of T is satiated. Then, $\mathit{hmin}(\mu, \mathsf{C}, T) = \mathit{min\text{-}wait}(T)$.*

Proof. Defining $\mathcal{S}$ as the set of all leaves of T such that $\omega(l)$ has an object of value at least e, we consider two cases, $\mathcal{S} = \emptyset$ and $\mathcal{S} \neq \emptyset$, separately. In the first case, as $\mathcal{S} = \emptyset$, by definition of $\mathcal{S}$ there is no object of value at least e in $\omega(l)$, for any leaf l of T. As T is a subtree of $N(Q)$, all leaves of T are in $N(Q)$. Hence, as values of all objects of $\omega(l)$ are less than e, by Definition 3.3 there is no C-remaining object in $\omega(l)$, for any leaf l of T. Therefore, by Definition 3.7 $\mathit{hmin}(\mu, \mathsf{C}, T) = \infty$. Furthermore, for a leaf l of T, since as we proved there is no object of value at least e in $\omega(l)$, by Lemma 3.1 $\mathit{find}(I, l, \mathsf{e}) = \text{END}$. Also, since by assumption every leaf of T is satiated, by Lemma 5.3 $\mathfrak{fi}[l] = \mathit{find}(I, l, \mathsf{e})$, for all leaves l of T. Hence, $\mathfrak{fi}[l] = \mathit{find}(I, l, \mathsf{e}) = \text{END}$, for every leaf l of T. Thus, by Definition 5.3 $\mathit{min\text{-}wait}(T) = \min_{l \in \mathit{leaves}(T)} \mu(\mathfrak{fi}[l]) = \min_{l \in \mathit{leaves}(T)}(\mu(\text{END})) = \infty$. So, as we proved earlier that $\mathit{hmin}(\mu, \mathsf{C}, T) = \infty$, $\mathit{hmin}(\mu, \mathsf{C}, T) = \mathit{min\text{-}wait}(T)$, completing proof for this case.

Now suppose $\mathcal{S} \neq \emptyset$ and consider a member v of $\mathcal{S}$ minimizing $\mu(\mathit{find}(I, v, \mathsf{e}))$ over all members of $\mathcal{S}$. We prove that $\mu(\mathit{find}(I, v, \mathsf{e})) \leq \mu(\mathit{find}(I, l, \mathsf{e}))$, for every leaf l of T, and then we conclude that $\mathit{hmin}(\mu, \mathsf{C}, T)$ and $\mathit{min\text{-}wait}(T)$ both equal $\mu(\mathit{find}(I, v, \mathsf{e}))$. Given a leaf l of T, in order to prove $\mu(\mathit{find}(I, v, \mathsf{e})) \leq \mu(\mathit{find}(I, l, \mathsf{e}))$, we consider the two cases $l \in \mathcal{S}$ and $l \notin \mathcal{S}$. If $l \in \mathcal{S}$ then by definition of v, $\mu(\mathit{find}(I, v, \mathsf{e})) \leq \mu(\mathit{find}(I, l, \mathsf{e}))$. If $l \notin \mathcal{S}$, by definition of $\mathcal{S}$ there is no object of value at least e in $\omega(l)$ and so by Lemma 3.1 $\mathit{find}(I, l, \mathsf{e}) = \text{END}$. Therefore, $\mu(\mathit{find}(I, l, \mathsf{e})) = \infty \geq \mathit{find}(I, v, \mathsf{e})$ (Observation 2.28). Consequently, in both cases $\mu(\mathit{find}(I, v, \mathsf{e})) \leq \mu(\mathit{find}(I, l, \mathsf{e}))$.

Next we prove that $\mathit{min\text{-}wait}(T) = \mu(\mathit{find}(I, v, \mathsf{e}))$ by first showing that v is a leaf minimizing $\mu(\mathfrak{fi}[l])$ over all leaves of T and then by using the definition to prove the claim. As by assumption all leaves l of T are satiated, by Lemma 5.3 $\mathfrak{fi}[l] = \mathit{find}(I, l, \mathsf{e})$, for every leaf l of T. So, as v is a leaf of T, $\mu(\mathfrak{fi}[v]) = \mu(\mathit{find}(I, v, \mathsf{e})) \leq \mu(\mathit{find}(I, l, \mathsf{e})) = \mu(\mathfrak{fi}[l])$, for every leaf l of T. Hence v is a leaf l of T minimizing $\mu(\mathfrak{fi}[l])$ over all leaves of T and thus by Definition 5.3 $\mathit{min\text{-}wait}(T) = \mu(\mathfrak{fi}[v]) = \mu(\mathit{find}(I, v, \mathsf{e}))$.

Finally, we complete the proof by showing that $\mathit{hmin}(\mu, \mathsf{C}, T) = \mu(\mathit{find}(I, v, \mathsf{e}))$. We first prove that $\mathcal{S}$ is the set of all leaves l of T with $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$ and then the correctness of lemma will follow from definitions of v and the function *hmin*. As $\mathcal{S}$ is the set of all

leaves with at least one object of value at least e, due to Lemma 3.1, for every leaf l, $l \in \mathcal{S}$, *find*(I, l, e) is an object of l and so *find*$(I, l, \mathsf{e}) \neq$ END. Also, for every leaf l, $l \notin \mathcal{S}$, by definition l has no object of value at least e and thus by Lemma 3.1 *find*$(I, l, \mathsf{e}) =$ END. So, as claimed, $\mathcal{S}$ is the set of all leaves l of T with *find*$(I, l, \mathsf{e}) \neq$ END. Hence, as by assumption v is a member l of $\mathcal{S}$ minimizing $\mu($*find*$(I, l, \mathsf{e}))$ over all leaves in $\mathcal{S}$, by Definition 3.7 *hmin*$(\mu, \mathsf{C}, T) = \mu($*find*$(I, v, \mathsf{e}))$. Thus, *hmin*$(\mu, \mathsf{C}, T) = \mu($*find*$(I, v, \mathsf{e})) =$ *min-wait*(T). □

Having found a sub-union tree of $N(Q)$ in which all leaves are satiated, the algorithm can use eliminating steps of the first type to increase e, as we now explain. Suppose Invariants 1 and 2 hold. Given a sub-union tree T of $N(Q)$, as by Definition 5.3 *min-wait*(T) is the minimum of $\mu(\mathfrak{fi}[l])$ over all leaves l of T, if *min-wait*$(T) > \mathsf{e}$ then for every leaf l of T, $\mu(\mathfrak{fi}[l]) > \mathsf{e}$ and so l is satiated (Lemma 5.4). Hence, in this situation, by Lemma 5.5 *hmin*$(\mu, C, T) =$ *min-wait*(T). Thus when *min-wait*$(T) > \mathsf{e}$, by Definition 3.8 the algorithm can easily set C equal to $\pi_T^-(\mu, \mathsf{C})$ by setting e equal to *min-wait*(T) and if the query tree has no speedy leaf by setting $\mathsf{e_s}$ equal to *min-wait*(T) also. Then since by assumption before these changes Invariant 1 held and so C was valid, by Lemma 3.12 after these changes C is valid and so Invariant 1 holds. This proves the next lemma.

Lemma 5.6 *Suppose Invariants 1 and 2 hold and consider a sub-union tree T of $N(Q)$ such that* min-wait$(T) > \mathsf{e}$. *Then, if the algorithm sets e equal to* min-wait(T) *and if the query tree has no speedy leaf it sets $\mathsf{e_s}$ equal to* min-wait(T) *also, Invariant 1 remains satisfied.* □

The algorithm is designed so that it finds $\max_T$ *min-wait*(T), where the maximum is taken over all sub-union trees T of $N(Q)$, and then by the approach explained in Lemma 5.6 the algorithm increases e. To explain how this maximum is computed, we first generalize its concept to all nodes, as follows.

Definition 5.4 *Given a node v of $N(Q)$ we define* maxmin(v) *to be the maximum of* min-wait(T) *over all sub-union trees T of $N(Q)[v]$.*

The following observation follows immediately from Definition 5.4.

Observation 5.7 *Given a node v of $N(Q)$ and a sub-union tree T of $N(Q)[v]$,* min-wait$(T) \leq$ *maxmin*(v).

Since by Definition 5.4 *maxmin*(root) equals *min-wait*(T), for a sub-union tree T of $N(Q)[\text{root}] = N(Q)$, the following observation follows immediately from Lemma 5.6.

Observation 5.8 *Suppose Invariants 1 and 2 are true and* maxmin(root) $>$ e. *If the algorithm sets* $\mathsf{e} :=$ *maxmin*(root) *and provided Q has no speedy leaf it sets* $\mathsf{e_s} :=$ *maxmin*(root) *then Invariant 1 remains satisfied.*

In the next section, we will discuss how to evaluate *maxmin*(root) and how to use Observation 5.8 to increase e.

Now we explain our approach for increasing e in the situation in which e is in the value set of the root in I. In this situation, by Lemma 2.22 there is a sub-intersection tree T of Q such that for every leaf l of T, $\omega(l)$ has an object of value e. If the algorithm finds such a sub-intersection tree, it can use the step π_T^+ to increase e, as stated in Definition 3.10. In order to find such a sub-intersection tree T, the algorithm has to find objects of value e in all leaves of T to ensure that every leaf of T has an object of value e. By Lemma 3.2 the object of value e in $\omega(l)$ is *find*(I, l, e), for every leaf l of T. Also, by Lemma 5.3 when Invariant 2 holds and l is a satiated leaf, $\text{fi}[l] =$ *find*(I, l, e). Due to these facts, the algorithm tries to find a sub-intersection tree T of Q such that $\mu(\text{fi}[l]) = \mathsf{e}$, for every leaf l of T.

Definition 5.5 *Given a node v of Q, a sub-intersection tree T of $Q[v]$ is an* eyewitness *for v if fi$[l]$ is an object of value e, for every leaf l of T.*

Once the algorithm has found an eyewitness for the root, the algorithm can update C and increase e as the next lemma shows.

Lemma 5.9 *Suppose Invariant 1 is true and there is an eyewitness T for the root. If the algorithm sets* $\mathsf{C} := (\mathsf{e}_+, \mathsf{e}_+, \Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (v, i, i))$ *such that $\omega(v)[i]$ is an object of value e, C remains a valid eliminating configuration.*

Proof. We first prove that $\pi_T^+(\mu, \mathsf{C})$ is defined and then we use the similarity between $\pi_T^+(\mu, \mathsf{C})$ and the new eliminating configuration being assigned to C to prove that C will be valid. As by assumption e is the value of the object $\omega(v)[i]$, by Definition 2.5 e is member of $\mathbb{V}$ and $\mathsf{e} \neq \infty$. Also, since by assumption T is an eyewitness, by Definition 5.5

$\mu(\text{fi}[l]) = \mathsf{e}$, for all leaves l of T. Therefore, since $\mathsf{e} \neq \infty = \mu(\text{END})$, $\text{fi}[l] \neq \text{END}$ and hence, by Definition 5.1 $\text{fi}[l]$ is an object of l, for every leaf l of T. So every leaf of T has an object of value e. Also, as we proved $\mathsf{e} \in \mathbb{V}$, e is a main value. Thus, by Definition 3.10 $\pi^+_T(\mu, \mathsf{C})$ is defined and hence according to Lemma 3.20, $\pi^+_T(\mu, \mathsf{C})$ is a valid eliminating configuration.

We now prove that sequences of values of objects in expansions of outputs of $\pi^+_T(\mu, \mathsf{C})$ and the new eliminating configuration being assigned to C are the same and then we use the fact that $\pi^+_T(\mu, \mathsf{C})$ is valid to show the correctness of the lemma. By Definition 3.10 $\pi^+_T(\mu, \mathsf{C}) = (\mathsf{e}+, \mathsf{e}+, \Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (l_T, j, j))$ where $j = pos(I, l_T, \mathsf{e})$ and l_T is a leaf of T. Since l_T is a leaf of T, as we proved, l_T has an object of value e and thus $find(I, l_T, \mathsf{e})$ is an object of value e, according to Lemma 3.2. Hence, as by Definition 3.4 $find(I, l_T, \mathsf{e}) = \omega(l_T)[pos(I, l_T, \mathsf{e})] = \omega(l_T)[j]$, the value of $\omega(l_T)[j]$ is e. So, $\omega(v)[i]$ and $\omega(l_T)[j]$ both are objects of value e. Also, defining $\Sigma = o_1, \ldots, o_n$ as the expansion of $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e})$, we can conclude from Observation 2.3 that expansions of $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (l_T, j, j)$ and $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (v, i, i)$ are $\Sigma \circ \Gamma_1$ and $\Sigma \circ \Gamma_2$, respectively, where Γ_1 and Γ_2 are sequences consisting of just one object $\omega(l_T)[j]$ and just one object $\omega(v)[i]$, respectively. Hence, by Definition 2.1 expansions of $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (l_T, j, j)$ and $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (v, i, i)$ are $o_1, o_2, \ldots, o_n, \omega(l_T)[j]$ and $o_1, o_2, \ldots, o_n, \omega(v)[i]$, respectively, where $\mu(\omega(l_T)[j]) = \mu((\omega(v)[i])$, as we proved. Thus, the sequence of values of objects of the expansion of $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (l_T, j, j)$ and that of $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (v, i, i)$ are the same. Hence, by Lemma 3.7 $(\mathsf{e}+, \mathsf{e}+, \Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \circ (v, i, i))$ is valid since, as we showed, $\pi^+_T(\mu, \mathsf{C}) = (\mathsf{e}+, \mathsf{e}+, \Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \circ (l_T, j, j))$ is valid. □

We now discuss the conditions that should hold so that the algorithm can use Lemma 5.9 to increase e in an efficient manner. As Lemma 5.9 shows, when e is in the value set of the root and we want to increase e using an eliminating step of the second type, the algorithm must evaluate $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e})$. By Definition 3.9 in order to compute $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e})$, the algorithm should evaluate $pos(I, sl(Q), \mathsf{e_s})$ first, when Q has a speedy leaf. Also due to Definition 3.4, $pos(I, sl(Q), \mathsf{e_s})$ is in fact the index of $find(I, sl(Q), \mathsf{e_s})$ in $\omega(sl(Q))$ and so to evaluate $pos(I, sl(Q), \mathsf{e_s})$ it suffices to compute $find(I, sl(Q), \mathsf{e_s})$. Motivated by these facts, the algorithm is designed in such a way that when there is a speedy leaf, the algorithm updates $\mathsf{e_s}$ so that the equation $find(I, sl(Q), \mathsf{e_s}) = \text{fi}[sl(Q)]$ holds at almost any point during the execution of the algorithm. In this way, the algorithm can evaluate $pos(I, sl(Q), \mathsf{e_s})$ by considering the index of $\text{fi}[sl(Q)]$ in $\omega(sl(Q))$. So, we design the algorithm such that the

following invariant holds.

Invariant 3 *If Q has a speedy leaf then $\mathfrak{fi}[sl(Q)] = find(I, sl(Q), \mathsf{e}_s)$.*

In order to use Lemma 5.9 to increase e, first the algorithm has to ensure that there is an eyewitness for the root. Before explaining how the algorithm checks whether there is an eyewitness for the root, we explain how the algorithm can recursively check if a certain sub-intersection tree T is an eyewitness. We discuss the two following situations separately: When the root of T is a union node and when the root of T is an intersection node. Recall the function $\top_\ominus$ from Section 2.4.1.

Lemma 5.10 *Given a sub-intersection tree T of $Q[v]$, for a union node v of Q, T is an eyewitness if and only if $\top_\ominus(T)$ is an eyewitness.*

Proof. We prove the lemma by showing that the definition of a sub-intersection tree being an eyewitness is based on its set of leaves and by arguing that sets of leaves of T and $\top_\ominus(T)$ are the same. Defining $\mathcal{S}$ as the set of leaves l of Q satisfying $\mu(\mathfrak{fi}[l]) = \mathsf{e}$, by Definition 5.5, T is an eyewitness if $leaves(T) \subseteq \mathcal{S}$. Moreover, by Definition 5.5, $\top_\ominus(T)$ is an eyewitness if $leaves(\top_\ominus(T)) \subseteq \mathcal{S}$. Hence, as by Observation 2.15 $leaves(T) = leaves(\top_\ominus(T))$, T is an eyewitness if and only if $\top_\ominus(T)$ is an eyewitness. □

Lemma 5.11 *Given a sub-intersection tree T of $Q[v]$, for an intersection node v of Q, T is an eyewitness if and only if $\top_\ominus(T, u)$ is an eyewitness, for every child u of v.*

Proof. To prove the lemma we first show that the set of leaves of T is the union of sets of leaves of $\top_\ominus(T, u)$, for all children u of v. Then, similar to the technique used in the previous lemma, we use the fact that the definition of a sub-intersection tree being an eyewitness is based on its set of leaves. Defining $\mathcal{S}$ as the set of leaves l of Q satisfying $\mu(\mathfrak{fi}[l]) = \mathsf{e}$, by Definition 5.5 T is an eyewitness if and only if $leaves(T) \subseteq \mathcal{S}$. Also, defining $\{u_1, \ldots, u_k\}$ to be the set of children of v, since by Observation 2.13 $T = \top_\oplus(\top_\ominus(T, u_1), \ldots, \top_\ominus(T, u_k))$, by Observation 2.16, $leaves(T) = \bigcup_{i=1}^{k} leaves(\top_\ominus(T, u_i))$. Hence, T is an eyewitness if and only if $\bigcup_{i=1}^{k} leaves(\top_\ominus(T, u_i)) \subseteq \mathcal{S}$, that is, $leaves(\top_\ominus(T, u_i)) \subseteq \mathcal{S}$, for every i, $1 \leq i \leq k$. By Definition 5.5, $leaves(\top_\ominus(T, u_i)) \subseteq \mathcal{S}$ if and only if $\top_\ominus(T, u_i)$ is an eyewitness, for every

i, $1 \leq i \leq k$. Therefore, T is an eyewitness if and only if $\top_{\ominus}(T, u_i)$ is an eyewitness, for every i, $1 \leq i \leq k$. □

In the next section we will see how we can use above results to keep track of nodes having eyewitnesses.

5.3 Evaluating *maxmin* and Finding Eyewitness

In this part we explain how the algorithm evaluates the function *maxmin*, how the algorithm checks if an eyewitness exists, and how these results are used to increase e. Given a node v, the evaluation of *maxmin* and the task of testing the existence of an eyewitness for v are accomplished using variables $\mathsf{M}[v]$ and $\mathsf{W}[v]$, respectively. The variable $\mathsf{M}[v]$ is used to store $\mathit{maxmin}(v)$, for every node v of $N(Q)$. The variable $\mathsf{W}[v]$ is used for determining if there is an eyewitness for v, for every node v of Q. This variable stores a member of $\{\text{BEG}\} \cup \mathit{Objects}(I)$ and when certain conditions hold, $\mathsf{W}[v]$ is an object of value e if and only if v has an eyewitness. Initially $\mathsf{W}[v]$ is set equal to BEG, for every node v of Q. We now explain how the value of each of these variables is evaluated.

We first talk about the variable $\mathsf{M}[v]$, for a node v of $N(Q)$, which is defined to store the value $\mathit{maxmin}(v)$. As we will prove, when v is a leaf, $\mathit{maxmin}(v) = \mu(\mathfrak{fi}[v])$ and when v is a union (intersection) node, $\mathit{maxmin}(v)$ is the maximum (the minimum) of $\mathit{maxmin}(u)$ over all children u of v in $N(Q)$.

Definition 5.6 *Given a node v of $N(Q)$, $\mathsf{M}[v]$ is* well-valued *if one the following conditions holds:*

1. *The node v is a leaf and $\mathsf{M}[v] = \mu(\mathfrak{fi}[v])$.*
2. *The node v is a union node and $\mathsf{M}[v]$ is the minimum of $\mathsf{M}[u]$, for all children u of v in $N(Q)$.*
3. *The node v is an intersection node and $\mathsf{M}[v]$ is equal to the maximum value of $\mathsf{M}[u]$, for all children u of v in $N(Q)$.*

Before proving formally that what is explained in Definition 5.6 is the right way to recursively evaluate the function *maxmin*, we first discuss how to evaluate the function *min-wait* recursively as the definition of the function *maxmin* is based in the definition of the function *min-wait*.

Lemma 5.12 *Suppose T is a sub-union tree rooted at an intersection node v in $N(Q)$. Then, min-wait(T) = min-wait$(\top_{\ominus}(T))$.*

Proof. Since by Observation 2.15 *leaves*(T) = *leaves*$(\top_{\ominus}(T))$, according to Definition 5.3 *min-wait*$(T) = \min_{l \in \mathit{leaves}(T)} \mu(\mathfrak{fi}[l]) = \min_{l \in \mathit{leaves}(\top_{\ominus}(T))} \mu(\mathfrak{fi}[l]) =$ *min-wait*$(\top_{\ominus}(T))$. □

Lemma 5.13 *Suppose $T = \top_{\oplus}(U_1, \ldots, U_k)$ is a sub-union tree of a node in $N(Q)$. Then, min-wait$(T) = \min_{i=1}^{k}$ min-wait(U_i).*

Proof.

min-wait(T)	$= \min_{l \in \mathit{leaves}(T)} \mu(\mathfrak{fi}[l])$	by Definition 5.3
	$= \min_{l \in \bigcup_{i=1}^{k} \mathit{leaves}(U_i)} \mu(\mathfrak{fi}[l])$	by Observation 2.16
	$= \min_{i=1}^{k} \min_{l \in \mathit{leaves}(U_i)} \mu(\mathfrak{fi}[l])$	
	$= \min_{i=1}^{k}$ *min-wait*(U_i)	by Definition 5.3.

□

We now can prove that if the array M is filled so that all its elements are well-valued then M[root] represents the value of *maxmin*(root).

Lemma 5.14 *Given a node v of $N(Q)$, if for every node u in $N(Q)[v]$, $\mathsf{M}[u]$ is well-valued then $\mathsf{M}[v]$ = maxmin(v).*

Proof. We use induction on the height of v to prove the lemma. The base case is when v is a leaf. Then, since by definition v is in $N(Q)[v]$, by assumption $\mathsf{M}[v]$ is well-valued and hence by Definition 5.6 $\mathsf{M}[v] = \mu(\mathfrak{fi}[v])$. Also, as v is a leaf, $N(Q)[v]$ has only on sub-union tree T having just one leaf v. Thus, by Definition 5.3, *min-wait*$(T) = \mu(\mathfrak{fi}[v])$ and hence by Definition 5.4 *maxmin*(v) = *min-wait*$(T) = \mu(\mathfrak{fi}[v])$. So, as we proved $\mathsf{M}[v] = \mu(\mathfrak{fi}[v])$, $\mathsf{M}[v]$ = *maxmin*(v).

Next suppose that v is an internal node of $N(Q)$ with k children $u_1, \ldots u_k$ in $N(Q)$ and that the lemma holds for all children of v. We consider two cases based on v being a union

node or an intersection node. Since the next two equations will be used in arguments of both cases, we express them in advance. By Definition 5.4,

$$\mathit{maxmin}(v) = \mathit{min\text{-}wait}(T), \tag{5.1}$$

for a sub-union tree T of $N(Q)[v]$ and

$$\mathit{maxmin}(u_i) = \mathit{min\text{-}wait}(T_i), \tag{5.2}$$

for a sub-union tree T_i of $N(Q)[u_i]$, for every i, $1 \leq i \leq k$.

Consider the case in which v is a union node. Since for every i, $1 \leq i \leq k$, by definition $\top_{\ominus}(T, u_i)$ is a sub-union tree of $N(Q)[u_i]$, by Observation 5.7 and Equation 5.2

$$\mathit{min\text{-}wait}(\top_{\ominus}(T, u_i)) \leq \mathit{maxmin}(u_i) = \mathit{min\text{-}wait}(T_i). \tag{5.3}$$

Also, by Observation 2.13 $T = \top_{\oplus}(\top_{\ominus}(T, u_1), \ldots, \top_{\ominus}(T, u_k))$. Thus, due to Lemma 5.13 and Equation 5.3

$$\mathit{min\text{-}wait}(T) = \min_{1 \leq i \leq k} \mathit{min\text{-}wait}(\top_{\ominus}(T, u_i)) \leq \min_{1 \leq i \leq k} \mathit{min\text{-}wait}(T_i). \tag{5.4}$$

Since T_i is a sub-union tree of $N(Q)[u_i]$, for every i, $1 \leq i \leq k$, by definition $U = \top_{\oplus}(T_1, \ldots, T_k)$ is a sub-union tree of $N(Q)[v]$. According to Lemma 5.13,

$$\mathit{min\text{-}wait}(U) = \min_{1 \leq i \leq k} \mathit{min\text{-}wait}(T_i) \tag{5.5}$$

and by Equation 5.4 $\mathit{min\text{-}wait}(T) \leq \min_{1 \leq i \leq k} \mathit{min\text{-}wait}(T_i)$. Thus,

$$\mathit{min\text{-}wait}(T) \leq \mathit{min\text{-}wait}(U). \tag{5.6}$$

In addition, as U is a sub-union tree of $N(Q)[v]$, by Observation 5.7, $\mathit{maxmin}(v) \geq \mathit{min\text{-}wait}(U)$. Hence, by Equation 5.1

$$\mathit{min\text{-}wait}(T) = \mathit{maxmin}(v) \geq \mathit{min\text{-}wait}(U). \tag{5.7}$$

Now Equations 5.6 and 5.7 yield in the the following equation.

$$\textit{min-wait}(T) = \textit{min-wait}(U). \tag{5.8}$$

Therefore,

$$\begin{aligned}
\textit{maxmin}(v) &= \textit{min-wait}(T) && \text{by Equation 5.1}\\
&= \textit{min-wait}(U) && \text{by Equation 5.8}\\
&= \min_{1\le i\le k} \textit{min-wait}(T_i) && \text{by Equation 5.5}\\
&= \min_{1\le i\le k} \textit{maxmin}(u_i) && \text{by Equation 5.2}\\
&= \min_{1\le i\le k} \mathsf{M}[u_i] && \text{by the induction hypothesis}\\
&= \mathsf{M}[v] && \text{by Definition 5.6, as by}\\
& && \text{assumption } \mathsf{M}[v] \text{ is well-valued.}
\end{aligned}$$

Next we consider the case in which v is an intersection node. For a node u of $N(Q)$, we denote the set of all sub-union trees of $N(Q)[u]$ by $\mathcal{T}(u)$. In addition, we define $\mathcal{S} = \{\top_{\ominus}(T) \mid T \in \mathcal{T}(v)\}$. First we prove $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{T}(u_i)$ and then we use the induction hypothesis to prove the lemma.

Now we first prove that $\mathcal{S} \subseteq \bigcup_{i=1}^{k} \mathcal{T}(u_i)$ and then we will show that $\bigcup_{i=1}^{k} \mathcal{T}(u_i) \subseteq \mathcal{S}$. Given a member T of $\mathcal{S}$, by the definition of $\mathcal{S}$, $T = \top_{\ominus}(U)$, for a member U of $\mathcal{T}(v)$, that is, for a sub-union tree U of $N(Q)[v]$. Hence, as $T = \top_{\ominus}(U)$, by definition of the operator $\top_{\ominus}$, T is a sub-union tree of $N(Q)[u_j]$, for a child u_j of v. Thus, by definition, $T \in \mathcal{T}(u_j) \subseteq \bigcup_{i=1}^{k} \mathcal{T}(u_i)$. So we proved every member of $\mathcal{S}$ is in $\bigcup_{i=1}^{k} \mathcal{T}(u_i)$. Therefore, $\mathcal{S} \subseteq \bigcup_{i=1}^{k} \mathcal{T}(u_i)$.

Now considering a member T of $\mathcal{T}(u_j)$, for a child u_j of v, we prove $T \in \mathcal{S}$ and then we conclude that $\bigcup_{i=1}^{k} \mathcal{T}(u_i) \subseteq \mathcal{S}$. By definition $\top_{\oplus}(T)$ is a sub-union tree of $N(Q)[v]$ and hence $\top_{\oplus}(T) \in \mathcal{T}(v)$. So, as by Observation 2.11 $T = \top_{\ominus}(\top_{\oplus}(T))$, T is in $\mathcal{S}$. Thus, $\mathcal{T}(u_j) \subseteq \mathcal{S}$ where u_j was selected as an arbitrary child of v. As a result, $\mathcal{T}(u_i) \subseteq \mathcal{S}$, for every child u_i of v. Hence, $\bigcup_{i=1}^{k} \mathcal{T}(u_i) \subseteq \mathcal{S}$. Therefore, $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{T}(u_i)$.

Now we can write

$$\begin{array}{lll}
\mathit{maxmin}(v) & = \max_{T\in\mathcal{T}(v)} \mathit{min\text{-}wait}(T) & \text{by Definition 5.4}\\
& = \max_{T\in\mathcal{T}(v)} \mathit{min\text{-}wait}(\top_{\ominus}(T)) & \text{by Lemma 5.12}\\
& = \max_{U\in\mathcal{S}} \mathit{min\text{-}wait}(U) & \text{by Definition of } \mathcal{S}\\
& = \max_{1\le i\le k} \max_{U\in\mathcal{T}(u_i)} \mathit{min\text{-}wait}(U) & \text{as } \mathcal{S}=\bigcup_{i=1}^{k}\mathcal{T}(u_i)\\
& = \max_{1\le i\le k} \mathit{maxmin}(u_i) & \text{by Defintion 5.4}\\
& = \max_{1\le i\le k} \mathsf{M}[u_i] & \text{by the induction hypothesis}\\
& = \mathsf{M}[v] & \text{by Definition 5.6, as by}\\
& & \text{assumption } \mathsf{M}[v] \text{ is well-valued.}
\end{array}$$

So, in both cases $\mathsf{M}[v] = \mathit{maxmin}(v)$. □

Now we discuss how to fill the array W and we define the concept of being well-valued for elements of this array. As explained above, given a node v of Q, we define a variable $\mathsf{W}[v]$ to store an object of value e only if there is an eyewitness for v. Since the definition of an eyewitness also depends on e and the value of e changes frequently, it will be too costly, in terms of time, to update the array W at the same rate that e changes. To solve this problem, we consider an additional variable E which is at most e. Once in a while E is set equal to e and the whole of the array W is filled from scratch so that after this updating for every node v, $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if v has an eyewitness. After that updating, during the execution of the algorithm the variable $\mathsf{W}[v]$ is updated so that as long as e is not changed (and so $\mathsf{E} = \mathsf{e}$), $\mathsf{W}[v]$ is an object of value e if and only if v has an eyewitness. When $\mathsf{E} \neq \mathsf{e}$ the algorithm will not use contents of elements of the array W as it is not guaranteed that $\mathsf{W}[v]$ shows if there is an eyewitness for v, for nodes v of Q.

Definition 5.7 *The variable E is* well-valued *if $\mathsf{E} \le \mathsf{e}$.*

Now assuming $\mathsf{E} = \mathsf{e}$ we explain how the algorithm updates W. Suppose there is an eyewitness T for a node v. If v is a union node, by Lemma 5.10 $\top_{\ominus}(T)$ is an eyewitness for its root, which is a child of v. Also, as we shown in the Lemma 5.11, if v is an intersection node, $\top_{\ominus}(T, u)$ is an eyewitness for u, for every child u of v. So as we will prove more formally, when v is a union node, v has an eyewitness if and only if a child of v has an

eyewitness, and when v is an intersection node, v has an eyewitness if and only if every child of v has an eyewitness. Moreover when v is a leaf, using Definition 5.5, it easily can be shown that v has an eyewitness if and only if $\mu(\mathfrak{fi}[v]) = \mathsf{e}$. The other point is that when $\mathsf{e} \in \{\infty, -\infty\}$, e is not the value of any object and so by Definition 5.5 there is no eyewitness for any node. Hence, as the algorithm uses the array W only to check if there is an eyewitness for the root, when $\mathsf{e} \in \{\infty, -\infty\}$ contents of the array W are not important to the algorithm. Regarding the facts that we stated, we now define the concept of being well-valued for elements of the array W as follows.

Definition 5.8 *Given a node v of Q, $\mathsf{W}[v]$ is* well-valued *if $\mathsf{E} \neq \mathsf{e}$, $\mathsf{e} \in \{-\infty, \infty\}$, or all of the following conditions hold:*

1. $\mu(\mathsf{W}[v]) \leq \mathsf{e}$.
2. *If v is a leaf, $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if $\mu(\mathfrak{fi}[v]) = \mathsf{e}$.*
3. *If v is a union node, $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if $\mu(\mathsf{W}[u]) = \mathsf{e}$, for a child u of v.*
4. *If v is an intersection node, $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if $\mu(\mathsf{W}[u]) = \mathsf{e}$, for every child u of v.*

Now we prove that if the array W is filled such that it satisfies the conditions of the above definition and $\mathsf{E} = \mathsf{e}$ then the algorithm can use elements of W to check the existence of eyewitnesses.

Lemma 5.15 *Suppose $\mathsf{E} = \mathsf{e}$, $\mathsf{e} \notin \{-\infty, \infty\}$, and v is a node of Q such that $\mathsf{W}[u]$ is well-valued, for every node u in $Q[v]$. Then, $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if there is an eyewitness for v.*

Proof. We use induction on the height of v to prove the lemma. Before presenting the proof for the base case and also for the induction step, let us first express a trivial fact that will be used in the proof. Since v is in $Q[v]$, the following fact follows from the assumption of the lemma.

Observation 5.16 *The variable $\mathsf{W}[v]$ is well-valued.*

First we discuss the base case. The base case is when v is a leaf. Then, as $\mathsf{e} = \mathsf{E}$, $\mathsf{e} \notin \{-\infty, \infty\}$, v is a leaf, and by Observation 5.16 $\mathsf{W}[v]$ is well-valued, by Definition 5.8 $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if $\mu(\mathfrak{fi}[v]) = \mathsf{e}$. Moreover, given a sub-intersection tree T of $Q[v]$, by definition v is in T and since v is the only node of $Q[v]$, there is no leaf other than v in T. So, *leaves*$(T) = \{v\}$ and hence by Definition 5.5 T is an eyewitness if $\mu(\mathfrak{fi}[v]) = \mathsf{e}$. Thus, as T was an arbitrary sub-intersection tree for $Q[v]$, v has an eyewitness if and only if $\mu(\mathfrak{fi}[v]) = \mathsf{e}$. Also, we proved that $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if $\mu(\mathfrak{fi}[v]) = \mathsf{e}$. So, when v is a leaf, $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if there is an eyewitness for v.

Now suppose v is an internal node with k children $u_1, \ldots u_k$ and the lemma is true for all children of v. To prove the lemma for v, we consider two cases based on the operator associated with v. Again we express a simple fact before starting the proof. Given a child u of v, since all nodes of $Q[u]$ are in $Q[v]$, it follows from the assumption of the lemma that $\mathsf{W}[w]$ is well-valued, for every node w of $Q[u]$. Also, by assumption $\mathsf{E} = \mathsf{e}$ and $\mathsf{e} \notin \{-\infty, \infty\}$. Hence, as by induction the lemma is true for children of v, the following observation follows.

Observation 5.17 *Given a child u of v, $\mu(\mathsf{W}[u]) = \mathsf{e}$ if and only if there is an eyewitness for u.*

First suppose v is a union node. We first prove the "if" part of the claim and then we show the correctness of the "only if" part. So suppose an eyewitness T exists for v. Consider the root u of $\top_{\ominus}(T)$. $\top_{\ominus}(T)$ is an eyewitness for u by Lemma 5.10. Thus by Observation 5.17 $\mu(\mathsf{W}[u]) = \mathsf{e}$. Hence, considering the facts that by Observation 5.16 $\mathsf{W}[v]$ is well-valued, $\mathsf{e} \notin \{-\infty, \infty\}$, and $\mathsf{E} = \mathsf{e}$, by the second condition of Definition 5.8 we can conclude that $\mu(\mathsf{W}[v]) = \mathsf{e}$. Now we prove the other side of the claim, that is, if $\mu(\mathsf{W}[v]) = \mathsf{e}$ then v has an eyewitness. If $\mu(\mathsf{W}[v]) = \mathsf{e}$, as $\mathsf{E} = \mathsf{e}$ and $\mathsf{e} \notin \{-\infty, \infty\}$, by the third condition of Definition 5.8 $\mu(\mathsf{W}[u]) = \mathsf{e}$, for a child u of v. So, by Observation 5.17 there is an eyewitness U for u and hence as by Observation 2.12 $U = \top_{\ominus}(\top_{\oplus}(U))$, $\top_{\ominus}(\top_{\oplus}(U))$ is an eyewitness for u. So, by Lemma 5.10 $\top_{\oplus}(U)$ is an eyewitness for its root, that is, for v.

Now consider the case in which v is an intersection node. First we suppose that there is an eyewitness for v and we prove that $\mu(\mathsf{W}[v]) = \mathsf{e}$. After that we will prove the "if" part of the lemma. If T is an eyewitness for v, by Lemma 5.11 $\top_{\ominus}(T, u)$ is an eyewitness for u, for every child u of v, and hence by Observation 5.17 $\mu(\mathsf{W}[u]) = \mathsf{e}$, for every child u

of v. Thus, as $\mathsf{E} = \mathsf{e}$, $\mathsf{e} \notin \{-\infty, \infty\}$, and by Observation 5.16 $\mathsf{W}[v]$ is well valued, by the fourth condition of Definition 5.8 $\mu(\mathsf{W}[v]) = \mathsf{e}$. Now we prove the "if" part of the lemma. Suppose $\mu(\mathsf{W}[v]) = \mathsf{e}$. Then, since $\mathsf{E} = \mathsf{e}$ and $\mathsf{e} \notin \{-\infty, \infty\}$, by the fourth condition of Definition 5.8 $\mu(\mathsf{W}[u]) = \mathsf{e}$, for every child u of v. So, by Observation 5.17 every child u_i of v has an eyewitness T_i. Now we prove that $T = \top_{\oplus}(T_1, \ldots, T_k)$ is an eyewitness for v. As by assumption T_i is an eyewitness for u_i and by Observation 2.14 $T_i = \top_{\ominus}(T, u_i)$, $\top_{\ominus}(T, u_i)$ is an eyewitness for u_i, for every child u_i of v. Therefore, by Lemma 5.11 T is an eyewitness for v.

Now we have proved the induction claim in both cases that v is a union node and that v is an intersection node. Consequently, the lemma is true for all nodes of Q. □

Given an intersection node, in order to keep $\mathsf{W}[v]$ well-valued, when $\mathsf{E} = \mathsf{e}$ and $\mathsf{e} \notin \{-\infty, \infty\}$, due to Definition 5.8 we have to keep track of the number of children u of v with $\mu(\mathsf{W}[u]) = \mathsf{e}$ so that as soon as for all children u of v the equation $\mu(\mathsf{W}[u]) = \mathsf{e}$ is satisfied, we can set $\mathsf{W}[v]$ equal to an object of value e. So for every intersection node v we define a variable $\mathsf{Counter}[v]$ storing the number of children u of v with $\mu(\mathsf{W}[u]) = \mathsf{e}$ when $\mathsf{e} = \mathsf{E}$.

Definition 5.9 *Given an intersection node v, $\mathsf{Counter}[v]$ is* well-valued *if it equals the number of children u of v with $\mu(\mathsf{W}[u]) = \mathsf{E}$.*

5.3.1 Updating C

The procedure described in Figure 5.1 is responsible for updating C using variables W[root] and M[root]. The procedure consists of two "if" blocks starting from lines 1 and 4. The first block will use Observation 5.8 and the second block will use Lemma 5.9 to update C. Suppose before starting the procedure its preconditions, listed in Figure 5.1, hold. We first explain what is happening in Block 1 and we prove that after executing this block postconditions of the procedure hold. Then we prove that if Block 1 is executed then the condition checked on line 4 is false and thus Block 4 is not run. In this way we conclude that if Block 4 is executed, its previous block has not been run and so all preconditions of the procedure still hold. Finally, we will show that if Block 4 is executed, after its execution

all postconditions of the procedure hold. Then we will summarize all these results and we will conclude that after exiting the procedure its postconditions hold.

We first discuss the first "if" block. Suppose before executing the procedure its preconditions hold. Since by the second precondition before executing the procedure all elements of the array M are well-valued, by Lemma 5.14 before executing the procedure $\mathsf{M}[\text{root}] = \mathit{maxmin}(\text{root})$. So as by the condition checked on line 1 $\mathsf{M}[\text{root}] > \mathsf{e}$, $\mathit{maxmin}(\text{root}) > \mathsf{e}$. Also, since we proved before executing the procedure $\mathit{maxmin}(\text{root}) = \mathsf{M}[\text{root}]$, by executing the block e is set equal to $\mathit{maxmin}(\text{root})$ (line 2) and if there is no speedy leaf, $\mathsf{e_s}$ is also set equal to $\mathit{maxmin}(\text{root})$ (line 3). Hence, since by precondition 1 Invariants 1 and 2 before executing the procedure held and as we showed $\mathit{maxmin}(\text{root}) > \mathsf{e}$, by Observation 5.8 after executing Block 1 Invariant 1 holds. So the following lemma is true.

Lemma 5.18 *Suppose before executing Update$\mathsf{C}$, its preconditions hold and during the execution of the procedure Block 1 is executed. Then, after executing this block Invariant 1 holds.* □

We now prove the correctness of other postconditions of the lemma after executing Block 1.

Lemma 5.19 *Suppose before executing Update$\mathsf{C}$ its preconditions hold and during its execution Block 1 is executed. Then, after executing Block 1 all postconditions of the procedure hold.*

Proof. We consider postconditions of the lemma one by one. The correctness of the first invariant follows from Lemma 5.18. Now let use prove Invariant 3. We consider two cases based on Q having a speedy leaf or not. If there is no speedy leaf, by its definition, Invariant 3 holds always. So consider the case in which there is a speedy leaf. Then the condition of the "if" command of line 3 is false and hence the only variable changed in the block is e. Therefore, as by its definition Invariant 3 does depend on e and by precondition 1 before executing the procedure Invariant 3 holds, after executing the block still Invariant 3 holds.

Finally we prove that Invariant 2 holds after executing the block. The block is executed only if $\mathsf{e} < \mathsf{M}[\text{root}]$ and the block sets $\mathsf{e} := \mathsf{M}[\text{root}]$. Hence, the block increases e. Also, as

Figure 5.1: Updating the eliminating configuration

Preconditions:

1. Invariants 1, 2, and 3 hold.
2. All variables are well-valued.
3. If $\mu(\mathsf{W}[\text{root}]) = \mathsf{E} = \mathsf{e}$, the speedy leaf (if it exists) is satiated.

Changing variables: C and $\mathfrak{fi}[sl(Q)]$.

function UpdateC **returns:** boolean;
begin

```
1   if M[root] > e then
2       e := M[root];                                              } Block 1
3       if there is no speedy leaf then e_s := M[root];
4   if μ(W[root]) = E = e and e ∉ {−∞, ∞} then                     } Block 4
5       e_s := e+;                                                 } Block 2
6       e := e+;
7       Φ := Φ ⊙ (l, i, i) where W[root] is the ith object of ω(l);
8       if there is a speedy leaf and μ(fi[sl(Q)]) = ⟨e⟩ then
9           if fi[sl(Q)] is the last object in ω(sl(Q)) then       } Block 3
                fi[sl(Q)] := END;
            else
10              fi[sl(Q)] := The object after fi[sl(Q)];
```

end

Postconditions: Invariants 1, 2, and 3 hold.

the block does not change the value of elements of the array fi and the definition of being a scanned object (Definition 5.1) depends only on the array fi, by executing the block the set of scanned objects does not change. Furthermore, before executing the procedure by precondition 1 Invariant 2 held and so the value of every scanned object was less than e. Thus, as e has been increased in the block, after executing the block, the value of every scanned object is still less than e and thus Invariant 2 holds. □

Now we prove that at most one of Blocks 1 and 4 is executed. Recall the usage of time stamps form the very beginning of the chapter.

Lemma 5.20 *Suppose before executing Update$\mathsf{C}$, its preconditions hold. Then, if Block 1 is executed, Block 4 is not executed.*

Proof. Defining *bef* and *aft* as times before and after executing Block 1, we prove $\mathsf{E}^{(aft)} < \mathsf{e}^{(aft)}$ and then we conclude that the condition in line 4 does not allow Block 4 to run. Since Block 1 does not modify M or E we do not use time stamps for these variables. Before executing the procedure, by the second precondition E is well-valued and so by Definition 5.7, $\mathsf{E} \leq \mathsf{e}^{(bef)}$. Also as Block 1 is executed, by the condition checked on line 1 $\mathsf{M}[\text{root}] > \mathsf{e}^{(bef)}$ and thus $\mathsf{E} < \mathsf{M}[\text{root}]$. Moreover, due to line 2, $\mathsf{e}^{(aft)} = \mathsf{M}[\text{root}]$. Hence, $\mathsf{E} < \mathsf{e}^{(aft)}$. So, the condition checked on line 4 is false and thus Block 4 is not executed. □

A consequence of the above lemma is that if before executing Update$\mathsf{C}$, its preconditions hold and Block 4 is executed, before executing Block 4 the procedure does not run any command and so when the procedure starts running Block 4 still all preconditions of the procedure hold.

Observation 5.21 *Suppose before executing Update$\mathsf{C}$ its preconditions hold. If Block 4 is executed, then just before running Block 4, preconditions of the procedure hold.*

Next we consider the block starting at line 4. This block will try to increase e using the approach justified in Lemma 5.9. In the next two lemmas we show if the conditions checked on line 4 let this block run, certain conditions that allow the algorithm to use the approach described in Lemma 5.9 for increasing e efficiently hold. As is clear from

Lemma 5.9, to use this approach the algorithm needs to evaluate $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e})$. The next lemma shows how the algorithm can compute this expression.

Lemma 5.22 *Suppose before calling Update$\mathsf{C}$, its preconditions holds. Then if Block 4 is executed, just before executing this block the equation $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) = \Phi$ holds.*

Proof. If the query tree has no speedy leaf, by Definition 3.9 the lemma is true. So we suppose there is a speedy leaf. By Definition 3.9, in order to prove $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) = \Phi$, it suffices to show that $pos(I, sl(Q), \mathsf{e_s}) = pos(I, sl(Q), \mathsf{e})$. We now prove this fact. Since Block 4 is executed and before executing the procedure its preconditions hold, by Observation 5.21 just before executing Block 4 preconditions of the procedure still hold. Also, the condition checked on line 4 is true and so $\mu(\mathsf{W}[\text{root}]) = \mathsf{E} = \mathsf{e}$. Hence by the third precondition $sl(Q)$ is satiated. Thus, since by precondition 1 Invariant 2 holds, by Lemma 5.3, $find(I, sl(Q), \mathsf{e}) = \mathfrak{fi}[sl(Q)]$. In addition, by precondition 1 Invariant 3 holds and by Invariant 3 $find(I, sl(Q), \mathsf{e_s}) = \mathfrak{fi}[sl(Q)]$. So, $find(I, sl(Q), \mathsf{e_s}) = \mathfrak{fi}[sl(Q)] = find(I, sl(Q), \mathsf{e})$. Therefore, by Lemma 3.4 $pos(I, sl(Q), \mathsf{e_s}) = pos(I, sl(Q), \mathsf{e})$. Hence, by Definition 3.9 $\Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) = \Phi$. □

In order to apply the method described in Lemma 5.9, it is needed that there is an eyewitness for the root, as explained in Lemma 5.9. We now prove that if before executing Update$\mathsf{C}$, its the preconditions hold and during the execution of Update$\mathsf{C}$ Block 4 is executed, an eyewitness for the root exists before executing Block 4. So, consider the time just before executing Block 4. At this time by Observation 5.21 the preconditions of the procedure hold. Also, Block 4 is run only if $\mathsf{E} = \mathsf{e}$ and $\mathsf{e} \notin \{-\infty, \infty\}$. Hence as by precondition 2 elements of the array W are well-valued, by Lemma 5.15 $\mu(\mathsf{W}[\text{root}]) = \mathsf{e}$ if and only if there is an eyewitness for the root. Also the block is executed only if $\mu(\mathsf{W}[\text{root}]) = \mathsf{e}$. Therefore, there is an eyewitness for the root before executing Block 4. Thus, the next lemma follows.

Lemma 5.23 *If before executing Update$\mathsf{C}$, its the preconditions hold and during the execution of Update$\mathsf{C}$ Block 4 is executed, before executing the block there is an eyewitness for the root.* □

We next talk about Block 4 of the procedure. We first discuss Block 2 and then we will explain the role of the "if" block starting at line 8. Block 2 adds an object of value e to the expansion of Φ and then increases e as explained in Lemma 5.9. As is clear from lines 5, 6, and 7 of Block 2, in Block 2 the algorithm is setting $\mathsf{C} := (\mathsf{e}_+, \mathsf{e}_+, \Phi \odot (l, i, i))$ where $\omega(l)[i] = \mathsf{W}[\text{root}]$. Since by the condition checked on line 4 $\mu(\mathsf{W}[\text{root}]) = \mathsf{e}$ and as mentioned $\omega(l)[i] = \mathsf{W}[\text{root}]$, $\mu(\omega(l)[i]) = \mathsf{e}$. Suppose before running the procedure, its preconditions hold. Then, as we proved in Lemma 5.22, $\Phi = \Phi_{sl}(\mathsf{C}, \mu, \mathsf{e})$. So, Block 2 of the algorithm sets $\mathsf{C} := (\mathsf{e}_+, \mathsf{e}_+, \Phi_{sl}(\mathsf{C}, \mu, \mathsf{e}) \odot (l, i, i))$ where $\mu(\omega(l)[i]) = \mathsf{e}$. Also, according to Observation 5.21, when executing Block 4 the preconditions still hold and hence Invariant 1 holds. Furthermore, by Lemma 5.23 before executing Block 4 there is an eyewitness for the root. So, by Lemma 5.9 after this setting C remains valid. Also, as it is clear from the figure, Block 3 does not modify C. So, after executing Block 4 C is valid and hence Invariant 1 is satisfied.

Lemma 5.24 *If before executing Update$\mathsf{C}$, its the preconditions hold and and during the execution of Update$\mathsf{C}$ Block 4 is executed, after executing this block Invariant 1 holds.* □

Before discussing the "if" block of line 8 we note that, as can be seen, if $\mathrm{fi}[sl(Q)]$ is not the last object, this "if" block sets $\mathrm{fi}[sl(Q)]$ to an object of $sl(Q)$; otherwise the "if" block sets $\mathrm{fi}[sl(Q)] :=$ END. So, in either case the value assigned to $\mathrm{fi}[sl(Q)]$ satisfies Definition 5.1.

Next we show that the "if" block of line 8 guarantees the correctness of Invariant 3. To see how this block works, suppose before executing the procedure its preconditions hold and during the execution of the procedure Block 4 is run. Hence, defining *bef* and *aft* as times just before and just after executing Block 4, we may conclude from Observation 5.21 that at the time *bef*, the preconditions of the procedure still hold. Furthermore, as Block 4 is executed, by the condition checked on line 4, $\mu(\mathsf{W}^{(bef)}[\text{root}]) = \mathsf{E} = \mathsf{e}$. So, due to the fact that the preconditions hold at the time *bef*, at the time *bef* the leaf $sl(Q)$ is satiated, according to the third precondition. Thus by Lemma 5.3 $\mathrm{fi}^{(bef)}[sl(Q)] = \mathit{find}(I, sl(Q), \mathsf{e}^{(bef)})$. Now recall from the definition of Invariant 3 that, to satisfy this invariant at the time *aft*, $\mathrm{fi}[sl(Q)]$ should be set equal to $\mathit{find}(I, sl(Q), \mathsf{e_s}^{(aft)})$. Hence as due to line 5, $\mathsf{e_s}^{(aft)} = \mathsf{e}^{(bef)}{}_+$, $\mathrm{fi}[sl(Q)]$ should be set equal to $\mathit{find}(I, sl(Q), \mathsf{e}^{(bef)}{}_+)$ while as we proved $\mathrm{fi}^{(bef)}[sl(Q)] = \mathit{find}(I, sl(Q), \mathsf{e}^{(bef)})$. In the next two lemmas we explain how the algorithm can use the

result of the expression $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}^{(\mathit{bef})})$ to evaluate the expression $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}^{(\mathit{bef})}+)$. We first consider the situation in which $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}^{(\mathit{bef})})$ does not have value $\mathsf{e}^{(\mathit{bef})}$ and then we consider the other cases.

Lemma 5.25 *Given a value b in $\mathbb{V}$, $b < \infty$, and a leaf l, if $\mu(\mathit{find}(I, l, b)) \neq b$ then $\mathit{find}(I, l, b+) = \mathit{find}(I, l, b)$.*

Proof. We consider two cases $\mathit{find}(I, l, b) = \text{END}$ and $\mathit{find}(I, l, b) \neq \text{END}$. First suppose $\mathit{find}(I, l, b) = \text{END}$. Then, there is no object of value at least b since otherwise by Lemma 3.1 $\mathit{find}(I, l, b)$ is an object while END is not an object. Hence, as $b < b+$, values of all objects of l are less than $b+$. So by Lemma 3.1 $\mathit{find}(I, l, b+) = \text{END} = \mathit{find}(I, l, b)$.

Now consider the case $\mathit{find}(I, l, b) \neq \text{END}$. We first show that $\mu(\mathit{find}(I, l, b)) > b+$ and then we prove values of objects with values less than $\mu(\mathit{find}(I, l, b))$ are less than $b+$. These results prove the lemma, as we will show. As $\mathit{find}(I, l, b) \neq \text{END}$, by Lemma 3.1 there is at least one object of value at least b and hence by the same lemma $\mathit{find}(I, l, b)$ is the smallest object of value at least b. As a result, $\mu(\mathit{find}(I, l, b)) \geq b$. So, as by assumption $\mu(\mathit{find}(I, l, b)) \neq b$, $\mu(\mathit{find}(I, l, b)) > b$ and thus by Observation 2.29 $\mu(\mathit{find}(I, l, b)) > b+$. Also values of all objects of l smaller than the object $\mathit{find}(I, l, b)$ are less than b as $\mathit{find}(I, l, b)$ is the smallest object of value at least b. Thus, due to the inequality $b < b+$, values of objects of l that are before $\mathit{find}(I, l, b)$ in $\omega(l)$ are less than $b+$. Hence, as we proved earlier that $\mu(\mathit{find}(I, l, b)) > b+$, $\mathit{find}(I, l, b)$ is the smallest object of value at least $b+$ and thus by Lemma 3.1 $\mathit{find}(I, l, b+) = \mathit{find}(I, l, b)$. Therefore, the lemma is true in both cases. □

Lemma 5.26 *Consider a value b in $\mathbb{V}$, $b < \infty$, a leaf l of Q, and suppose $\mu(\mathit{find}(I, l, b)) = b$. Then $\mathit{find}(I, l, b)$ is an object o of l and $\mathit{find}(I, l, b+)$ can be evaluated as follows.*

$$\mathit{find}(I, l, b+) = \begin{cases} \text{END} & \textit{if } o \textit{ is the last object of } \omega(l) \\ p & \textit{if } o \textit{ is not the last object of } \omega(l) \textit{ where } p \textit{ is the object following } o \textit{ in } \omega(l). \end{cases}$$

Proof. We first prove that $\mathit{find}(I, l, b)$ is an object and then we show that $\mathit{find}(I, l, b+)$ can be computed in the way that is claimed by the lemma. By Lemma 3.1 $\mathit{find}(I, l, b)$ is either

END or an object of l. Therefore, since by assumption $\mu(find(I, l, b)) = b < \infty = \mu(\text{END})$, $find(I, l, b) \neq \text{END}$ and hence, $find(I, l, b)$ is an object of l.

To prove the second claim, we consider two cases based on o being the last object in $\omega(l)$ or not. First suppose o is the last object. Since by assumption $\mu(o) = \mu(find(I, l, b)) = b < b+$, the value of o and values of all objects of l smaller than o are less than $b+$. Therefore, as by assumption o is the last object, there is no object of value at least $b+$ in $\omega(l)$ and thus by Lemma 3.1 $find(I, l, b+) = \text{END}$. So the claim of the lemma is true in this case.

Now consider the case in which o is not the last object. Then, as μ is ordered, the value of the object p following o is more than $\mu(o) = b$ and hence its value is more than $b+$ by Observation 2.29. Also, since o is the biggest object of l smaller than p, the value of every object that is before p in $\omega(l)$ is at most $\mu(o) = b < b+$. So p is the smallest object of l with a value at least $b+$ and thus $find(I, l, b+) = p$, according to Lemma 3.1. □

Now we use Lemmas 5.25 and 5.26 to show that after executing the "if" block of line 8, Invariant 3 holds. Since E is not modified in the procedure, we will not use any time stamp for this variable.

Lemma 5.27 *Suppose before executing Update$\mathsf{C}$, its preconditions hold. If during execution of Update$\mathsf{C}$ Block 4 is executed, then after executing this block Invariant 3 holds.*

Proof. Defining *bef*, t, and *aft* as times just before executing the procedure, the time just before executing the "if" block of line 8, and the time just after executing the procedure, respectively, we prove that the "if" block starting at line 8 evaluates $find(I, sl(Q), \mathsf{e_s}^{(aft)})$ in the way that is described in Lemmas 5.25 and 5.26 and assigns it to $\mathrm{fi}[sl(Q)]$. For this purpose we first extract relations between values of variables $\mathrm{fi}[sl(Q)]$, e, and $\mathsf{e_s}$ at times *bef*, t, and *aft* and then we consider different cases based on the conditions checked on lines 8 and 9.

We now explain relations between values of variables at different times, as mentioned. Since by assumption Block 4 is executed, by Lemma 5.20 Block 1 is not executed and so values of variables when executing line 4 is the same as that values at the time *bef*. Due to

lines 5 and 6 and the fact that $\mathsf{e_s}$ is not modified in Block 3, the following equations hold.

$$\mathsf{e_s}^{(aft)} = \mathsf{e}^{(bef)}{}_+ \tag{5.9}$$

$$\mathsf{e}^{(t)} = \mathsf{e}^{(bef)}{}_+ \tag{5.10}$$

Since Block 4 is executed, when executing line 4 the condition $\mu(\mathsf{W}[\text{root}]) = \mathsf{E} = \mathsf{e}$ of this line was true. Hence, $\mu(\mathsf{W}^{(bef)}[\text{root}]) = \mathsf{E} = \mathsf{e}^{(bef)}$ and so, due to the third precondition, the speedy leaf is satiated at the time *bef*. Hence, by Lemma 5.3 $\mathfrak{fi}^{(bef)}[sl(Q)] = find(I, sl(Q), \mathsf{e}^{(bef)})$. Also, since $\mathfrak{fi}[sl(Q)]$ is not modified in the procedure before Block 3, $\mathfrak{fi}^{(t)}[sl(Q)] = \mathfrak{fi}^{(bef)}[sl(Q)]$. Hence,

$$\mathfrak{fi}^{(t)}[sl(Q)] = find(I, sl(Q), \mathsf{e}^{(bef)}). \tag{5.11}$$

Now we consider three cases based on the conditions checked on lines 8 and 9 being true or false. First suppose the condition checked on line 8 is false. Thus, either there is no speedy leaf or $\mu(\mathfrak{fi}^{(t)}[sl(Q)]) \neq \langle \mathsf{e}^{(t)} \rangle$. If there is no speedy leaf, Invariant 3 holds always by its definition and so the proof of the lemma is trivial. So, suppose there is a speedy leaf and hence $\mu(\mathfrak{fi}^{(t)}[sl(Q)]) \neq \langle \mathsf{e}^{(t)} \rangle$. Consequently, by Equations 5.11 and 5.10 $\mu(find(I, sl(Q), \mathsf{e}^{(bef)})) \neq \langle \mathsf{e}^{(bef)}{}_+ \rangle = \mathsf{e}^{(bef)}$. Therefore, by Lemma 5.25 $find(I, sl(Q), \mathsf{e}^{(bef)}{}_+) = find(I, sl(Q), \mathsf{e}^{(bef)})$. Hence, due to Equations 5.9 and 5.11,

$$find(I, sl(Q), \mathsf{e_s}^{(aft)}) = find(I, sl(Q), \mathsf{e}^{(bef)}{}_+) = find(I, sl(Q), \mathsf{e}^{(bef)}) = \mathfrak{fi}^{(t)}[sl(Q)].$$

Now, since by assumption the condition checked on line 8 is false, Block 3 is not executed and thus $\mathfrak{fi}^{(aft)}[sl(Q)] = \mathfrak{fi}^{(t)}[sl(Q)]$. Consequently, $find(I, sl(Q), \mathsf{e_s}^{(aft)}) = \mathfrak{fi}^{(aft)}[sl(Q)]$ and so at the time *aft* Invariant 3 is met.

Next we consider the other case, in which the condition checked on line 8 is true. Hence, there is a speedy leaf and $\mu(\mathfrak{fi}^{(t)}[sl(Q)]) = \langle \mathsf{e}^{(t)} \rangle$. Thus, by Equations 5.11 and 5.10, $\mu(find(I, sl(Q), \mathsf{e}^{(bef)})) = \langle \mathsf{e}^{(bef)}{}_+ \rangle = \mathsf{e}^{(bef)}$. Therefore, as by the condition checked on line 4 $\mathsf{e}^{(bef)} \neq \infty$, by Lemma 5.26, depending on $find(I, sl(Q), \mathsf{e}^{(bef)})$ being the last object or not, $find(I, sl(Q), \mathsf{e}^{(bef)}{}_+)$ equals END or the object following the object $find(I, sl(Q), \mathsf{e}^{(bef)})$. Thus, considering Equation 5.11, we may conclude that depending on $\mathfrak{fi}^{(t)}[sl(Q)]$ being the

last object or not, $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}^{(bef)}+)$ equals END or equals the object following $\mathfrak{fi}^{(t)}[\mathit{sl}(Q)]$. Moreover, since by assumption the condition checked on line 8 is true, Block 3 is executed and so, as can be seen in Figure 5.1, depending on $\mathfrak{fi}^{(t)}[\mathit{sl}(Q)]$ being the last object or not, $\mathfrak{fi}[\mathit{sl}(Q)]$ is set equal to END or equal to the object following $\mathfrak{fi}^{(t)}[\mathit{sl}(Q)]$. As a result, Block 3 sets $\mathfrak{fi}[\mathit{sl}(Q)]$ equal to $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e}^{(bef)}+) = \mathit{find}(I, \mathit{sl}(Q), \mathsf{e_s}^{(aft)})$ (Equation 5.9) and hence $\mathfrak{fi}^{(aft)}[\mathit{sl}(Q)] = \mathit{find}(I, \mathit{sl}(Q), \mathsf{e_s}^{(aft)})$. Therefore, at the time *aft* Invariant 3 holds in this case as well. □

We now prove the correctness of the next postcondition of Update$\mathsf{C}$, the second invariant, after a legal call to this procedure during which Block 4 is executed.

Lemma 5.28 *If before executing Update$\mathsf{C}$ the preconditions hold and during the execution of the procedure Block 4 is executed, after executing this block Invariant 2 holds.*

Proof. Defining *bef* and *aft* as the time just before executing the procedure and the time just after executing the procedure, respectively, to prove the lemma we first describe the relation between values of variables e and $\mathfrak{fi}$ at times *aft* and *bef*. Then, using these relations, we prove the correctness of Invariant 2 for *aft*-scanned objects of non-speedy leaves and *aft*-scanned objects of the speedy leaf, when it exists, separately. Since by assumption Block 4 is executed, by Lemma 5.20 Block 1 is not executed. Thus the only changes to variables e and $\mathsf{e_s}$ are in lines 5 and 6. So, due to lines 5 and 6,

$$\mathsf{e}^{(aft)} = \mathsf{e_s}^{(aft)} = \mathsf{e}^{(bef)}+ > \mathsf{e}^{(bef)}. \tag{5.12}$$

We now consider a non-speedy leaf l and we show that values of *aft*-scanned objects of l are less than $\mathsf{e}^{(aft)}$. Elements of the array $\mathfrak{fi}$ may change only in Block 3. Also, due to the condition checked on line 8, Block 3 is executed only if there is a speedy leaf and this block only modifies $\mathfrak{fi}[\mathit{sl}(Q)]$. Therefore, as by assumption l is not a speedy leaf, $\mathfrak{fi}[l]$ is not modified by the procedure and hence $\mathfrak{fi}^{(aft)}[l] = \mathfrak{fi}^{(bef)}[l]$. Now consider an *aft*-scanned object o of l. Then, by Definition 5.1 $\mu(o) < \mu(\mathfrak{fi}^{(aft)}[l]) = \mu(\mathfrak{fi}^{(bef)}[l])$ and thus by the same definition o is also *bef*-scanned. As a result, since by precondition 1 at the time *bef* Invariant 2 holds, $\mu(o) < \mathsf{e}^{(bef)}$. So, as by Equation 5.12 $\mathsf{e}^{(bef)} < \mathsf{e}^{(aft)}$, $\mu(o) < \mathsf{e}^{(aft)}$. Consequently, values of all *aft*-scanned objects of non-speedy leaves are less than $\mathsf{e}^{(aft)}$.

We now prove that the value of every *aft*-scanned object of $sl(Q)$, when a speedy leaf exists, is less than e. Since by assumption Block 4 is executed and also before executing Update$\mathsf{C}$ the preconditions of the procedure hold, by Lemma 5.27 after exiting the procedure Invariant 3 holds. So, $find(I, sl(Q), \mathsf{e_s}^{(aft)}) = \mathfrak{fi}^{(aft)}[sl(Q)]$. Therefore, as by Equation 5.12 $\mathsf{e}^{(aft)} = \mathsf{e_s}^{(aft)}$, $find(I, sl(Q), \mathsf{e}^{(aft)}) = \mathfrak{fi}^{(aft)}[sl(Q)]$. We now consider two cases depending on whether there is an object with a value at least $\mathsf{e}^{(aft)}$ in $\omega(sl(Q))$. If there is no such an object, values of all objects of $sl(Q)$ are less than $\mathsf{e}^{(aft)}$ and hence the value of every *aft*-scanned object of $sl(Q)$ is less than $\mathsf{e}^{(aft)}$. Now consider the case in which there is an object of value at least $\mathsf{e}^{(aft)}$ in $\omega(sl(Q))$. Then, by Definition 3.1 $find(I, sl(Q), \mathsf{e}^{(aft)})$ is the smallest object of value at least $\mathsf{e}^{(aft)}$. Hence as we proved $find(I, sl(Q), \mathsf{e}^{(aft)}) = \mathfrak{fi}^{(aft)}[sl(Q)]$, the value of every object of l that is before $\mathfrak{fi}^{(aft)}[sl(Q)]$ in $\omega(l)$ is less than $\mathsf{e}^{(aft)}$. Moreover, for every *aft*-scanned object o of the speedy leaf, as $\mu(o) < \mu(\mathfrak{fi}^{(aft)}[sl(Q)])$ (Definition 5.1), o is before $\mathfrak{fi}^{(aft)}[sl(Q)]$ in $\omega(l)$. Consequently, the value of every *aft*-scanned object is less than $\mathsf{e}^{(aft)}$. Therefore in either case the value of every *aft*-scanned object is less than $\mathsf{e}^{(aft)}$.

Now that we have proved values of all *aft*-scanned objects are less than $\mathsf{e}^{(aft)}$, by definition we can conclude that after exiting the procedure Invariant 2 holds. □

We now can prove that if before calling Update$\mathsf{C}$ the preconditions of the procedure hold, after executing the procedure, its postconditions hold. We have proved in Lemma 5.20 that if before calling Update$\mathsf{C}$ the preconditions of the procedure hold, at most one of Blocks 1 and 4 is executed. So let us consider the three possible cases based on each of these two blocks being run or not. In Lemma 5.19 we proved if Block 1 is executed, after executing this block all postconditions hold. In this situation as Block 4 is not executed, after executing Block 1 the procedure immediately exits and so the postconditions of the procedure remain satisfied. Also, if Block 4 is executed, after exiting the procedure, by Lemmas 5.24, 5.27, and 5.28, Invariants 1, 3, and 2 hold and so all postconditions are met in this case also. The last possible case is one in which neither of the two Blocks 1 and 4 is executed and hence the procedure does not modify the value of any variable. Then, as before executing the procedure by precondition 1 Invariants 1, 2, and 3 were satisfied, after executing the procedure these invariants still are true and so the postconditions are met. Furthermore as is clear from Figure 5.1, the procedure does not directly modify any

variable other than its changing variables and also there is no procedure call during the execution of the procedure. Hence, during the execution of the procedure no variable other than changing variables of the procedure may be modified.

Lemma 5.29 *Suppose before executing Update$\mathsf{C}$, the preconditions hold. Then after exiting the procedure, its postconditions hold. Also, during the execution of Update$\mathsf{C}$ only its changing variables are modified.* □

5.4 Visiting Functions

Recall from Section 5.1 that there are two visit procedures called by the algorithm repeatedly. In this part we explain what happens when a node v of the query tree is visited, that is, when one of the visit procedures with v as parameter is called. We first investigate visits to leaves and then we discuss visits to internal nodes.

5.4.1 Visiting Leaves

The goal of visiting a leaf l is to update $\mathfrak{fi}[l]$ so that l becomes satiated. To do this, the algorithm must change $\mathfrak{fi}[l]$ such that as many waiting objects of values less than e as possible become scanned. For this purpose, the algorithm uses *gallop search* [DLOM00] to find the biggest object o_l with a value less than e in the sequence of waiting objects of l. Gallop search works in two phases. In the first phase, in n steps, the smallest n such that o_l is among the 2^n smallest waiting objects of l is found. There is a variable $\mathsf{Step}[l]$ which represents the current value of 2^n and is initially 1. In each step of the first phase, the value of $\mathsf{Step}[l]$ is doubled. The first phase finishes when l has less than $\mathsf{Step}[l]$ waiting objects or the $\mathsf{Step}[l]$th waiting object of l has a value at least e. In the second phase, a binary search computes the exact position of o_l, using another $O(n)$ comparisons. So the algorithm works in time $O(\log a)$ where a is the number of waiting objects of l smaller than o_l. When the Gallop(l), for a leaf l, start, it checks values of all objects of l are less than e, instead of doubling $\mathsf{Step}[l]$, it sets $\mathsf{Step}[l]$ equal to the number of waiting objects. In this way, all objects become scanned in the same visit, as we will see in the next chapter.

Figure 5.2: The gallop search algorithm

Preconditions: Invariants 1, 2, and 3 hold.

Changing Variables: $\mathfrak{fi}[l]$, $\mathsf{W}[l]$, $\mathsf{M}[l]$, $\mathsf{Step}[l]$, $\mathsf{e_s}$, and Φ.

Procedure Gallop(leaf l);
begin

11 $i := index(I, l, \mathfrak{fi}[l])$;
12 **if** $i \leq \text{length}(\omega(l))$ *and* $\mu(\omega(l)[\text{length}(\omega(l))]) < e$ **then**
$\mathsf{Step}[l] := \text{length}(\omega(l)) - i + 1$;
else
$\mathsf{Step}[l] := \mathsf{Step}[l] \times 2$;
13 $j := i + \mathsf{Step}[l] - 1$;
14 **if** $j > length(\omega(l))$ *or* $\mu(\omega(l)[j]) \geq e$ **then**
15 **if** $\mu(\mathfrak{fi}[l]) \geq e$ **then**
16 $j := i - 1$;
else
17 Use Binary search to determine the biggest k such that $i \leq k \leq \min(j, \text{length}(\omega(l)))$ and $\mu(\omega(l)[k]) < \mathsf{e}$ and set $j := k$;
18 $\mathsf{Step}[l] := 1$;
} Block 5
19 **if** $j \geq i$ **then**
20 $\mathfrak{fi}[l] := \omega(l)[j+1]$;
21 **if** l *is in* $N(Q)$ **then** $\mathsf{M}[l] := \mu(\mathfrak{fi}[l])$;
22 **if** $\mu(\mathfrak{fi}[l]) = e$ **then** $\mathsf{W}[l] := \mathfrak{fi}[l]$; **else** $\mathsf{W}[l] :=$ BEG;
23 **if** l *is a speedy leaf* **then**
24 $\Phi := \Phi \odot (sl(Q), i, j)$;
25 $\mathsf{e_s} := \mu(\omega(l)[j])+$;
} Block 6
} Block 7
end

Postconditions:
1. Invariants 1, 2, and 3 hold.
2. $\mathsf{W}[l]$ and $\mathsf{M}[l]$are well-valued.

Every time that the algorithm visits l, it performs one step of the first phase of the gallop search. When the first phase is completed, the whole of the second phase is executed in the same visit. In every visit, the algorithm considers the biggest object o found to have a value less than e and sets $\mathfrak{fi}[l]$ to be the object following o so that after this setting, all objects that are known to have values less than e have become scanned.

Figure 5.2 shows the procedure. The procedure has two local variables i and j which are defined to store indices of the first and the last waiting objects that are going to become scanned in this visiting. When there is no waiting object of value at least e and hence no waiting object is going to become scanned, j will be set equal to $i-1$; otherwise the value of j will be at least i and the first $j-i+1$ waiting objects will become scanned. Since the first waiting object by Lemma 5.1 is $\mathfrak{fi}[l]$, the variable i is assigned the index of $\mathfrak{fi}[l]$ in $\omega(l)$ (line 11). Lines 12 and 13 perform a step of the first phase by doubling the integer stored in $\mathsf{Step}[l]$ and then assigning the index of the $\mathsf{Step}[l]$th waiting object, that is $i+\mathsf{Step}[l]-1$, to j.

The "if" block of line 14 is responsible for doing the second phase, if the first phase is over. The algorithm in line 14 checks if the first phase is completed by checking if the value assigned to j is the index of a waiting object with a value less than e. If the result of this checking is negative, then the first phase is completed since there are fewer than $\mathsf{Step}[l]$ waiting objects with values less than e in $\omega(l)$. In such situations, the algorithm, then, accomplishes the second phase (the binary search) in Block 5. First, in line 15 it is checked that if l is already satiated by checking if $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}$ (Lemma 5.4). If l is satiated, by definition l does not have any waiting object of value at least e and so j is assigned the integer $i-1$; otherwise in line 17, using a binary search in the sequence of waiting objects of l, the index of the biggest waiting object with a value less than e in $\omega(l)$ is found and is assigned to j. Then, the procedure resets $\mathsf{Step}[l]$ to 1 to restart the first phase the next time that Gallop(l) is called (line 18).

Now let us first explain a property of the variable $\mathsf{Step}[l]$ and then we discuss the binary search further. There will be an initialization procedure which initialize $\mathsf{Step}[l]$ to 1. After that, $\mathsf{Step}[l]$ only is modified by Gallop(l) and as we now explain Gallop(l) keeps $\mathsf{Step}[l]$ positive: if the condition checked on line 12 is true the "if" block of line 12 sets $\mathsf{Step}[l] :=$ length$(\omega(l))-i+1$ while due to the condition checked on line 12 length$(\omega(l))-i+1 \geq 1$;

otherwise the "if" block of line 12 doubles $\mathsf{Step}[l]$. Also, if line 18 is executed, it sets $\mathsf{Step}[l] := 1$. Hence, every line that modifies $\mathsf{Step}[l]$ keeps $\mathsf{Step}[l]$ greater than zero.

Observation 5.30 *At any time after initializing $\mathsf{Step}[l]$ to 1, $\mathsf{Step}[l] \geq 1$.*

We now explain the binary search in more detail. As mentioned, the binary search of line 17 assigns to j the index k of the biggest waiting object with a value less than e. So, as k is going to be the index of a waiting object, it should be at least i as i is the index of $\mathfrak{fi}[l]$ and by Lemma 5.1 $\mathfrak{fi}[l]$ is the first waiting object of l. Also, line 17 belongs to Block 5 and this block is executed if the first phase is completed, that is, if j is more than the index of the last waiting object with a value less than e. As a result, the index k will be at most j because, as mentioned, k is going to be the index of a waiting object with a value less than e. Hence, as can be seen in the figure, the binary search looks for the biggest k satisfying $\omega(l)[k] < \mathsf{e}$ in the interval $[i, \min(j, \text{length}(\omega(l)))]$. In the next lemma we prove that such an index k exists in this range. Then in Lemma 5.32 we will prove that when executing the binary search, $i \geq 1$ and hence the index k will be in the range $[1, \text{length}(\omega(l))]$. This will yield the fact that k will be the index of an object of l.

Lemma 5.31 *Consider an execution of Gallop(l), for a leaf l, during which the procedure executes line 17. Then, just before executing line 17 there is an integer k satisfying $i \leq k \leq \min(j, \text{length}(\omega(l)))$ and $\mu(\omega(l)[k]) < \mathsf{e}$.*

Proof. We prove that $k = i$ satisfies both inequalities stated in the lemma, first proving the second inequality. Due to line 11, $i = \mathit{index}(I, l, \mathfrak{fi}[l])$ and so by Definition 2.8 $\omega(l)[i] = \mathfrak{fi}[l]$. Also, since line 17 is executed, the condition of line 15 is false and hence $\mu(\mathfrak{fi}[l]) < \mathsf{e}$. So, $\mu(\omega(l)[i]) < \mathsf{e}$ and thus the second inequality holds for $k = i$.

Now we prove the first inequality for $k = i$. The inequality $i \leq k$ follows by equation $k = i$. We now prove the inequality $k \leq \min(j, \text{length}(\omega(l)))$ for $k = i$ by showing the correctness of inequalities $i \leq \text{length}(\omega(l))$ and $i \leq j$, in order. Since, as we proved, $\mu(\mathfrak{fi}[l]) < \mathsf{e}$, $\mu(\mathfrak{fi}[l]) \neq \infty$ and hence $\mathfrak{fi}[l] \neq \text{END}$. So, by Definition 5.1 $\mathfrak{fi}[l]$ is an object of l and thus $\mathfrak{fi}[l] = \omega(l)[m]$, for some m, $m \leq \text{length}(\omega(l))$. Therefore, since we showed $\mathfrak{fi}[l] = \omega(l)[i]$, $i = m$ and hence $i \leq \text{length}(\omega(l))$. Also, as by Observation 5.30 always $\mathsf{Step}[l] \geq 1$, after executing line 13, $j = i + \mathsf{Step}[l] - 1 \geq i$. Therefore, just before running

line 17 $i \leq j$. Hence, as we proved above that when executing line 17 $i \leq \text{length}(\omega(l))$, we may conclude that the inequality $i \leq \min(j, \text{length}(\omega(l)))$ holds just before execution of line 17. Consequently, $i \leq k \leq \min(j, \text{length}(\omega(l)))$ holds for $k = i$ just before executing line 17. □

Lemma 5.32 *When executing Gallop(l), right after executing line 11,* $1 \leq i$.

Proof. We first prove that after executing line 11 $\omega(l)[i] = \text{fi}[l]$ and then using the definition of fi we prove the lemma. Since line 11 assigns $\mathit{index}(I, l, \text{fi}[l])$ to i, after executing this line by Definition 2.8 $\omega(l)[i] = \text{fi}[l]$. Hence by Definition 5.1 $\omega(l)[i]$ will be either END or an object in $\omega(l)$. Also, by Definition 2.7 $\omega(l)[i] = \text{END}$ only if $i = \text{length}(\omega(l)) + 1$ and $\omega(l)[i]$ is an object only if $1 \leq i \leq \text{length}(\omega(l))$. So, in either case the lemma is true. □

Using the next lemma we will prove that whenever Block 7 is executed, all objects with indexes at most j have values less than e and hence the algorithm can change fi so that they become scanned, without violating Invariant 2.

Lemma 5.33 *When during the execution of Gallop(l), for a leaf l, the procedure reaches line 19, if* $j \geq i$ *then* $j \leq \text{length}(\omega(l))$ *and* $\mu(\omega(l)[j]) < \mathsf{e}$.

Proof. We consider two cases bases on Block 5 being executed or not. Due to the condition checked on line 14, Block 5 is not executed only if $j \leq \mathit{length}(\omega(l))$ and $\mu(\omega(l)[j]) < \mathsf{e}$. Hence when this block is not run the lemma is true. So suppose Block 5 is executed. Then, the value of j is determined by one of lines 16 or 17. If line 16 is executed, j is set equal to $i - 1$ and so the equation $j \geq i$ will not hold. Consequently, the lemma is true in this case. Now suppose line 17 is executed rather than line 16. Then, as is clear from the figure, the binary search assigns to j an integer k such that $k \leq \text{length}(\omega(l))$ and $\mu(\omega(l)[k]) < \mathsf{e}$ (by Lemma 5.31 this is feasible). Thus, after executing this line both inequalities hold. Hence, in all cases the lemma is correct. □

Recall that, as mentioned after Definition 5.1, for each line modifying the elements of the array fi we must prove that after the modification elements of the array fi still satisfy Definition 5.1. Now, before discussing Block 7, since line 20 modifies $\text{fi}[l]$, we prove this

claim for the modification applied on this line. Due to the condition checked on line 19, line 20 is executed only if $j \geq i$ and hence by Lemmas 5.32 and 5.33, $1 \leq i \leq j \leq \text{length}(\omega(l))$. Thus, when executing line 20, $1 \leq j+1 \leq \text{length}(\omega(l))+1$ and so by Definition 2.7 $\omega(l)[j+1]$ is either END or an object of l. Hence, by modifying $\mathfrak{fi}[l]$ on line 20, the condition of Definition 5.1 is not violated by $\mathfrak{fi}[l]$.

Now we discuss the next block, that is, Block 7, which is executed when $j \geq i$. In this block, objects $\omega(l)[i], \ldots, \omega(l)[j]$ become scanned and other variables change to meet their definitions of being well-valued. Also, if l is a speedy leaf, Block 6 is executed. This block adds to the solution all objects that in this visiting have become scanned and causes Invariant 3 to be satisfied, as we will show in Lemma 5.36.

We now prove that providing that the preconditions of the procedure hold before the execution of the procedure, after its execution the postconditions of the procedure hold. As Invariant 1 depends on C, we first discuss how the procedure modifies C. Since throughout the procedure the value of the variable e does not change, we do not use any time stamp for this variable in any proof in this section.

Lemma 5.34 *Suppose just after the time bef the algorithm calls Gallop(l), for a leaf l, and at the time bef, the preconditions stated in Figure 5.2 hold. Also, suppose the procedure modifies C. Then, l is a speedy leaf and $\mathsf{C}^{(\text{aft})} = (\mathsf{e}, b, \Phi_{sl}(\mathsf{C}^{(\text{bef})}, \mu, b))$, for some b, $\mathsf{e_s}^{(\text{bef})} \leq b \leq \mathsf{e}$, where aft is the time just after finishing execution of the procedure.*

Proof. We first prove that Block 6 is executed and we conclude that $l = sl(Q)$. Then, defining t as the time just before executing this block and

$$b = \mu\left(\omega(l)[j^{(t)}]\right) +, \tag{5.13}$$

we first show that $\mathsf{e_s}^{(bef)} \leq b \leq \mathsf{e}$ and then we prove that Block 6 sets C equal to $(\mathsf{e}, b, \Phi_{sl}(\mathsf{C}^{(bef)}, \mu, b))$. The argument for proving the claim that Block 6 is executed is simple: the only place that C is changed is in Block 6 and by assumption C is changed. So, the block is executed. As a result, the conditions of lines 19 and 23 are true and thus

as contents of i and j are not modified in Block 7, the following equations hold.

$$l = sl(Q) \tag{5.14}$$

$$j^{(t)} \geq i^{(t)} \tag{5.15}$$

As the first step towards proving $\mathsf{e_s}^{(bef)} \leq b \leq \mathsf{e}$ we now show that $\mathsf{e_s}^{(bef)} \leq b$. By Lemma 5.33, when reaching lines 20 $j \leq \text{length}(\omega(l))$ and $\mu(\omega(l)[j]) < \mathsf{e}$. So, as between lines 20 and 23 j does not change, the following equations hold.

$$j^{(t)} \leq \text{length}(\omega(l)) \tag{5.16}$$

$$\mu(\omega(l)[j^{(t)}]) < \mathsf{e} \tag{5.17}$$

Due to the precondition, when starting the procedure Invariant 3 holds and so $\mathfrak{fi}^{(bef)}[sl(Q)] = find(I, sl(Q), \mathsf{e_s}^{(bef)})$. Therefore, by Equation 5.14 $\mathfrak{fi}^{(bef)}[l] = find(I, l, \mathsf{e_s}^{(bef)})$. Thus, by executing line 11, i is set equal to $index(I, l, find(I, l, \mathsf{e_s}^{(bef)}))$. As after line 11 the variable i does not change, $i^{(t)} = index(I, find(I, l, \mathsf{e_s}^{(bef)}))$ and hence by Definition 2.8

$$\omega(l)[i^{(t)}] = find(I, l, \mathsf{e_s}^{(bef)}). \tag{5.18}$$

By Lemma 5.32 right after executing line 11, $1 \leq i$. Consequently, as after line 11 i is not modified, $1 \leq i^{(t)}$. So, by Equations 5.15 and 5.16,

$$1 \leq i^{(t)} \leq j^{(t)} \leq \text{length}(\omega(l)). \tag{5.19}$$

Therefore, by Definition 2.7 $\omega(l)[i^{(t)}]$ and $\omega(l)[j^{(t)}]$ are objects of l. As a result, as μ is ordered and by Equation 5.15 $j^{(t)} \geq i^{(t)}$, $\mu(\omega(l)[j^{(t)}]) \geq \mu(\omega(l)[i^{(t)}])$. So by Equation 5.18

$$\mu(\omega(l)[j^{(t)}]) \geq \mu(find(I, l, \mathsf{e_s}^{(bef)})). \tag{5.20}$$

By Lemma 3.1 $find(I, sl(Q), \mathsf{e_s}^{(bef)})$ is either END or an object of value at least $\mathsf{e_s}^{(bef)}$ and since $\mu(\text{END}) = \infty$, in either case $\mu(find(I, sl(Q), \mathsf{e_s}^{(bef)})) \geq \mathsf{e_s}^{(bef)}$. Consequently, due to

Equations 5.13 and 5.20,

$$b = \mu(\omega(l)[j^{(t)}])+ > \mu(\omega(l)[j^{(t)}]) \geq \mu(\mathit{find}(I, \mathit{sl}(Q), \mathsf{e_s}^{(bef)})) \geq \mathsf{e_s}^{(bef)}.$$

Therefore, the claim $b \geq \mathsf{e_s}^{(bef)}$ is proved.

Next we complete the proof of the claim $\mathsf{e_s}^{(bef)} \leq b \leq \mathsf{e}$ by showing that $b \leq \mathsf{e}$. It follows from Equation 5.17 that $\langle \mu(\omega(l)[j^{(t)}])+\rangle = \mu(\omega(l)[j^{(t)}]) < \mathsf{e}$ and hence $\mathsf{e} \notin \langle \mu(\omega(l)[j])+\rangle$. Therefore, according to Observation 2.30, $\mathsf{e} \not< \mu(\omega(l)[j])+$ and so $\mu(\omega(l)[j^{(t)}])+ \leq \mathsf{e}$. Consequently, due to Equation 5.13 $b \leq \mathsf{e}$. Thus, as we proved earlier that $\mathsf{e_s}^{(bef)} \leq b$, $\mathsf{e_s}^{(bef)} \leq b \leq \mathsf{e}$.

Due to the definition of $\Phi_{sl}(C, \mu, b)$ (Definition 3.9), in order to prove that line 24 sets $\Phi := \Phi_{sl}(C^{(bef)}, \mu, b)$, we first show that $i^{(t)} = \mathit{pos}(I, \mathit{sl}(Q), \mathsf{e_s}^{(bef)})$ and then prove that $j^{(t)} + 1 = \mathit{pos}(I, \mathit{sl}(Q), b)$. By Equation 5.18 $\omega(l)[i^{(t)}] = \mathit{find}(I, l, \mathsf{e_s}^{(bef)})$. Therefore, due to Equation 5.14, $\omega(\mathit{sl}(Q))[i^{(t)}] = \mathit{find}(I, \mathit{sl}(Q), \mathsf{e_s}^{(bef)})$. Moreover, $\omega(\mathit{sl}(Q))[\mathit{pos}(I, \mathit{sl}(Q), \mathsf{e_s}^{(bef)})] = \mathit{find}(I, \mathit{sl}(Q), \mathsf{e_s}^{(bef)})$, according to Definition 3.4. Hence, $\omega(\mathit{sl}(Q))[i^{(t)}]$ is equal to $\omega(\mathit{sl}(Q))[\mathit{pos}(I, \mathit{sl}(Q), \mathsf{e_s}^{(bef)})]$. This results in the following equation.

$$i^{(t)} = \mathit{pos}(I, \mathit{sl}(Q), \mathsf{e_s}^{(bef)}) \tag{5.21}$$

To prove that $j^{(t)} + 1 = \mathit{pos}(I, \mathit{sl}(Q), b)$, we consider two cases based on $j^{(t)}$ equaling the length of $\omega(l)$ or not. If they are equal, $\omega(l)[j^{(t)}]$ is the last object of l and hence all objects of l have values at most $\mu(\omega(l)[j^{(t)}]) < \mu(\omega(l)[j^{(t)}])+ = b$ (Equation 5.13). Therefore, values of all objects of l are less than b. So, by Definition 3.4 $\mathit{pos}(I, l, b)$ is one more than the length of $\omega(l)$, that is, $j^{(t)} + 1$. Hence, in this case, $\mathit{pos}(I, \mathit{sl}(Q), b) = \mathit{pos}(I, l, b) = j^{(t)} + 1$.

Now consider the other case, in which the length of $\omega(l)$ is not $j^{(t)}$. We prove that the smallest integer k, satisfying $1 \leq k \leq \text{length}(\omega(l))$ and $\mu(\omega(l)[k]) \geq b$ is $j^{(t)} + 1$. A consequence of this result will be the fact that $\omega(l)[j^{(t)} + 1]$ is an object of l of value at least b and thus l has an object of value at least b. Then, it follows from Definition 3.4 that $\mathit{pos}(I, l, b) = j^{(t)} + 1$. To prove the aforementioned claim we prove the following three sub-claims. First we show that $1 \leq j^{(t)} + 1 \leq \text{length}(\omega(l))$. Then we prove that $\mu(\omega(l)[j^{(t)} + 1]) \geq b$ and finally, we will argue that $\mu(\omega(l)[k]) \ngeq b$, for every k, $1 \leq k < j^{(t)} + 1$.

We now present proofs for the above sub-claims. Let us first prove the sub-claim $1 \leq j^{(t)}+1 \leq \text{length}(\omega(l))$. It follows from Equation 5.19 and the fact that by assumption $\text{length}(\omega(l)) \neq j^{(t)}$ that $1 \leq j^{(t)} < \text{length}(\omega(l))$. Hence, as j is an integer variable the inequality $1 \leq j^{(t)}+1 \leq \text{length}(\omega(l))$ is correct. Now we prove the inequality $\mu(\omega(l)[j^{(t)}+1]) \geq b$. It follows from Equation 5.19 that $1 \leq j^{(t)} \leq \text{length}(\omega(l))$ and thus $\omega(l)[j^{(t)}]$ is an object of l. Furthermore, as we proved above $1 \leq j^{(t)}+1 \leq \text{length}(\omega(l))$ and hence $\omega(l)[j^{(t)}+1]$ is also an object of l. Therefore, as μ is ordered, $\mu(\omega(l)[j^{(t)}+1]) > \mu(\omega(l)[j^{(t)}])$, and so by Observation 2.29 $\mu(\omega(l)[j^{(t)}+1]) > \mu(\omega(l)[j^{(t)}])_+ = b$ (Equation 5.13). So the second sub-claim is proved.

Next we prove the last sub-claim by considering an integer k, $1 \leq k < j^{(t)}+1$, and by proving that $\mu(\omega(l)[k]) \not\geq b$. Since $1 \leq k < j^{(t)}+1$, $1 \leq k \leq j^{(t)}$. So, $1 \leq k \leq j^{(t)} \leq \text{length}(\omega(l))$ (Equation 5.19) and thus $\omega(l)[k]$ and $\omega(l)[j^{(t)}]$ are objects of l. Hence as $k \leq j^{(t)}$, $\mu(\omega(l)[k]) \leq \mu(\omega(l)[j^{(t)}]) < \mu(\omega(l)[j^{(t)}])_+ = b$ (Equation 5.13). Consequently, $\mu(\omega(l)[k]) \not\geq b$. Thus, the smallest integer k, satisfying $1 \leq k \leq \text{length}(\omega(l))$ and $\mu(\omega(l)[k]) \geq b$ is $j^{(t)}+1$. Hence, by Definition 3.4 $pos(I, sl(Q), b) = pos(I, l, b) = j^{(t)}+1$. So, in both cases, $j^{(t)} = \text{length}(\omega(l))$ and $j^{(t)} \neq \text{length}(\omega(l))$, we have proved that

$$pos(I, sl(Q), b) = j^{(t)} + 1. \tag{5.22}$$

As a result,

$$\begin{array}{lll}
\Phi_{sl}(C^{(bef)}, \mu, b) = \Phi^{(bef)} \odot (sl(Q), pos(I, sl(Q), \mathsf{e_s}^{(bef)}), & & \\
\qquad pos(I, sl(Q), b) - 1) & & \text{by Definition 3.9} \\
= \Phi^{(bef)} \odot (sl(Q), i^{(t)}, pos(I, sl(Q), b) - 1) & & \text{by Equation 5.21} \\
= \Phi^{(bef)} \odot (sl(Q), i^{(t)}, j^{(t)}) & & \text{by Equation 5.22.}
\end{array}$$

Now we can prove that the procedure modifies C in the same way that is claimed by the lemma. Since we proved Block 6 is executed, lines 24 and 25 set Φ equal to $\Phi^{(bef)} \odot (sl(Q), i^{(t)}, j^{(t)}) = \Phi_{sl}(C^{(bef)}, \mu, b)$ and $\mathsf{e_s}$ equal to $\mu(\omega(l)[j^{(t)}])_+ = b$ (Equation 5.13), respectively. Hence C is set equal to $(\mathsf{e}, b, \Phi_{sl}(C^{(bef)}, \mu, b))$ while we showed that $\mathsf{e_s}^{(bef)} \leq b \leq \mathsf{e}$. □

We next use the previous lemma to prove that after a legal call to the procedure Gallop

Invariant 1 holds. Once again we emphasize that since e does not change throughout the procedure, in our proofs we do not use time stamps for the variable e.

Lemma 5.35 *Suppose before calling Gallop(l), for a leaf l, preconditions stated in Figure 5.2 hold. Then, after executing Gallop(l), Invariant 1 holds.*

Proof. Defining *bef* and *aft* as times before and after running Gallop(l), respectively, we consider the two cases $\mathsf{C}^{(bef)} = \mathsf{C}^{(aft)}$ and $\mathsf{C}^{(bef)} \neq \mathsf{C}^{(aft)}$, separately. In the first case since by the precondition at the time *bef* Invariant 1 holds and thus $\mathsf{C}^{(bef)}$ is valid, $\mathsf{C}^{(aft)}$ is also valid and hence the invariant holds. So suppose $\mathsf{C}^{(bef)} \neq \mathsf{C}^{(aft)}$. Then, by Lemma 5.34

$$l = sl(Q) \tag{5.23}$$

and $\mathsf{C}^{(aft)} = (\mathsf{e}, b, \Phi_{sl}(\mathsf{C}^{(bef)}, \mu, b))$, for some b,

$$\mathsf{e_s}^{(bef)} \leq b \leq \mathsf{e}. \tag{5.24}$$

Hence the following equations hold.

$$\mathsf{e_s}^{(aft)} = b \tag{5.25}$$

$$\Phi^{(aft)} = \Phi_{sl}(\mathsf{C}^{(bef)}, \mu, b) \tag{5.26}$$

We first prove that $\mathsf{C}^{(aft)}$ is an eliminating configuration and then we show that this eliminating configuration is valid. By Equations 5.25 and 5.24 $\mathsf{e_s}^{(aft)} = b \leq \mathsf{e}$ and so as by Lemma 5.34 Q has a speedy leaf, by Definition 3.2 $\mathsf{C}^{(aft)}$ is an eliminating configuration. Now we prove that $\mathsf{C}^{(aft)}$ is valid by showing that the conditions defined in Definition 3.5 are met by $\mathsf{C}^{(aft)}$, one by one, starting from the third one. Then we prove the first and the second conditions, in order.

The proof of the third condition is simple. As by Equation 5.23 $l = sl(Q)$, Q has a speedy leaf. Also, as by the precondition at the time *bef* Invariant 1 holds, $\mathsf{C}^{(bef)}$ is valid. So, since $\mathsf{e_s}^{(bef)} \leq b$ (Equation 5.24), by Lemma 3.16 $\Phi_{sl}(C^{(bef)}, \mu, b)$ is a partial solution to I. Hence, as by Equation 5.26 $\Phi^{(aft)} = \Phi_{sl}(\mathsf{C}^{(bef)}, \mu, b)$, $\Phi^{(aft)}$ is a partial solution to I and thus the third condition is true.

Now we prove the first condition by proving the following two statements. First, the value of every object in the expansion of $\Phi^{(aft)}$ is less than $\mathsf{e_s}^{(aft)}$. Second, for every member a of the value set of the root satisfying $a < \mathsf{e_s}^{(aft)}$, there is an object of value a in the expansion of $\Phi^{(aft)}$. Once we have proved these two statements, we may conclude that for every member a of the value set of the root, $a < \mathsf{e_s}^{(aft)}$ if and only if there is an object of value a in the expansion of $\Phi^{(aft)}$ and so the first condition is proved.

We now prove the two above statements, starting from the first one. As explained, $\mathsf{C}^{(bef)}$ is valid and by Equation 5.24 $\mathsf{e_s}^{(bef)} \leq b \leq \mathsf{e}$. Hence, by Lemma 3.18 the value of every object in the expansion of $\Phi_{sl}(\mu, \mathsf{C}^{(bef)}, b)$ is less than b. So, since $\Phi^{(aft)} = \Phi_{sl}(\mu, \mathsf{C}^{(bef)}, b)$ (Equation 5.26) and $b = \mathsf{e_s}^{(aft)}$ (Equation 5.25), the value of every object in the expansion of $\Phi^{(aft)}$ is less than $\mathsf{e_s}^{(aft)}$. Therefore, the first statement is correct. To prove the second statement, we consider a member a of the value set of the root satisfying $a < \mathsf{e_s}^{(aft)}$ and we prove that there is an object of value a in the expansion of $\Phi^{(aft)}$. As by Equation 5.25 $b = \mathsf{e_s}^{(aft)}$, a is a member of the value set of the root less than b. Hence, as $\mathsf{C}^{(bef)}$ is valid and $b \leq \mathsf{e}$ (Equation 5.24), by Lemma 3.19 there is an object of value a in the expansion of $\Phi_{sl}(\mu, \mathsf{C}^{(bef)}, b)$. So, due to Equation 5.26, an object of value a exists in $\Phi^{(aft)}$. Hence the second statement defined above is also correct. Thus, the first condition of Definition 3.5 is true, as explained.

Finally, we consider the second condition of Definition 3.5. We first prove that for every value a in the value set of the root in $N(I)$ such that $a < \mathsf{e}$, $a < \mathsf{e_s}^{(aft)}$ and then we use this result to show that the second condition of Definition 3.5 is equivalent to the first condition of that definition. So consider a value a in the value set of the root in $N(I)$ satisfying $a < \mathsf{e}$. Since, as explained, $\mathsf{C}^{(bef)}$ is valid and by assumption $a < \mathsf{e}$, by the second condition of Definition 3.5 for $\mathsf{C}^{(bef)}$, there is an object of value a in the expansion of $\Phi^{(bef)}$. Also, as a is in the value set of the root in $N(I)$ and by Observation 2.32 the value set of the root in $N(I)$ is a subset of the value set of the root in I, a is in the value set of the root in I. Hence, as we proved an object of value a is in the expansion of $\Phi^{(bef)}$ we may conclude that $a < \mathsf{e_s}^{(bef)}$, due to the first condition of Definition 3.5 for $\mathsf{C}^{(bef)}$. So every member of the value set of the root in $N(I)$ less than e is less than $\mathsf{e_s}^{(bef)} \leq b = \mathsf{e_s}^{(aft)}$ (Equations 5.24 and 5.25). Moreover, every member of the value set of the root in $N(I)$ less than $\mathsf{e_s}^{(aft)}$ is also less than e because $\mathsf{e_s}^{(aft)} = b \leq \mathsf{e}$ (Equations 5.25 and 5.24). Hence,

an object of the value set of the root in $N(I)$ is less than e if and only if it is less than $\mathsf{e_s}^{(aft)}$. Also, since by Observation 2.32 a value in the value set of the root in $N(I)$ is in the value set of the root in I, by the first condition of Definition 3.5 for $\mathsf{C}^{(aft)}$ (proved above), there is an object of value a in the expansion of $\Phi^{(aft)}$ if and only if $a < \mathsf{e_s}^{(aft)}$. Combining these two results, we can conclude that there is an object of value a in the expansion of $\Phi^{(aft)}$, for an a in the value set of the root in $N(I)$, if and only if $a < \mathsf{e}$, proving the second condition of Definition 3.5.

Having proved that $\mathsf{C}^{(aft)}$ satisfies all the three conditions of Definition 3.5, we now have completed the proof of the claim that $\mathsf{C}^{(aft)}$ is valid (Definition 3.5) and so Invariant 1 holds when the procedure exits. □

We now show the correctness of Invariant 3 after the execution of the procedure.

Lemma 5.36 *Suppose before calling Gallop(l), for a leaf l, the preconditions stated in Figure 5.2 hold. Then, after executing the procedure, Invariant 3 holds.*

Proof. Considering the fact that, by its definition, Invariant 3 depends only on $\mathsf{e_s}$ and $\mathfrak{fi}[sl(Q)]$, we first prove that if Block 7 is not executed or l is not a speedy leaf then neither $\mathsf{e_s}$ nor $\mathfrak{fi}[sl(Q)]$ change and so after executing the procedure Invariant 3 holds as by the precondition before running it this invariant holds. After that, we prove the lemma in other cases. Variables $\mathsf{e_s}$ and $\mathfrak{fi}[sl(Q)]$ only may change in lines 20 and 25, respectively, and both of these lines are in Block 7. So when Block 7 is not executed nothing happens to the aforementioned variables and so the lemma is true, as explained. Thus, suppose Block 7 is run. Then, if l is not a speedy leaf line 20 does not change $\mathfrak{fi}[sl(Q)]$ and due to the conditions checked on line 23, line 25 is not executed to change $\mathsf{e_s}$. Hence again the lemma is true since $\mathsf{e_s}$ and $\mathfrak{fi}[sl(Q)]$ are not changed.

Now in the case that Block 7 is run and $l = sl(Q)$ we prove that after executing the procedure Invariant 3 holds. Defining t and *aft* as times just before executing line 19 and just after exiting the procedure and

$$o = \omega(l)[j^{(t)}], \tag{5.27}$$

we take the following steps. We first prove that o is an object and then we show that

$\omega(l)[j^{(t)}+1] = \mathit{find}(I, \mathit{sl}(Q), \mu(o)+)$. After that, we prove that Block 6 is executed and $\mathsf{e_s}$ is modified so that Invariant 3 is satisfied, completing the proof of the lemma.

To show that $o = \omega(l)[j^{(t)}]$ is an object of l, we prove that $1 \leq i^{(t)} \leq j^{(t)} \leq \text{length}(\omega(l))$. By Lemma 5.32, right after executing line 11, $i \geq 1$. Hence, as after line 11 i does not change, $1 \leq i^{(t)}$. Also, since by assumption Block 7 is executed, it follows from the condition checked on line 19 that $i^{(t)} \leq j^{(t)}$. So, $1 \leq j^{(t)}$ and by Lemma 5.33 $j^{(t)} \leq \text{length}(\omega(l))$. Hence $1 \leq j^{(t)} \leq \text{length}(\omega(l))$ and so $o = \omega(l)[j^{(t)}]$ is an object.

Next we prove that $\omega(l)[j^{(t)}+1] = \mathit{find}(I, \mathit{sl}(Q), \mu(o)+)$. Since as we proved o is an object of l and so l has an object of value $\mu(o)$, by Lemma 3.2 $\mathit{find}(I, l, \mu(o))$ is an object of value $\mu(o)$. Therefore, as $\mathit{find}(I, l, \mu(o))$ and o are objects of l with the same value and thus $o = \mathit{find}(I, l, \mu(o))$. Also, as $\mu(\mathit{find}(I, l, \mu(o))) = \mu(o)$, applying Lemma 5.26 for $b = \mu(o)$ and considering the equation $o = \mathit{find}(I, l, \mu(o))$, we may conclude that $\mathit{find}(I, l, \mu(o)+) = \text{END}$ if o is the last object and $\mathit{find}(I, l, \mu(o)+)$ is the object following o, otherwise. If o is the last object, as by Equation 5.27 $o = \omega(l)[j^{(t)}]$, by definition $\omega(l)[j^{(t)}+1] = \text{END}$. Also if o is not the last object since $o = \omega(l)[j^{(t)}]$, by definition $\omega(l)[j^{(t)}+1]$ is the object after o. Hence, in both cases $\mathit{find}(I, l, \mu(o)+) = \omega(l)[j^{(t)}+1]$.

Now we complete the proof by showing that Block 6 changes $\mathsf{e_s}$ so that the equation $\mathfrak{fi}[\mathit{sl}(Q)] = \mathit{find}(I, \mathit{sl}(Q), \mathsf{e_s}^{(aft)})$ and hence Invariant 3 holds. By assumption Block 7 is run. Also, as by assumption $l = \mathit{sl}(Q)$, the condition of the "if" statement of line 23 is true and hence Block 6 is executed. Thus, because of line 25 $\mathsf{e_s}^{(aft)} = \mu(\omega(l)[j^{(t)}])+$. So, as by Equation 5.27 $o = \omega(l)[j^{(t)}]$,

$$\mathsf{e_s}^{(aft)} = \mu(o)+. \tag{5.28}$$

Also due to line 20, $\mathfrak{fi}^{(aft)}[l] = \omega(l)[j^{(t)}+1]$ and hence as by assumption $l = \mathit{sl}(Q)$, $\mathfrak{fi}^{(aft)}[\mathit{sl}(Q)] = \omega(l)[j^{(t)}+1]$. So, as we proved $\mathit{find}(I, l, \mu(o)+) = \omega(l)[j^{(t)}+1]$, $\mathfrak{fi}^{(aft)}[\mathit{sl}(Q)] = \mathit{find}(I, \mathit{sl}(Q), \mu(o)+)$. As a result, due to Equation 5.28, $\mathfrak{fi}^{(aft)}[\mathit{sl}(Q)] = \mathit{find}(I, \mathit{sl}(Q), \mathsf{e_s}^{(aft)})$. Therefore, after executing the algorithm Invariant 3 holds. □

Since there is no line in the procedure Gallop directly modifying a variable other than changing variables and local variables i and j and there is no procedure call line, the correctness of the next Observation is trivial. We now in Lemma 5.38 prove that if before executing the procedure Gallop its preconditions hold, after executing the procedure, its

postconditions hold.

Observation 5.37 *By executing Gallop(l), for a leaf l of Q, no variable other than local variables and changing variables of the procedure, listed in Figure 5.2, change.*

Lemma 5.38 *Suppose before calling Gallop(l), for a leaf l, the preconditions stated in Figure 5.2 hold. Then, after executing the procedure, its postconditions hold.*

Proof. We first prove that after existing the procedure Invariant 2 holds and then we show that changing variables will be well-valued. Also, by Lemmas 5.35 and 5.36 after executing the procedure Invariants 1 and 3 hold. So, after proving the above claims all postconditions of the procedure will have been proved to hold after the execution of the procedure. Consider the times *bef*, *aft*, and t just before and just after executing the procedure and just before executing line 19, respectively.

We now prove that Invariant 2 holds after the execution of the procedure. To do this, we consider an *aft*-scanned object o of a leaf v and we prove that $\mu(o) < \mathsf{e}$. We first prove the claim in the case in which Block 7 is not executed or $v \neq l$ and then we consider other cases. As the array $\mathfrak{fi}$ is modified only in line 20 of Block 7 and only the element $\mathfrak{fi}[l]$ of the array $\mathfrak{fi}$ is modified, when Block 7 is not executed or $v \neq l$ the variable $\mathfrak{fi}[v]$ is not changed. So, in such cases, $\mathfrak{fi}^{(aft)}[v] = \mathfrak{fi}^{(bef)}[v]$. Consequently, as by assumption o is *aft*-scanned, by Definition 5.1 $\mu(o) < \mathfrak{fi}^{(aft)}[l] = \mathfrak{fi}^{(bef)}[l]$ and hence by Definition 5.1 o is *bef*-scanned. Therefore, as by precondition Invariant 2 holds at the time *bef*, $\mu(o) < \mathsf{e}$.

Now we consider the situation in which Block 7 is executed and $v = l$. Then, in line 20 $\mathfrak{fi}[l]$ is set equal to $\omega(l)[j^{(t)} + 1]$. Hence, as by assumption o is *aft*-scanned, by Definition 5.1 its value is less than $\mu(\omega(l)[j^{(t)} + 1])$. So, as o is an object of l, $\mu(o) \leq \mu(\omega(l)[j^{(t)}])$. Moreover, by Lemma 5.33 when executing line 19 $\mu(\omega(l)[j]) < \mathsf{e}$. Hence, $\mu(o) \leq \mu(\omega(l)[j^{(t)}]) < \mathsf{e}$. So, in any case $\mu(o) < \mathsf{e}$ where o was selected as an arbitrary *aft*-scanned object. As a result, at the time *aft* Invariant 2 holds.

To complete the proof, we now show that after executing the procedure $\mathsf{W}[l]$ and $\mathsf{M}[l]$ (in the case in which l is in $N(Q)$) are well-valued. As l is a leaf, definitions of these two variables being well-valued (Definition 5.8 and 5.6) depend only on variables $\mathfrak{fi}[l]$, $\mathsf{W}[l]$, $\mathsf{M}[l]$, e, and E and values of these variables only may change in Block 7. Hence if this block is not executed values of these variables remain well-valued as before executing the procedure by

precondition values of these variables were well-valued. So we suppose Block 7 is executed. We first prove that $\mathsf{M}[l]$ becomes well-valued and then we show that $\mathsf{W}[l]$ will be well-valued when the procedure exits. So first consider the variable $\mathsf{M}[l]$. When l is in $N(Q)$, since by executing line 21, $\mathsf{M}[l]$ is set equal to $\mu(\mathfrak{fi}[l])$ and after running this line values of $\mathsf{M}[l]$ and $\mathfrak{fi}[l]$ do no change, after executing the procedure $\mathsf{M}[l] = \mu(\mathfrak{fi}[l])$ and thus by Definition 5.6 $\mathsf{M}[l]$ is well-valued.

Now we show that the variable $\mathsf{W}[l]$ becomes well-valued if Block 7 is executed. As when $\mathsf{e} \in \{-\infty, \infty\}$, by Definition 5.8 $\mathsf{W}[v]$ is always well-valued, we suppose $\mathsf{e} \notin \{-\infty, \infty\}$. Since after executing line 22 values of $\mathfrak{fi}[l]$, $\mathsf{W}[l]$, e, and E do not change and Definition 5.8 only depends on these variables, it suffices to prove that right after executing line 22, $\mathsf{W}[l]$ satisfies the four conditions of Definition 5.8 and hence is well values. To prove the first and second conditions of Definition 5.8 we discuss the two cases $\mu(\mathfrak{fi}[l]) = \mathsf{e}$ and $\mu(\mathfrak{fi}[l]) \neq \mathsf{e}$ separately. If $\mu(\mathfrak{fi}[l]) = \mathsf{e}$, line 22 sets $\mathsf{W}[l] := \mathfrak{fi}[l]$. Therefore, after this line $\mu(\mathsf{W}[l]) = \mu(\mathfrak{fi}[l]) = \mathsf{e}$ and so the first and the second conditions of Definition 5.8 are satisfied. When $\mu(\mathfrak{fi}[l]) \neq \mathsf{e}$, line 22 sets $\mathsf{W}[l] :=$ BEG and hence as we supposed $\mathsf{e} \notin \{-\infty, \infty\}$, after this setting we will have $\mathsf{e} < -\infty = \mu(\text{BEG}) = \mu(\mathsf{W}[l])$. Thus again the first and the second conditions of Definition 5.8 will be satisfied. The third and the fourth conditions of Definition 5.8 are satisfied as l is a leaf. So, after executing line 22 $\mathsf{W}[v]$ is well-valued. Consequently after exiting the procedure $\mathsf{W}[v]$ is well-valued, as explained.

Now we have proved that after executing the procedure all of its postconditions hold and so the lemma is correct. □

As the last result of this section, we prove that every sub-intersection tree that was an eyewitness before calling the procedure Gallop remains an eyewitness after executing the procedure Gallop. This result will be used to determine that which changes to contents of the array W are needed to ensure that its elements remain well-valued after calling Gallop.

Lemma 5.39 *Suppose before calling Gallop(l), for a leaf l, preconditions of Figure 5.2 hold and T is an eyewitness of a node v of Q. Then after exiting Gallop(l) T is still an eyewitness of v.*

Proof. Since the definition of T being an eyewitness (Definition 5.5) only depends on e and $\mu(\mathfrak{fi}[u])$, for all leaves u of T, it suffices to prove that when T is an eyewitness, during

execution of procedure Gallop(l) contents of variables e and fi[u], for leaves u of T, do not change. By Observation 5.37 only changing and local variables i and j of the procedure are altered. The variable e is not a changing variable and for every leaf u of T, fi[u] is a changing variable only if $u = l$. Thus, to prove the lemma it suffices to show that if l is a leaf of T, fi[l] does not change. Since fi[l] is only modified in Block 7, we suppose this block is executed as otherwise, fi[l] remains unchanged and hence the lemma is true, as explained. Defining *bef*, *aft*, and t as times just before and just after executing Gallop(l) and the time just before executing line 19, we first prove that $\mu(\text{fi}^{(aft)}[l]) \geq \mu(\text{fi}^{(bef)}[l])$ and then we show that $\mu(\text{fi}^{(aft)}[l]) > \mu(\text{fi}^{(bef)}[l])$ is not possible. In this way it is proved that $\mu(\text{fi}^{(aft)}[l]) = \mu(\text{fi}^{(bef)}[l])$ and hence, as explained, the lemma is true because fi[l] is not modified.

To prove $\mu(\text{fi}^{(aft)}[l]) \geq \mu(\text{fi}^{(bef)}[l])$ we first discuss how $\text{fi}^{(aft)}[l]$ is evaluated. The variable i is initialized in line 11 and is never modified after that. Therefore by assignment on line 11 $i^{(t)} = \mathit{index}(I, l, \text{fi}^{(bef)}[l])$ and so by Definition 2.8 $\omega(l)[i^{(t)}] = \text{fi}^{(bef)}[l]$. Line 20 assigns $\omega(l)[j^{(t)} + 1]$ to fi[l] and after line 20 the procedure does not change fi[l]. Hence, $\text{fi}^{(aft)}[l] = \omega(l)[j^{(t)} + 1]$. So, as we showed that $\omega(l)[i^{(t)}] = \text{fi}^{(bef)}[l]$, in order to prove $\mu(\text{fi}^{(aft)}[l]) \geq \mu(\text{fi}^{(bef)}[l])$ it suffices to show that $\mu(\omega(l)[j^{(t)} + 1]) \geq \mu(\omega(l)[i^{(t)}])$.

To prove $\mu(\omega(l)[j^{(t)} + 1]) \geq \mu(\omega(l)[i^{(t)}])$, we first prove that $\mu(\omega(l)[j^{(t)}]) \geq \mu(\omega(l)[i^{(t)}])$ and then we show that $\mu(\omega(l)[j^{(t)} + 1]) \geq \mu(\omega(l)[j^{(t)}])$. Since after line 11 i does not change by Lemma 5.32 $1 \leq i^{(t)}$. Also as by assumption Block 7 is executed, by condition check on line 19 when reaching line 19 $j \geq i$ and hence $j^{(t)} \geq i^{(t)}$. As a result, by Lemma 5.33 $j^{(t)} \leq \text{length}(\omega(l))$. Thus, as we proved earlier that $1 \leq i^{(t)}$ and $j^{(t)} \geq i^{(t)}$, $1 \leq i^{(t)} \leq j^{(t)} \leq \text{length}(\omega(l))$. So, $\omega(l)[i^{(t)}]$ and $\omega(l)[j^{(t)}]$ are objects of l and hence as μ is ordered and $i^{(t)} \leq j^{(t)}$, the inequality $\mu(\omega(l)[j^{(t)}]) \geq \mu(\omega(l)[i^{(t)}])$ is correct.

In order to show the correctness of the inequality $\mu(\omega(l)[j^{(t)} + 1]) \geq \mu(\omega(l)[j^{(t)}])$, we consider two cases based on $\omega(l)[j^{(t)}]$ being the last object or not. If $\omega(l)[j^{(t)}]$ is not the last object of l, $\omega(l)[j^{(t)} + 1]$ is an object of l and thus since μ is ordered, $\mu(\omega(l)[j^{(t)} + 1]) \geq \mu(\omega(l)[j^{(t)}])$ is true. If $\omega(l)[j^{(t)}]$ is the last object of l then by definition $\omega(l)[j^{(t)} + 1] = \text{END}$ and hence $\mu(\omega(l)[j^{(t)} + 1]) = \mu(\text{END}) = \infty \geq \mu(\omega(l)[j^{(t)}])$. Therefore, in both cases $\mu(\omega(l)[j^{(t)} + 1]) \geq \mu(\omega(l)[j^{(t)}])$. So, as we proved earlier that $\mu(\omega(l)[j^{(t)}]) \geq \mu(\omega(l)[i^{(t)}])$, $\mu(\omega(l)[j^{(t)} + 1]) \geq \mu(\omega(l)[i^{(t)}])$. Hence, the inequality $\mu(\text{fi}^{(aft)}[l]) \geq \mu(\text{fi}^{(bef)}[l])$ is correct, as

explained before.

Now we show that $\mu(\mathfrak{fi}^{(aft)}[l]) \not> \mu(\mathfrak{fi}^{(bef)}[l])$. Assume to the contrary that $\mu(\mathfrak{fi}^{(aft)}[l]) > \mu(\mathfrak{fi}^{(bef)}[l])$. Then, by Definition 5.1 $\mathfrak{fi}^{(bef)}[l]$ is an *aft*-scanned object. Also, by Definition 5.5 $\mu(\mathfrak{fi}^{(bef)}[l]) = \mathsf{e}$ since at the time *bef* by assumption l is a leaf of the eyewitness T. Therefore, at the time *aft* Invariant 2 does not hold because the value of the *aft*-scanned object $\mathfrak{fi}^{(bef)}[l]$ is not less than e. So after executing Gallop(l) postconditions of this procedure do not hold. This contradicts Lemma 5.38. Hence the assumption $\mu(\mathfrak{fi}^{(aft)}[l]) > \mu(\mathfrak{fi}^{(bef)}[l])$ is wrong.

Now as we proved $\mu(\mathfrak{fi}^{(aft)}[l]) \not> \mu(\mathfrak{fi}^{(bef)}[l])$ and $\mu(\mathfrak{fi}^{(aft)}[l]) \geq \mu(\mathfrak{fi}^{(bef)}[l])$, we may conclude that $\mu(\mathfrak{fi}^{(aft)}[l]) = \mu(\mathfrak{fi}^{(bef)}[l])$. Hence, $\mathfrak{fi}^{(aft)}[l] = \mathfrak{fi}^{(bef)}[l]$ as distinct objects of the same leaf have different values. So, the lemma is true. □

5.4.2 Visiting Internal Nodes

In this part we discuss the negative and positive visit procedures. We explain how they differ, how each of them selects the sequence of children to be visited, and how they update the arrays W and M.

Figures 5.3 and 5.4 show the negative and positive visit procedures. As can be seen, when a node v is being negative-visited (positive-visited), depending on v being a leaf, an intersection node, or a union node, one of the three "case" parts of the "switch" block starting at line 26 (line 45, respectively) is executed. If v is a leaf, Visit$^-(v)$ or Visit$^+(v)$, whichever is being executed, just calls Gallop(v). If v is an internal node, as explained before, for a number of times, each time a child of v is selected and is recursively visited.

Now we explain how the visit algorithms select an appropriate subset of children of a node v, when visiting v, to be visited. Each of our visiting algorithms has its own policy. To match the difficulty function, the negative visit algorithm is designed such that when a node v is being negative-visited, the algorithm spends roughly at most $L^-(v)$ time. If v is an intersection node, in the loop starting at line 28, the algorithm visits every child u of v once. Then, as we will prove in the next chapter, assuming every child u of v has spent at most its own $L^-(u)$ quota in time, the whole visit to v does not take more than $\sum_u L^-(u) = L^-(v)$ time. If v is a union node, this $L^-(v)$ quota is first saved in a variable $\mathsf{Credit}[v]$ (line 36), and after visiting every child u of v the algorithm decreases $\mathsf{Credit}[v]$ as

much as $L^-(u)$ (line 40). The algorithm stops visiting children of v when $\mathsf{Credit}[v]$ is not big enough to visit the next node u that it selects, that is, $\mathsf{Credit}[v] < L^-(u)$ (line 38). In this situation, $\mathsf{Credit}[v]$ is not reset. This value is kept as "savings" for v to be used in the next negative-visiting of v.

The positive-visiting of nodes is similar. The positive visit algorithm spends at most $L^+(v)$ when visiting a node v. So, similar to what the negative visit algorithm does, when positive-visiting a union node v, the algorithm visits every child of v in the loop starting at line 47. When positive-visiting an intersection node v, the visit algorithm increases $\mathsf{Credit}[v]$ as much as $L^+(v)$ (line 52) and after visiting every child u of v, it decreases $\mathsf{Credit}[v]$ as much as $L^+(u)$ (line 57).

The other issue is how to select the next child of a node to be visited. Suppose elements of the arrays W and M are well-valued. As we mentioned, while visiting an intersection node in the negative visit algorithm or visiting a union node in the positive visit algorithm, all children are visited. In these cases the algorithm visits the children in an arbitrary order. When visiting a union node in the negative visit algorithm, each time the algorithm selects a child u of v minimizing $\mathsf{M}[u]$ and visits u. The reason is that the negative-visit procedure is optimized for when e is not in the value set of the root in $N(I)$ and in such cases the algorithm tends to use Observation 5.8 to increase e. According to Observation 5.8, the bigger the result of the expression *maxmin*(root) is, the bigger the value that can be assigned to e. Also, as by assumption elements of the array M are well-valued, *maxmin*(root) = M[root] (Lemma 5.14). Therefore, the algorithm tries to maximize M[root]. Consequently, as by Definition 5.14 the value of M[root] is evaluated recursively, the algorithm seeks to maximize $\mathsf{M}[u]$, for all descendants u of v. When visiting v, in order to maximize $\mathsf{M}[v]$, the algorithm needs to maximizes $\min_u \mathsf{M}[u]$ where the minimum is taken over all children u of v (Definition 5.6). Hence, the negative visit algorithm selects the child u minimizing $\mathsf{M}[u]$ and visits it in hope of increasing $\mathsf{M}[u]$.

The story when visiting an intersection node in the positive visit algorithm is similar. In such situations, each time the algorithm selects a child u minimizing $\mathsf{W}[u]$ and visits u. The reason is that the positive visit algorithm is optimized for when e is in the value set of the root in I and in such cases the algorithm tends to use Lemma 5.9 to increase e. Hence, due to Lemma 5.9, the positive visit algorithm wishes to set $\mathsf{W}[v]$ to an object of value e

and for this purpose, because of Definition 5.8, first for every child u of v, $\mathsf{W}[u]$ should be an object of value e. So the algorithm selects a child u such that $\mu(\mathsf{W}[u]) \neq \mathsf{e}$. As by the first condition of Definition 5.8 $\mu(\mathsf{W}[u]) \leq \mathsf{e}$, for every child u, to select a child u satisfying $\mu(\mathsf{W}[u]) \neq \mathsf{e}$ it suffices to select a child u minimizing $\mu(\mathsf{W}[u])$.

Now we explain how the next child to be visited is selected in an efficient manner. For every node v, we create a heap $\mathsf{H}[v]$ that stores all children u of v in $N(Q)$ with the key $\mathsf{M}[u]$ if v is a union node, and stores all children u of v with the key $\mathsf{W}[u]$ if v is an intersection node. Also, for the sake of efficiency of updating $\mathsf{M}[v]$, we create a heap $\mathsf{G}[v]$, for every intersection node v, storing all children u of v with the key $-\mathsf{M}[u]$. When negative-visiting (positive-visiting) a union (an intersection) node v the algorithm easily can select a node u minimizing $\mathsf{M}[u]$ (minimizing $\mathsf{W}[u]$) by looking at the top of the heap $\mathsf{H}[v]$ (lines 37, 44, 53, and 64).

Now we describe the method of updating elements of the arrays H, G, M, and W. We explain the key points of the algorithm in this part and we leave more details to be explained in the proofs. Some time after visiting children of v, the algorithm must update $\mathsf{H}[v]$ and in the case in which v is an intersection node, $\mathsf{G}[v]$ and $\mathsf{Counter}[v]$, and then using these variables, it should update $\mathsf{W}[v]$ and $\mathsf{M}[v]$. After that the negative (positive) visit algorithm visits a child u of a union node (an intersection node) v, it updates $\mathsf{H}[v]$ and if v is an intersection node, it updates $\mathsf{G}[v]$ also (lines 42 and **??**), immediately. But, when negative-visiting (positive-visiting) an intersection (a union) node v, since all children are visited in an arbitrary order, while visiting children, contents of $\mathsf{H}[v]$, $\mathsf{G}[v]$, $\mathsf{W}[v]$, and $\mathsf{M}[v]$ are not used for selecting the next child. So, instead of updating heaps after visiting each child, the algorithm rebuilds $\mathsf{H}[v]$ and in the case of intersection node $\mathsf{G}[v]$ from scratch after visiting all children of v (lines 34 and 50).

After updating the heaps, the algorithm uses the heaps for updating $\mathsf{M}[v]$, for every internal node v. If v is a union node, by Definition 5.6 the algorithm has to set $\mathsf{M}[v]$ equal to the minimum of $\mathsf{M}[u]$, for all children u of v in $N(Q)$. The algorithm evaluates this minimum by looking at the top the heap $\mathsf{H}[v]$ (lines 43 and 51). If v is an intersection node, due to Definition 5.6, the algorithm sets $\mathsf{M}[v]$ equal to the maximum value of $\mathsf{M}[u]$, for all children u of v by looking at the top of the heap $\mathsf{G}[v]$ (lines 35 and 63).

Definition 5.10 *Given a union node v, $\mathsf{H}[v]$ is* well-valued *if it is a heap storing all*

Figure 5.3: The outline of the negative algorithm

Preconditions:
1. Invariants 1, 2, and 3 hold.
2. Feature variables of nodes in $Q[v]$ are well-valued.

Changing Variables: Feature variables of nodes in $Q[v]$, Φ, and $\mathsf{e_s}$.

Procedure Visit$_L^-$(node v);
26 **switch** *type of* v **do**
27 **case** *leaf node:* Gallop(v);
case *intersection node:*
28 **for** *all children* u *of* v **do** Visit$^-(u)$;
29 $\mathsf{Counter}[v] := 0$;
30 **for** *all children* u *of* v **do**
31 **if** $\mu(W[u]) = E$ **then** $\mathsf{Counter}[v] = \mathsf{Counter}[v] + 1$;
} Block 8
32 **if** $Counter[v] = d(v)$ *and* $E = e$ /* $d(v)$is the number of children of v.*/ **then** $\mathsf{W}[v] := \mathsf{W}[u]$;
33 **else** $\mathsf{W}[v] :=$ BEG;
} Block 9
34 Rebuild $\mathsf{H}[v]$ and $\mathsf{G}[v]$ with keys $\mu(\mathsf{W}[u])$ and $-\mathsf{M}[u]$ for children u of v;
35 $\mathsf{M}[v] = \mathsf{M}[\mathsf{G}[v].\text{top}]$;
case *union node:*
36 $\mathsf{Credit}[v] := \mathsf{Credit}[v] + L^-(v)$;
37 Find child u of v with the minimum key in $\mathsf{H}[v]$;
38 **while** $L^-(u) \geq Credit[v]$ **do**
39 Visit$_L^-(u)$;
40 $\mathsf{Credit}[v] := \mathsf{Credit}[v] - L^-(u)$;
} Block 10
41 **if** $\mu(W[u]) = E = e$ **then** $\mathsf{W}[v] := \mathsf{W}[u]$;
42 **if** u *is in* $N(Q)$ *and* $KeyOf(H[v], u) \neq M[u]$ **then** Update the key of u in $\mathsf{H}[v]$ as $\mathsf{M}[u]$;
43 $\mathsf{M}[v] := \mathsf{M}[\mathsf{H}[v].\text{top}]$;
44 Find child u of v with the minimum key in $\mathsf{H}[v]$;
} Block 11

Postconditions: Conditions listed in "Precondition" part hold.

Figure 5.4: The outline of the positive algorithm

Preconditions and Changing Variables: The same as those in Figure 5.3.

```
Procedure Visit_L^+(node v);
45  switch type of v do
46      case leaf node: Gallop(v);
        case union node:
47          for all children u of v do Visit^+(u);
            ------------------------------------------------------
48          W[v] := BEG;                                              } Block 12
            for all children u of v do
49              if μ(W[u]) = E = e then W[v] := W[u];                  } Block 12
            ------------------------------------------------------
50          Rebuild H[v] with keys M[u] for children u of v;
51          M[v] := M[H[v].top];
        case intersection node:
52          Credit[v] := Credit[v] + L^+(v);
53          Find child u of v with the minimum key in H[v];
54          while L^+(u) ≥ Credit[v] do
                ------------------------------------------------------
55              if μ(W[u]) = E then Counter[v] = Counter[v] − 1;
                ------------------------------------
56              Visit_L^+(u);                                         } Block 13
57              Credit[v] := Credit[v] − L^+(u);                      } Block 13
                ------------------------------------
58              if μ(W[u]) = E then Counter[v] = Counter[v] + 1;
                ------------------------------------
59              if Counter[v] = d(v) and E = e then                   } Block 14
60                  W[v] := W[u];                                     } Block 14
                else W[v] := BEG;                                     } Block 14
                ------------------------------------
61              if KeyOf(H[v], u) is not up-to-date then              } Block 15
                    Update the key of u in H[v] as μ(W[v]);           } Block 15
62              if u is in N(Q) and KeyOf(G[v], u) ≠ −M[u] then       } Block 15
                    Update the key of u in G[v] as −M[u];             } Block 15
63              M[v] = M[G[v].top];                                   } Block 15
64              Find child u of v with the minimum key in H[v];       } Block 15
                ------------------------------------------------------
```

Postconditions: The same as those in Figure 5.3.

children of v in $N(Q)$ and the key of every child u in $\mathsf{H}[v]$ is $\mathsf{M}[u]$.

Definition 5.11 *Given a child u of an intersection node v,* the key of u in $\mathsf{H}[v]$ is up-to-date *if $\mathsf{e} \neq \mathsf{E}$, $\mathsf{e} \in \{-\infty, \infty\}$, or the following condition holds: if $\mu(\mathsf{W}[u]) = \mathsf{e}$ the key of u in $\mathsf{H}[v]$ is e; otherwise the key of u in $\mathsf{H}[v]$ is a value less than e. Given an intersection node v, $\mathsf{H}[v]$ is* well-valued *if H is a heap storing all children of v and the key of every child u of v in $\mathsf{H}[v]$ is up-to-date.*

Definition 5.12 *Given an intersection node v, $\mathsf{G}[v]$ is* well-valued *if it is a heap storing all children v and the key of every child u in $\mathsf{G}[v]$ is $-\mathsf{M}[u]$.*

For easier reference in the future, we define all variables defined for a node v so far as feature variables of v in the next definition.

Definition 5.13 *Given a node v of Q,* feature variables of v *are $\mathsf{M}[v]$ (if v is in $N(Q)$), $\mathsf{W}[v]$, $\mathsf{Counter}[v]$ (if v is an intersection node), $\mathsf{Step}[v]$ (if v is a leaf), $\mathsf{H}[v]$ (if v is an internal node), $\mathsf{G}[v]$ (if v is an intersection node), $\mathsf{Credit}[v]$ (if v is an internal node), and $\mathit{fi}[v]$ (if v is a leaf).*

We now define the concept of being well-valued for variables for which this concept is not defined so far, as follows.

Definition 5.14 *Aside from elements of the arrays M, W, $\mathsf{Counter}$, H, and G and the variable E, all variables of the algorithm are always well-valued.*

Before proving that after visiting a node the postconditions stated in Figure 5.3 hold, we explain what kind of changes to contents of variables may cause feature variables of a node not to be well-valued anymore. In Table 5.1, for each array or variable v we have listed the arrays and variables whose change may stop v (or the elements of v in the case of arrays) stop being well-valued.

Lemma 5.40 *Given a node v of Q, the definition of being well-valued for every feature value of a node v depends only on contents of feature variables of v, on contents of feature variables of children of v, on e, and on E.*

The variable or array	Being well-valued depends on these variables and arrays
M	M and fi
W	W, fi, e, and E
Counter	W, Counter, and E
H	M, W, H, e, and E
G	M and G
E	e and E
The rest	Always well-valued

Table 5.1: Dependency list of definitions of elements of arrays and variables.

Proof. The correctness of the lemma for variables $\mathsf{M}[v]$, $\mathsf{W}[v]$, $\mathsf{Counter}[v]$, $\mathsf{Step}[v]$, $\mathsf{H}[v]$, $\mathsf{G}[v]$, $\mathsf{Credit}[v]$, and $\mathrm{fi}[v]$ follows immediately from Definitions 5.6, 5.8, 5.9, 5.14, 5.10 and 5.11, 5.12, 5.14, and 5.14, respectively. □

The next observation follows immediately from the previous lemma.

Observation 5.41 *Consider a node v and a part of the algorithm that does not change any feature variable of v or any child of v and also does not change any of variables* e *or* E*. Every feature variable of v that before executing this part was well-valued is well-valued after the execution of this part.*

The next observation, following immediately from the definitions of Invariants 1 and 3 and Observation 5.2, provides a result similar to Lemma 5.40 for the three invariants.

Observation 5.42 *The correctness of Invariants 1, 2, and 3 only depends on elements of the array fi and on* C*.*

In the rest of the section we show that changing variables and postconditions of the procedures Visit$^-$ and Visit$^+$ are chosen wisely. We first prove that the procedures Visit$^-$ and Visit$^+$ do not modify any variable but their changing variables. After that we show that after a legal call to one of these two procedures, the postconditions listed in Figure 5.3 hold.

Lemma 5.43 *Given a node v, by calling Visit$^-(v)$ or Visit$^+(v)$ the algorithm does not change any variable other than the changing variables of these procedures listed in Figure 5.3.*

Proof. We use induction on the height of v to prove the lemma. First we prove the lemma for the base case in which v is a leaf. When v is a leaf, each of the procedures Visit$^-(v)$ and Visit$^+(v)$ calls Gallop(v) (lines 27 and 46) and then exits. By Observation 5.37 during the execution of Gallop(v), no variable other than $\mathsf{fi}[v]$, $\mathsf{W}[v]$, $\mathsf{M}[v]$ (when v is not a speedy leaf), $\mathsf{Step}[v]$, $\mathsf{e_s}$, and Φ changes. Hence, as v is a node of $Q[v]$ and all variables listed above except $\mathsf{e_s}$ and Φ are feature variables of v, no variable other than feature variables of nodes in $Q[v]$ and $\mathsf{e_s}$ and Φ change. Thus, as feature variables of nodes in $Q[v]$ and $\mathsf{e_s}$ and Φ are changing variables, the lemma is true in this case.

Now we suppose v is an internal node and the lemma is true for children of v and we prove the lemma for v. No line in the procedure Visit$^-$ or Visit$^+$ exists that directly modifies a variable other than feature variables of v. Also, as v is in $Q[v]$, feature variables of v are changing variables. Moreover, as can be seen, every procedure executed directly by Visit$^-(v)$ or Visit$^+(v)$ is of the form Visit$^-(u)$ or Visit$^+(u)$, where u is a child of v and so by induction during the execution of this procedure call no variable other than feature variables of nodes in $Q[u]$, $\mathsf{e_s}$, and Φ are modified. As u is a child of v, nodes of $Q[u]$ are in $Q[v]$. Thus all variables modified are changing variables of the procedure. □

Now we prove the properties listed in the next definition for every node v of Q.

Definition 5.15 *A node v of Q is* well-behaved *if whenever the algorithm makes a legal call to Visit$^-(v)$ or Visit$^+(v)$ the following properties hold:*

1. *After executing the procedure postconditions described in Figure 5.3 hold.*
2. *Every procedure call made during the execution of Visit$^-(v)$ or Visit$^+(v)$, whichever that is called, is legal.*

One reason that the second property has been added to the above definition is that we need to make sure that every call to the procedure Gallop is legal so that we can use Lemma 5.39. We will use induction on heights of the nodes of Q to show that every node

of Q is well-behaved. We first provide a proof for the base case of this induction. Since E and e are not changing variables of the two procedures and so by Lemma 5.43 their values throughout the running of the procedures do not change, we do not use time stamps for these variables in our arguments.

Lemma 5.44 *Every leaf of Q is well-behaved.*

Proof. Given a leaf l of Q we need to prove that if before calling Visit$^-(l)$ or Visit$^+(l)$ the preconditions described in Figure 5.3 hold, the two properties explained in Definition 5.15 are met. We prove these two properties in order, starting from the first one. As can be seen, Visit$^-(l)$ (Visit$^+(v)$, respectively) calls Gallop(l) in line 27 (in line 46) and then exits. Since by precondition 1 before starting the execution of the procedure, Invariants 1, 2, and 3 hold, when calling Gallop(v) this condition sill holds and hence the precondition of Gallop(v), before calling it, is satisfied. Therefore, by Lemma 5.38 after executing Gallop(v) Invariants 1, 2, and 3 hold and $\mathsf{W}[v]$ and $\mathsf{M}[v]$ are well-valued. Furthermore, $\mathsf{W}[v]$, $\mathsf{M}[v]$, $\mathsf{Step}[v]$, and $\mathrm{fi}[v]$ are all feature variables of v. So as after exiting Gallop the algorithm exits Visit$^-(v)$ (Visit$^+(v)$, respectively), after exiting Visit$^-(v)$ (Visit$^+(v)$) the three invariants hold and feature variables of the only node of $Q[v]$ are well-valued.

Now we prove the second property of Definition 5.39. Visit$^-(v)$ (Visit$^+(v)$, respectively) calls directly just one procedure, that is Gallop(v), and inside the procedure Gallop there is no procedure call. Also we proved above that when calling Gallop(v) in Visit$^-(v)$ (in Visit$^+(v)$), preconditions of the procedure Gallop hold. Thus every procedure call made during the execution of Visit$^-(v)$ (Visit$^+(v)$, respectively) is legal. So, v is well-behaved as both conditions of Definition 5.15 are proved to hold for v. □

Next we prove that if all children of an internal node v of Q are well-behaved then v is also behaved. We present the proof in a few steps. We first prove that after a legal call to Visit$^-(l)$ or Visit$^+(v)$, Invariants 1, 2, and 3 hold and feature variables of all nodes of $Q[v]$ except perhaps v are well-valued (Lemma 5.45). Then we prove the correctness of the second condition of Definition 5.15 for v (Lemma 5.46). Finally we complete the proof of the first condition of Definition 5.15 for v by showing that feature variables of v also become well-valued when exiting the procedure (Lemmas 5.48, 5.49, 5.51, and 5.52).

Lemma 5.45 *Suppose v is an internal node of Q such that all children of v are well-behaved and suppose the algorithm makes a legal call to Visit$^-(l)$ (Visit$^+(v)$, respectively). Then after executing each line of the procedure, Invariants 1, 2, and 3 hold and all feature variables of all nodes of $Q[v]$ except perhaps v are well-valued.*

Proof. We consider a line ℓ of Visit$^-(v)$ (Visit$^+(v)$, respectively) and we suppose before executing line ℓ Invariants 1, 2, and 3 hold and feature variables of nodes of $Q[v]$ except perhaps v are well-valued. We prove that after executing line ℓ these properties still hold. We consider two cases based on line ℓ being a procedure call or not.

First consider the case in which ℓ is a procedure call. As can be seen in Figure 5.3 (in Figure 5.4), every procedure calling executed directly by Visit$^-(v)$ (by Visit$^+(u)$) is of the form Visit$^-(u)$ (from Visit$^+(u)$), where u is a child of v. Since nodes of $Q[u]$ are in $Q[v]$ and are not equal to v, when executing line ℓ, by assumption feature variables of nodes of $Q[u]$ ar well-valued. Also, by assumption Invariants 1, 2, and 3 hold. Hence, when calling Visit$^-(u)$ (Visit$^+(u)$, respectively) in line ℓ, all its preconditions hold and thus the procedure call is legal. So, as u is a child of v and hence by assumption u is well-behaved, by Definition 5.15 after executing line ℓ, postconditions of Visit$^-(u)$ (of Visit$^+(u)$) hold. Therefore, after line ℓ, Invariants 1, 2, and 3 hold.

Now we consider a node w of $Q[v]$ other than v and we prove that feature variables of w are well-valued after exiting Visit$^-(u)$ (Visit$^+(u)$, respectively). We consider two cases based on w being in $Q[u]$ or not. First suppose w is not in $Q[u]$. Then, no child of w is in $Q[u]$. So, as according to Lemma 5.43 by executing Visit$^-(u)$ (Visit$^+(u)$, respectively) only feature variables of nodes of $Q[u]$, Φ, and $\mathsf{e_s}$ change, feature variables of w, feature variables of children of w, E, and e remain unchanged. Also, before executing Visit$^-(u)$ (Visit$^+(u)$, respectively), feature variables of w are well-valued, as w is in $Q[v]$ and $w \neq v$. So, after executing Visit$^-(u)$ (Visit$^+(u)$, respectively) by Observation 5.41 feature variables of w remain well-valued. Now, consider the other case, in which w is in $Q[u]$. We proved above that after exiting Visit$^-(u)$ (Visit$^+(u)$, respectively) all postconditions of Visit$^-(u)$ (Visit$^+(u)$, respectively) hold. Therefore, as w is in $Q[u]$, by the second postcondition of Visit$^-(u)$ (of Visit$^+(u)$), feature variables of w are well-valued after executing line ℓ. So, after the execution of line ℓ feature variables of all nodes of $Q[v]$ except perhaps v are well-valued. Hence, when line ℓ is a procedure call, after executing line ℓ both claims are

proved.

Now we consider the case in which ℓ is not a procedure call. As can be seen in Figure 5.3 (in Figure 5.4) non-procedure call lines do not change C nor elements of the array fi. Thus, since by Observation 5.42 the correctness of Invariants 1, 2, and 3 depend only on variables C and fi, after executing line ℓ the three invariants hold as by assumption before executing line ℓ they hold. Also, every variable changed by a non-procedure call line of Figure 5.3 (of Figure 5.4) is a feature variable of v. So, given a node w of $Q[v]$ other than v, feature variables of w and children of w, C, and E remain unchanged. Hence by Observation 5.41 after executing this line feature variables of w remain well-valued as by assumption before executing this line they were well-valued. Thus, in both cases the lemma is true. □

Next we prove the second condition of Definition 5.15 for an internal node all of whose children are well-behaved.

Lemma 5.46 *Consider an internal node v such that every child of v is well-behaved. Then, every procedure call made during the execution of a legal procedure call Visit$^-(v)$ or Visit$^+(v)$ is legal.*

Proof. We first prove that every procedure call made directly by Visit$^-(v)$ or Visit$^+(v)$ is legal and then we use the induction hypothesis to prove the lemma. As can be seen in Figures 5.3 and 5.4, every procedure called directly by Visit$^-(v)$ or Visit$^+(v)$ is Visit$^-(u)$ or Visit$^+(u)$, for a child u of v. It follows from Lemma 5.45 that before executing every line Invariants 1, 2, and 3 hold and all feature variables of nodes of $Q[v]$ except perhaps v are well-valued. Thus, before every procedure call Visit$^-(u)$ or Visit$^+(u)$, for a child u of v, the preconditions of these procedure calls hold and so the procedure call is legal. Also, as by assumption all children of v are well-behaved, by Definition 5.15 during the execution of a legal call to Visit$^-(u)$ or Visit$^+(u)$, for a child u of v, all procedure calls are legal. Hence, all procedure calls during the execution of Visit$^-(v)$ or Visit$^+(v)$ are legal. □

Now it remains to prove that after a legal call to Visit$^-(v)$ or Visit$^+(v)$ all feature variables of v are well-valued. Since Block 9 of Figure 5.3 is the same as Block 14 of Figure 5.4, we first prove that this block, in whichever procedure that appears, causes the variable $\mathsf{W}[v]$, for an intersection node v, to become well-valued.

Lemma 5.47 *If Block 9 of Figure 5.3 appears in any part of the algorithm and before executing that instance of the block, v is an intersection node and* Counter$[v]$ *is well-valued, after executing this block* W$[v]$ *is well-valued.*

Proof. We prove the lemma by considering two cases depending on whether the condition checked on line 32 is true or false. Also, we will suppose $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$ as otherwise $\mathsf{W}[v]$ is well-valued, whatever its value is. First suppose the condition checked on line 32 is true. Then, $\mathsf{Counter}[v]$ equals the number of children u of v. Also, as by assumption $\mathsf{Counter}[v]$ is well-valued, by Definition 5.9, $\mathsf{Counter}[v]$ is the number of children u of v with $\mu(\mathsf{W}[u]) = \mathsf{E}$. Hence for all children u of v, $\mu(\mathsf{W}[u]) = \mathsf{E} = \mathsf{e}$. Moreover, since by assumption the condition of the "if" statement is true, the statement $\mathsf{W}[v] := \mathsf{W}[u]$, for a child u of v is executed. So as u is a child of v, $\mu(\mathsf{W}[u]) = \mathsf{e}$. Consequently, $\mathsf{W}[v]$ is set equal to an object of value e. Therefore, after running this line $\mathsf{W}[v]$ and $\mathsf{W}[w]$, for all children w of v, are objects of value e and thus $\mathsf{W}[v]$ satisfies all four conditions listed in Definition 5.8. Hence, $\mathsf{W}[v]$ got well-valued.

Now, consider the other case, in which the condition checked on line 32 is not true. Then, line 33 is executed and so $\mathsf{W}[v]$ is set equal to BEG. After running this line $\mu(\mathsf{W}[v]) = \mu(\text{BEG}) = -\infty < \mathsf{e}$ (as by assumption $\mathsf{e} \neq -\infty$) and hence the first condition of Definition 5.8 is satisfied. Also since by assumption the condition of line 32 is false and we suppose $\mathsf{e} = \mathsf{E}$, the statement $\mathsf{Counter}[v] = d(v)$ is false. Therefore, since as explained $\mathsf{Counter}[v]$ shows the number of children w of v with $\mu(\mathsf{W}[w]) = \mathsf{E}$, there is a child w of v with $\mu(\mathsf{W}[u]) \neq \mathsf{E} = \mathsf{e}$. Hence as after executing this line $\mu(\mathsf{W}[v]) = \mu(\text{BEG}) < \mathsf{e}$ (proved above), the fourth condition of Definition 5.8 is satisfied by $\mathsf{W}[v]$. Furthermore, the second and the third conditions of Definition 5.8 are met because by assumption v is an intersection node. Thus $\mathsf{W}[v]$ becomes well-valued. □

We now consider four cases based on v being a union node or an intersection node and also on the call being to Visit$^-(v)$ or to Visit$^+(v)$ and in each case we prove that after visiting v all feature variables of v are well-valued.

Lemma 5.48 *Suppose v is an intersection node of Q such that all children of v are well-behaved and suppose the algorithm makes a legal call to Visit$^-(v)$. Then after executing the procedure all feature variables of v are well-valued.*

Proof. We first prove that after executing the loop block of line 30, $\mathsf{Counter}[v]$ is well-valued, then we show that lines 32 to 35 cause the rest of feature variables of v become well-valued. As Figure 5.3 shows, in the "for" block of line 28 $\text{Visit}^-(u)$ is called sequentially, for all children u of v. Then in Block 8 the number of children u of v satisfying $\mu(\mathsf{W}[u]) = \mathsf{E}$ is evaluated and saved in the variable $\mathsf{Counter}[v]$. As a result, by Definition 5.9 when reaching Block 9 $\mathsf{Counter}[v]$ is well-valued. Consequently, by Lemma 5.47 after executing this Block, $\mathsf{W}[v]$ becomes well-values. After that, by executing line 34 the two heaps H and G are created from scratch and become well-valued. So, after running this line, by Definition 5.12 the top element of the heap $\mathsf{G}[v]$ is a child w of v such that $-\mathsf{M}[w] = \min_u -\mathsf{M}[u] = -\max_u \mathsf{M}[u]$ ad thus $\mathsf{M}[w] = \max_u \mathsf{M}[u]$ where the minimum and the maximum are taken over all children u of v in $N(Q)$. Therefore, by executing line 35 $\mathsf{M}[v]$ is set to the maximum of $\mathsf{M}[u]$ over all children u of v in $N(Q)$ and hence by Definition 5.6 it becomes well-valued. So, as by Definition 5.14 the feature variables of v that we have not discussed here are always well-valued, all feature variables of v are well-valued when the procedure exits. □

Lemma 5.49 *Suppose v is a union node of Q such that all children of v are well-behaved and suppose the algorithm makes a legal call to $\text{Visit}^-(v)$. Then after each execution of Block 11 of $\text{Visit}^-(v)$, all feature variables of v are well-valued.*

Proof. According to Figure 5.3, when v is a union node, repeatedly, each time a child u of v is selected (lines 37 and 44), Block 11 is executed. The only line executed by $\text{Visit}^-(v)$ before starting Block 11 only changes $\mathsf{Credit}[v]$. Also, by assumption the call to $\text{Visit}^-(v)$ is legal and so before starting the procedure by assumption all feature variables of v are well-valued. Hence, as the definition of being well-valued for no variable depends on the value of $\mathsf{Credit}[v]$, when starting the while loop all feature variables of v are well-valued. We now prove that after each execution of the body of the "while" loop, all feature variables of v remain well-valued. The first line of Block 11, line 39, executes $\text{Visit}^-(u)$ where u is a child of v. Suppose *bef* and *aft* are the times just before and just after executing line 39.

We first prove that $\mathsf{W}[v]$ becomes well-valued and then we discuss other feature variables of v. We consider several cases based on contents of $\mathsf{W}^{(bef)}[v]$ and E. If $\mathsf{E} \neq \mathsf{e}$ or $\mathsf{e} \in \{-\infty, \infty\}$, by Definition 5.8, $\mathsf{W}[v]$ is always well-valued. So suppose $\mathsf{E} = \mathsf{e}$ and $\mathsf{e} \notin \{-\infty, \infty\}$. Then, since, as explained, before executing line 39 feature values of v were

well-valued, by Definition 5.8 $\mu(\mathsf{W}^{(bef)}[v]) = \mathsf{e}$ if and only if there is a child w of v with $\mu(\mathsf{W}^{(bef)}[w]) = \mathsf{e}$. Now we consider the three following cases: First, when $\mu(\mathsf{W}^{(bef)}[v]) = \mu(\mathsf{W}^{(bef)}[u]) = \mathsf{e}$. Second, when $\mu(\mathsf{W}^{(bef)}[v]) = \mu(\mathsf{W}^{(bef)}[w]) = \mathsf{e}$, for a child w of v other than u. Third, when neither of the above cases occurs and thus $\mu(\mathsf{W}^{(bef)}[v]) \neq \mathsf{e}$.

Case 1: $\mu(\mathbf{W}^{(\boldsymbol{bef})}[v]) = \mu(\mathbf{W}^{(\boldsymbol{bef})}[u]) = \mathbf{e}$.
In this case we first prove that before executing Visit$^-(u)$, u has an eyewitness T. Then we prove that after the execution of Visit$^-(u)$ T is still an eyewitness for u and we conclude that $\mathsf{W}^{(aft)}[u] = \mathsf{e}$. Finally we show that the condition checked on line 41 is true and thus line 41 causes $\mathsf{W}[v]$ to become well-valued.

Now we present the first part of the argument. Since u is a node of $Q[v]$ and $u \neq v$, by Lemma 5.45 before execution line 39, $\mathsf{W}[u]$ was well-valued. Also, as by assumption $\mu(\mathsf{W}^{(bef)}[u]) = \mathsf{e}$, $\mathsf{e} = \mathsf{E}$, and $\mathsf{e} \notin \{-\infty, \infty\}$, by Lemma 5.15 before calling Visit$^-(u)$ there was an eyewitness T for u.

Claim 5.50 *After executing Visit$^-(u)$ (line 39), T is still an eyewitness for u.*

Proof. We suppose the claim is not true and we consider the sequence of lines of the algorithm executed when calling Visit$^-(u)$ and the first line ℓ in this sequence such that after executing line ℓ, T is not an eyewitness of v. We consider two cases based on which procedure line ℓ belongs to. The only procedures called directly by Visit$^-$ are Visit$^-$ and Gallop and Gallop does not call any procedure. Thus ℓ is in Visit$^-$ or Gallop. Also, the definition of being an eyewitness (Definition 5.5) depends only on variables e and the array fi and there is no line in the procedure Visit$^-$ changing fi or e. Hence ℓ is a line of the procedure Gallop. So, once during the execution of Visit$^-(u)$ the procedure Gallop is called (not necessarily directly by Visit$^-(u)$) and before running Gallop, T was an eyewitness while after running it T stopped being an eyewitness. Thus, due to Lemma 5.39, before executing Gallop, its preconditions did not hold and hence calling Gallop was not legal. So during executing a legal call to Visit$^-(u)$ a non-legal call occurred and thus by Definition 5.15, u is not well-behaved. This contradicts the assumption of the lemma. □

Now we prove that $\mu(\mathsf{W}^{(aft)}[u]) = \mathsf{e}$ and we conclude that in line 41 v becomes well-valued. Since u is in $Q[v]$ and $u \neq v$, by Lemma 5.45 after executing line 39 feature variables of the node u are well-valued. Thus, as by assumption $\mathsf{E} = \mathsf{e}$ and $\mathsf{e} \notin \{-\infty, \infty\}$ and we proved that T is still an eyewitness for u, by Lemma 5.15 $\mu(\mathsf{W}^{(aft)}[u]) = \mathsf{e}$. So, since by assumption $\mathsf{E} = \mathsf{e}$, the condition checked on line 41 is true and thus $\mathsf{W}[v]$ is set equal to $\mathsf{W}^{(aft)}[u]$. As we proved $\mu(\mathsf{W}^{(aft)}[u]) = \mathsf{e}$, after executing this line, $\mu(\mathsf{W}^{(aft)}[v]) = \mu(\mathsf{W}^{(aft)}[u]) = \mathsf{e}$ and thus $\mathsf{W}[v]$ satisfies all four conditions of Definition 5.8. Hence by Definition 5.8 after executing this assignment $\mathsf{W}[v]$ becomes well-valued.

Case 2: $\mu(\mathbf{W}^{(bef)}[v]) = \mu(\mathbf{W}^{(bef)}[w]) = \mathsf{e}$, for a child w of v other than u.
As w and v are not in $Q[u]$ and so by Lemma 5.43 Visit$^-(u)$ does not modify $\mathsf{W}[v]$ nor $\mathsf{W}[w]$, $\mu(\mathsf{W}^{(aft)}[v]) = \mu(\mathsf{W}^{(aft)}[w]) = \mathsf{e}$. Also, if the condition checked on line 41 is true, $\mu(\mathsf{W}^{(aft)}[u]) = \mathsf{e}$ and $\mathsf{W}[v]$ is set equal to an object of value e. So, whether the condition of line 41 is true or false, after line 41 $\mathsf{W}[v]$ is an object of value e. So, as we proved $\mu(\mathsf{W}^{(aft)}[w]) = \mathsf{e}$, by Definition 5.8 after line 41 $\mathsf{W}[v]$ is well-valued because it satisfies all four conditions of Definition 5.8.

Case 3: $\mu(\mathbf{W}^{(bef)}[v]) \neq \mathsf{e}$.
We consider two cases based on the condition checked on line 41 being true or false. If that condition is true, $\mu(\mathsf{W}^{(aft)}[u]) = \mathsf{e}$ and $\mathsf{W}[v]$ is set equal to an object of value e. Hence, after this assignment, $\mathsf{W}[v]$ and $\mathsf{W}[u]$ are both objects of value e and thus $\mathsf{W}[v]$ meets all conditions of Definition 5.8.

Now consider the case in which the condition checked on line 41 is false. Since by assumption $\mu(\mathsf{W}^{(bef)}[v]) \neq \mathsf{e}$ and before executing Visit$^-(u)$ $\mathsf{W}[v]$ was well-valued, $\mu(\mathsf{W}^{(bef)}[w]) \neq \mathsf{e}$, for every child w of v. As other than u, no child of v is in $Q[u]$, Visit$^-(u)$ does not modify $\mathsf{W}[w]$, for every child w of v, $w \neq u$. Hence, $\mu(\mathsf{W}^{(aft)}[w]) \neq \mathsf{e}$, for every child w of v other than u. Similarly, as v is not in $Q[u]$, $\mu(\mathsf{W}^{(aft)}[v]) = \mu(\mathsf{W}^{(bef)}[v]) \neq \mathsf{e}$. Also as by assumption the condition checked on line 41 is false, $\mathsf{W}^{(aft)}[u]$ is not an object of value e. Hence, $\mu(\mathsf{W}^{(aft)}[w]) \neq \mathsf{e}$, for every child w of u. Furthermore as the assignment on line 41 is not executed, $\mathsf{W}[v]$ is not modified and so after this line still we have $\mu(\mathsf{W}[v]) = \mu(\mathsf{W}^{(aft)}[v]) = \mu(\mathsf{W}^{(bef)}[v]) \neq \mathsf{e}$. Also, as by assumption at the time *bef* $\mathsf{W}[v]$ was well-valued and the inequalities $\mu(\mathsf{W}^{(bef)}[v]) \neq \mathsf{e}$, $\mathsf{E} = \mathsf{e}$, and $\mathsf{e} \notin \{-\infty, \infty\}$ held, by

Definition 5.8 $\mu(\mathsf{W}^{(bef)}[v]) < \mathsf{e}$. Consequently, after line 41 $\mu(\mathsf{W}[v]) < \mathsf{e}$. Thus, after line 41 $\mathsf{W}[v]$ satisfies the four conditions of Definition 5.8 and so $\mathsf{W}[v]$ is well-valued. Hence in both cases after line 41 $\mathsf{W}[v]$ is well-valued.

Next, we prove that $\mathsf{H}[v]$ becomes well valued and then we conclude that $\mathsf{M}[v]$ also becomes well-valued. For every child w of v in $N(Q)$ other than u, as w is not in $Q[u]$, by Lemma 5.43 $\mathsf{M}[w]$ is not modified by Visit$^-(u)$ and thus $\mathsf{M}^{(bef)}[w] = \mathsf{M}^{(aft)}[w]$. Also, since by assumption before executing Block 11 feature variables of v were well-valued, by Definition 5.10 before executing Visit$^-(u)$ the key of w in $\mathsf{H}[v]$ was $\mathsf{M}^{(bef)}[w]$ and Visit$^-(u)$ does not change $\mathsf{H}[v]$ as v is not in $Q[u]$. Therefore, after executing Visit$^-(u)$ the key of w in $\mathsf{H}[v]$ is $\mathsf{M}^{(bef)}[w] = \mathsf{M}^{(aft)}[w]$. Furthermore, by executing line 42 if u is in $N(Q)$, the key of u in $\mathsf{H}[v]$ is set to $\mathsf{M}^{(aft)}[u]$. Hence, after line 42 the key of every child w of v in $N(Q)$ in $\mathsf{H}[v]$ is $\mathsf{M}^{(aft)}[w]$ and thus by Definition 5.10 $\mathsf{H}[v]$ becomes well-valued. So by definition the top element of $\mathsf{H}[v]$ is the child w of v in $N(Q)$ minimizing $\mathsf{M}[w]$. Therefore, line 43 sets $\mathsf{M}[v]$ equal to minimum of $\mathsf{M}[w]$ over all children of w. So by Definition 5.6, $\mathsf{M}[v]$ becomes well-valued. Now we have proved that $\mathsf{M}[v]$, $\mathsf{W}[v]$, and $\mathsf{H}[v]$ become well-valued after executing Block 11 and since v is a union node, $\mathsf{Counter}[v]$ and $\mathsf{G}[v]$ are not defined. Also, other feature variables of v are well-valued by Definition 5.14. So the lemma is correct. □

Lemma 5.51 *Suppose v is a union node of Q such that all children of v are well-behaved and suppose the algorithm makes a legal call to Visit$^+(v)$. Then after executing the procedure all feature variables of v are well-valued.*

Proof. We first prove that by executing the loop block of line 47, $\mathsf{W}[v]$ becomes well-valued and then we show that the next lines cause the rest of feature variables of v to become well-valued. To prove that $\mathsf{W}[v]$ becomes well-valued we suppose $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$ as otherwise $\mathsf{W}[v]$ is always well-valued. As Figure 5.4 shows, in the "for" block of line 47 for all children u of v, Visit$^+(u)$ is called sequentially. Then in Block 12 first $\mathsf{W}[v]$ is set equal to BEG, then if there is any child u of v satisfying $\mu(\mathsf{W}[u]) = \mathsf{E} = \mathsf{e}$ the statement $\mathsf{W}[v] := \mathsf{W}[u]$ is executed. Hence, as by assumption $\mathsf{e} = \mathsf{E}$, after executing this block, if $\mu(\mathsf{W}[u]) = \mathsf{e}$, for a child u of v, then $\mathsf{W}[v]$ is an object of value e; otherwise $\mathsf{W}[v] =$ BEG

and so $\mu(\mathsf{W}[v]) = -\infty < \mathsf{e}$ as by assumption $\mathsf{e} \neq -\infty$. As a result, all conditions of Definition 5.8 are satisfied and thus $\mathsf{W}[v]$ becomes well-valued. Also by executing line 50 the heap H is created from scratch and so it becomes well-valued. Therefore, by Definition 5.10 the top element of the heap $\mathsf{H}[v]$ is the minimum of $\mathsf{M}[u]$ over all children u of v in $N(Q)$. Thus, after executing line 51 by Definition 5.6 $\mathsf{M}[v]$ is well-valued. Therefore, as by Definition 5.14 feature variables of v other than $\mathsf{M}[v]$, $\mathsf{W}[v]$ and $\mathsf{H}[v]$ (and $\mathsf{Counter}[v]$ and $\mathsf{G}[v]$ as they are not defined) are always well-valued, all feature variables of v become well-valued. □

Lemma 5.52 *Suppose v is an intersection node of Q such that all children of v are well-behaved and suppose the algorithm makes a legal call to Visit$^+(v)$. Then after each execution of Block 15 of Visit$^+(v)$, all feature variables of v are well-valued.*

Proof. According to Figure 5.4, in the "while" block of line 54 each time a child u of v is selected (lines 53 and 64), Block 15 is executed. Before starting the while loop by assumption all feature variables of v are well-valued. We now prove that after each execution of Block 15, all feature variables of v remain well-valued. We first prove our claim for $\mathsf{Counter}[v]$ and $\mathsf{W}[v]$. Then we consider variables $\mathsf{H}[v]$, $\mathsf{G}[v]$, and $\mathsf{M}[v]$, and finally we show the correctness of the lemma for the rest of feature variables of v.

Now we first discuss variables $\mathsf{Counter[v]}$ and $\mathsf{W}[v]$. Since by precondition 2 before executing line 55 $\mathsf{Counter[v]}$ is well-valued, by Definition 5.9 it was the number of children w of v including u with $\mu(\mathsf{W}[w]) = \mathsf{E}$. Line 55 checks if $\mathsf{W}[u]$ is an object of value E, and if so, it decreases $\mathsf{Counter}[v]$. After executing this line $\mathsf{Counter}[v]$ will store the number of children w of v other than u with $\mu(\mathsf{W}[w]) = \mathsf{E}$. Line 56 executes Visit$^+(u)$ where u is a child of v. After executing this line $\mathsf{Counter}[v]$ still shows the number of children w of v other than u with $\mu(\mathsf{W}[w]) = \mathsf{E}$ because, for every child w of v other than u, w and v are not in $Q[u]$ and so by Lemma 5.43 by executing Visit$^+(u)$, $\mathsf{W}[w]$ and $\mathsf{Counter}[v]$ do not change. Line 58 increases $\mathsf{Counter}[v]$ in the case $\mu(\mathsf{W}[u]) = \mathsf{E}$. Hence, after executing line 58 $\mathsf{Counter}[v]$ is the number of children w of v including u with $\mu(\mathsf{W}[w]) = \mathsf{E}$ and thus $\mathsf{Counter}[v]$ becomes well-valued, by Definition 5.8. So, since Block 14 is a copy of Block 9, by Lemma 5.47 after its execution $\mathsf{W}[v]$ becomes well-valued.

Next we prove that $\mathsf{H}[v]$ and $\mathsf{G}[v]$ are well valued and then we conclude that $\mathsf{M}[v]$ also becomes well-valued. Before executing Visit$^+(u)$, $\mathsf{H}[v]$ and $\mathsf{G}[v]$ were well-valued and so by Definition 5.11 the key of w in $\mathsf{H}[v]$ was up-do-date and by Definition 5.12 the key of w in $\mathsf{G}[v]$ was $-\mathsf{M}[w]$, for every child w of v. Moreover, Visit$^+(u)$ does not change $\mathsf{H}[v]$, $\mathsf{G}[v]$, $\mathsf{M}[w]$, nor $\mathsf{W}[w]$, for any child w of v other than u. Hence, before executing line 61, for every child w of v other than u, by Definition 5.11 the key of w in $\mathsf{H}[v]$ is still up-do-date (because $\mathsf{W}[w]$, e, and E are not modified) and the key of w in $\mathsf{G}[v]$ is $-\mathsf{M}[w]$. By executing line 61, if the key of u in $\mathsf{H}[v]$ is not up-do-date, it is set to $\mu(\mathsf{W}[v])$ and hence becomes up-do-date. Also, by executing line 61 the key of u in $\mathsf{G}[v]$ is set to $-\mathsf{M}[u]$. Thus, after line 62 the key of every child w of v in $\mathsf{H}[v]$ (in $\mathsf{G}[v]$) is up-to-date (is $-\mathsf{M}[w]$) and thus by Definitions 5.11 and 5.12 $\mathsf{H}[v]$ and $\mathsf{G}[v]$ are well-valued. So by definition the top element of $\mathsf{H}[v]$ is the child w of v with the minimizing $-\mathsf{M}[w]$. Thus, line 43 sets $\mathsf{M}[v]$ equal to $\mathsf{M}[u]$ where u is a node minimizing $-\mathsf{M}[w]$, that is the maximizing of $\mathsf{M}[w]$, over all children of v. So by Definition 5.6, $\mathsf{M}[v]$ becomes well-valued.

Now we have proved that $\mathsf{M}[v]$, $\mathsf{W}[v]$, $\mathsf{Counter}[v]$, $\mathsf{H}[v]$, and $\mathsf{G}[v]$ become well-valued. Also by Definition 5.14 other feature variables of v are always well-valued. Hence, the lemma is proved. □

In the next lemma we summarize the previous results and we show that all nodes are well-behaved.

Lemma 5.53 *All nodes of Q are well-behaved.*

Proof. We use induction on height of nodes v of Q to prove that v is well-behaved. The base case is when v is a leaf. In this case the induction claim is true by Lemma 5.44. Now we consider an internal node v whose all children are well-behaved and we show that v is also well-behaved. The correctness of the second condition of Definition 5.15 for v follows immediately from Lemma 5.46. To prove the first condition, we suppose the algorithm makes a legal call to Visit$^-(v)$ or Visit$^+(v)$ and we prove that after completing the execution of these procedures the postconditions of the procedures hold, that is, the three invariants hold and feature variables of nodes of $Q[v]$ are well-valued. It follows by Lemma 5.45 that since the call is legal, after execution the last line of Visit$^-(v)$ or Visit$^+(v)$, the three invariants hold and feature variables of nodes of $Q[v]$ except perhaps v are well-valued.

Moreover, if v is an intersection (a union) node and the procedure call is a negative-visiting (a positive-visiting), by Lemma 5.48 (Lemma 5.51, respectively), after finishing the execution of the procedure feature variables of v are well-valued. Also, if v is a union (an intersection) node and the procedure call is a negative-visiting (a positive-visiting), by Lemma 5.49 (Lemma 5.52, respectively), after last time execution of Block 11 (Block 15, respectively) feature variables of v are well-valued. Furthermore, after this last execution of Block 11 (Block 15, respectively), as is clear from Figure 5.3 (Figure 5.4, respectively), the execution of the procedure is ended. So, when the algorithm finishes visiting v, feature variables of v are also well-valued. Hence, the first condition of Definition 5.15 holds for v and thus v is well-behaved. Consequently, all nodes are well-behaved. □

5.4.3 Updating E

Recall from Definition 5.8 that contents of the array W are meaningful to the algorithm only if E = e. Because of this, we do not know any way to determine if there is any eyewitness for the root in constant time when E ≠ e. We have designed a procedure named UpdateE, shown in Figure 5.5, which sets E equal to e and makes necessary changes to other variables so that all variables become well-valued. We run UpdateE every once in a while so that if e changes, after a short delay, again W[root] can be used to know if there is an eyewitness for the root. The scheduling for executing this procedure will be such that at any time t, asymptotically the overall amount of the time taken by all executions of this procedure before time t will be the same as the overall amount of the time taken by other parts of the algorithm before time t. This will cause running UpdateE to increase the running time of the algorithm just by a constant factor and for UpdateE to be executed frequently enough so that whenever e is increased, after a short delay UpdateE is executed and E is set equal to e.

The procedure UpdateE is a recursive procedure with one parameter which is a node of the query tree. When UpdateE(v) is executed, for a node v, the procedure sets E := e and modifies the feature variables of nodes in $Q[v]$ so that they become well-valued. After setting E := e, the procedure calls itself recursively for all children of v, if v has any children. Next, if v is an intersection node, the procedure evaluates Counter[v] by counting

Figure 5.5: Updating E.

Preconditions: Invariants 1, 2, and 3 hold and all feature variables of nodes of $Q[v]$ except perhaps elements of the array W are well-valued.

Changing Variables:
1. The variable E and $\mathsf{W}[u]$, for all nodes u in $Q[v]$.
2. For intersection nodes u of $Q[v]$: Counter$[u]$ and $\mathsf{H}[u]$.

Procedure UpdateE(node v);
begin

```
65  | E := e;
66  | for every child u_i of v do UpdateE(u_i);
67  | switch type of v do
    |   case leaf
68  |   |  if μ(fi[l]) = e then W[l] := fi[l]; else W[l] := BEG;
    |   case intersection node
    |   |  Counter[v] := 0;                                          } Block 16
    |   |  for all children u of v do                                }
    |   |  |  if μ(W[u]) = E then Counter[v] = Counter[v] + 1;        }
    |   |
69  |   |  if Counter[v] = d(v) and E = e then                       } Block 17
    |   |  |  W[v] := W[u];                                          }
    |   |  else                                                      }
    |   |  |  W[v] := BEG;                                           }
70  |   |  Rebuild H[v];
    |   case union node
    |   |  W[v] := BEG;                                              } Block 18
    |   |  for all children u of v do                                }
    |   |  |  if μ(W[u]) = E = e then  W[v] := W[u];                  }
end
```

Postconditions:
1. E = e.
2. Invariants 1, 2, and 3 hold and all feature variables of nodes in $Q[v]$ are well-valued.

the number of children u of v with $\mu(\mathsf{W}[u]) = \mathsf{E}$. Afterward, depending on v being a leaf, an intersection node, or a union node, line 68, Blocks 16 and 17, or Block 18 is executed and, as we show in the next lemma, $\mathsf{W}[v]$ is evaluated so that it satisfies the four conditions of Definition 5.8. When v is an intersection node, since the definition of $\mathsf{H}[v]$ being well-valued depends on $\mathsf{W}[u]$, for children u of v, the algorithm rebuilds the heap $\mathsf{H}[v]$ from scratch (line 70) as well.

We now prove that UpdateE may modify only its changing variables and also that if the algorithm makes a legal call to UpdateE, after the execution of this procedure, its postconditions hold.

Lemma 5.54 *Given a node v of Q, UpdateE(v) does not modify any variable other than changing variables listed in Figure 5.5.*

Proof. We use induction on the height of v to prove the lemma. Suppose if v has any children then the lemma is true for all children of v. It is clear from the figure that when UpdateE(v) is called, the procedure directly changes only E, $\mathsf{W}[v]$, $\mathsf{Counter}[v]$ (if v is an intersection node) and $\mathsf{H}[v]$ (if v is an intersection node). The procedure also calls UpdateE(u), for children u of v, and hence in this recursion, by the assumption of the lemma only changing variables of UpdateE(u) may change. But, as the definition of changing variables of the procedure shows (Figure 5.5), given a child u of v, as nodes of $Q[u]$ are in $Q[v]$, changing variables of UpdateE(u) are also changing variables of UpdateE(v). So, by executing UpdateE(v) only changing variables of UpdateE(v) are modified. □

Lemma 5.55 *If before calling UpdateE(v), for a node v, the precondition of Figure 5.5 holds, after running it, the postconditions of the procedure listed in the figure hold.*

Proof. We use induction on the height of the node v to prove the lemma. First we prove the lemma for the base case, that is when v is a leaf, and next we discuss the induction step.

For the case of leaves, we first discuss how $\mathsf{W}[v]$ becomes well-valued and then we explain why the other postconditions hold after the execution of the procedure. When v is a leaf, first in line 65 E is set equal to e and then line 68 determines the content to

be assigned to $\mathsf{W}[v]$ exactly in the same way that line 22 does. Then, exactly the same argument that shows when Block 7 is executed line 22 causes the variable $\mathsf{W}[l]$ to become well-valued (the two paragraphs prior to the last paragraph in the proof of Lemma 5.38) proves that after executing line 65 $\mathsf{W}[v]$ becomes well-valued.

Now we prove the other claims of the postconditions when v is a leaf. Due to what we explained, the procedure just modifies $\mathsf{W}[v]$ and E. Also, by the precondition, before executing the procedure all feature variables of v except perhaps $\mathsf{W}[v]$ are well-valued. Hence, as the definition of being well-valued for feature variables of v except $\mathsf{W}[v]$ do not depend on $\mathsf{W}[v]$ and E, after executing the procedure the other feature variables of v are still well-valued. Furthermore, since the definitions of the three invariants do not depend on E and $\mathsf{W}[v]$ (Observation 5.42) and since by the precondition before executing the procedure these three hold, after the execution of the procedure they still hold. So the second postcondition holds. The correctness of the first postcondition follows immediately from the assignment on line 65.

Now we consider an internal node v and we suppose the lemma is true for every child of v. We first prove that when reaching line 67 certain properties hold. Then we consider two cases based on the type of the node v and we discuss the behavior of the procedure after line 67.

We now prove that right after calling Update$\mathsf{E}$ recursively for i children of v by line 66, for $0 \leq i \leq k$, the following properties hold. In the following, u_i is the ith child of v considered in line 66, for every i, $1 \leq i \leq k$.

1. The three invariants hold.
2. $\mathsf{E} = \mathsf{e}$.
3. All feature variables of nodes in $Q[u_j]$ are well-valued,

   for every j, $1 \leq j \leq i$.
4. All feature variables of nodes in $Q[u_j]$ except perhaps the elements of the array W are well-valued, for every j, $i < j \leq k$.
5. All feature variables of v except perhaps the following variables are well-valued: the variable $\mathsf{W}[v]$ and if v is an intersection node, the variables $\mathsf{H}[v]$ and $\mathsf{Counter}[v]$.

We use induction on i to prove the above claims. To prove the base case we prove that right after the execution of line 65 the above claims hold for $i = 0$. Line 65 just modifies E and since the correctness of no invariant or definition of being well-valued for any variable except elements of the array W depends on E, after executing this line the precondition still holds. Thus, claims 1, 3, 4, and 5 are true. The correctness of the second claim also follows immediately from the assignment executed in this line. So the five above claims are true for $i = 0$.

Next we suppose the claims are correct for $i-1$ and we prove that after the execution of Update$\mathsf{E}(u_i)$ the claims are true for i, $1 \leq i \leq k$. By the first and the third above claims for $i-1$, when calling Update$\mathsf{E}(u_i)$ its precondition holds and hence as by induction the lemma is correct for the children of v, after executing Update$\mathsf{E}(u_i)$ the postconditions of Update$\mathsf{E}(u_i)$ hold. Thus by the first postcondition of Update$\mathsf{E}(u_i)$ after calling Update$\mathsf{E}(u_i)$, $\mathsf{E} = \mathsf{e}$ and since by induction before calling Update$\mathsf{E}(u_i)$ the equality $\mathsf{E} = \mathsf{e}$ held, E is not modified by Update$\mathsf{E}(u_i)$. Also all changing variables of Update$\mathsf{E}(u_i)$ except E are feature variables of nodes in $Q[u_i]$. So by executing Update$\mathsf{E}(u_i)$ only feature variables of nodes in $Q[u_i]$ are modified.

Now we consider the five claims one by one and we prove them for i. The first two claims follow from the postconditions of Update$\mathsf{E}(u_i)$. Now consider the third claim. As mentioned, Update$\mathsf{E}(u_i)$ has modified only feature variables of nodes in $Q[u_i]$. Also, by the third condition for $i-1$ feature variables of nodes of $Q[u_j]$ are well-valued before calling Update$\mathsf{E}(u_i)$, for every j, $1 \leq j \leq i-1$. Hence they are will-valued still after calling Update$\mathsf{E}(u_i)$, according to Observation 5.41. Feature variables of nodes of $Q[u_i]$ are also well-valued by the postconditions of Update$\mathsf{E}(u_i)$. Thus, the third claim is true. The proof of the fourth claim is similar. By the fourth claim for $i-1$, apart from the elements of the array W, feature variables of nodes of $Q[u_j]$, $i-1 < j \leq k$, were well-valued before calling Update$\mathsf{E}(u_i)$. Also, Update$\mathsf{E}(u_i)$ has not modified feature variables of nodes of $Q[u_j]$ for j, $i < j \leq k$. So, by Observation 5.41, after calling Update$\mathsf{E}(u_i)$, except perhaps elements of the array W, feature variables of nodes of $Q[u_j]$, $i-1 < j \leq k$, are well-valued. Finally, consider the fifth claim. By the induction hypothesis, before calling Update$\mathsf{E}(u_i)$, feature variables of v except perhaps variables excluded in the statement of the fifth claim are well-valued. Moreover, and except $\mathsf{W}[v]$ and when v is an intersection node $\mathsf{H}[v]$ and

$\mathsf{Counter}[v]$, the definition of being well-valued for no feature variable of v depends on the changing variables of Update$\mathsf{E}(u_i)$. Therefore, after the execution of Update$\mathsf{E}(u_i)$ feature variables of v except perhaps those excluded in the claim five are well-valued. Hence, all five claims are correct for i.

Now let us explain which parts of postconditions are derived from the correctness of the above five claims for $i = k$. After executing line 66, Update$\mathsf{E}(v)$ does not call any procedure and only modifies feature variables of v. Hence, the correctness of the first postcondition follows from the second claim proved above for $i = k$. Moreover since by Observation 5.42 the correctness of no invariant depends on a feature variable of an internal node, due to the first above claim for $i = k$, when exiting the procedure Invariants 1, 2, and 3 hold. Furthermore, after executing line 66, due to the third claim for $k = i$, all feature variables of all nodes in $Q[v]$ except perhaps v are well-valued. So, by Observation 5.41 when exiting the procedure all these variables still are well-valued as the next lines of the procedure may modify only the feature variables of v. Consequently, it remains to prove that when exiting the procedure, feature variables of v are also well-valued. We consider two cases depending on whether v is a union node or an intersection node.

First suppose v is a union node. Then, similar to Block 12, in Block 18 first $\mathsf{W}[v]$ is set equal to END, then if there is any child u of v satisfying $\mu(\mathsf{W}[u]) = \mathsf{E} = \mathsf{e}$ the statement $\mathsf{W}[v] := \mathsf{W}[u]$ is executed. Hence, the same argument used in the proof of Lemma 5.51 shows that after the execution of Block 18 $\mathsf{W}[v]$ is well-valued. Other feature variables of v were well-valued before executing the procedure and their definition of being well-valued depends on the elements of arrays M, fi, and feature variables of v other than $\mathsf{W}[v]$. Thus, since these variables are not changing variables, after exiting the procedure they are well-valued. So the lemma is true in the case of union nodes.

Finally, consider the case in which v is an intersection node. In this case, in Block 16 the number of children u of v satisfying $\mu(\mathsf{W}[u]) = \mathsf{E}$ is evaluated and saved in the variable $\mathsf{Counter}[v]$. As a result, by Definition 5.7 when reaching Block 17 $\mathsf{Counter}[v]$ is well-valued. So, as Block 17 is a copy of Block 9, by Lemma 5.47 after the execution of this block $\mathsf{W}[v]$ becomes well-valued. Then, by executing line 70 the heap H is created from scratch and becomes well-valued. The same argument as the union case shows that the other feature variables remain well-valued since their definition of being well-valued does not depend on

changing variables of the procedure. Thus, the lemma is true in the case of intersection nodes, as well. □

5.5 The Main Algorithm

In this section we describe how different procedures we described before are combined. There are three procedures that the algorithm alternates between calling: the negative and the positive visit procedures and the procedure UpdateE. Figure 5.6 shows the detail. In the "while" loop starting at line 72 the algorithm each time selects one of these three procedures and executes it. The algorithm has two policies, explained in Sections 5.4.2 and 5.4.3, being applied simultaneously to select one of the three procedures and to execute it.

The first policy is that the same amount of time consumed for executing UpdateE(root) is consumed for the execution of the two visit procedures. For this purpose, the algorithm uses a variable CBalance to decide whether UpdateE or the two visit procedures have taken more time so far. As we will prove in the next chapter, the cost of executing UpdateE(root) is $\Omega(|leaves(Q)|)$ and costs of Visit$^-$(root) and Visit$^+$(root) are about L^-(root) and L^+(root), respectively. So after each execution of Visit$^-$(root) and Visit$^+$(root), the algorithm increases CBalance by L^-(root) and L^+(root), respectively, and after each execution of UpdateE(root) the algorithm decreases CBalance by $|leaves(Q)|$. Then, in line 77, based on CBalance being positive or not, the algorithm decides to call UpdateE or to call one of the two visit procedures.

The next policy is that the same amounts of the time are consumed for the execution of Visit$^-$(root) and for the execution of Visit$^+$(root). Hence, the algorithm considers another variable VBalance to decide which of the two visit procedures has taken less time so far and so should be selected. Thus, after each execution of Visit$^-$(root) the algorithm increases VBalance by L^-(root) and after each execution of Visit$^+$(root) the algorithm decreases VBalance by L^+(root). Then, when the algorithm decides not to execute UpdateE and instead run one of the two visit procedures, in line 79 based on VBalance being positive or not, the algorithm decides to call Visit$^+$(root) or to call Visit$^-$(root).

We now explain the heading part of the "while" loop, before line 77. As can be seen, in

Figure 5.6: The main algorithm

```
    begin
71      Initialize();
        VBalance := CBalance := 0;
72      while e < ∞ do
73          if μ(W[root]) = E = e and there is a speedy leaf then
74              while μ(fi[sl(Q)]) < e do                          } Block 19
75                  Gallop(sl(Q));
76          UpdateC;
77          if CBalance > 0 then
78              UpdateE(root);                                     } Block 20
                CBalance := CBalance − |leaves(Q)|;
            else
79              if VBalance > 0 then
80                  Visit+_L(root);
                    VBalance := VBalance − L+(root);
                    CBalance := CBalance + L+(root);
                else
81                  Visit−_L(root);
                    VBalance := VBalance + L−(root);
                    CBalance := CBalance + L−(root);
82      while there is a speedy leaf and fi[sl(Q)] ≠ END do
83          Gallop(sl(Q));
    end
```

the "while" loop starting at line 72, the algorithm calls the procedure UpdateC once (line 76) to increase e if possible, before selecting one of the aforementioned three procedures. But due to the third precondition of UpdateC, when $\mu(\mathsf{W}[\text{root}]) = \mathsf{e}$ the algorithm needs to satiate the speedy leaf (if it exists) before calling UpdateC. Hence, in this situation repeatedly the algorithm calls Gallop($sl(Q)$) (line 75) until the speedy leaf is satiated. Each time that Gallop($sl(Q)$) is executed, more objects of the speedy leaf with values less than e become scanned and thus, as we will prove more formally in the next chapter, ultimately at some point the speedy leaf becomes satiated.

The last point about the algorithm is how to initialize the variables so that they become well-valued. As can be seen, as the first command, the algorithm calls the procedure Initialize on line 71. The procedure Initialize is shown in Figure 5.7. In the next lemma we explain how every variable is initialized and why variables become well-valued.

Lemma 5.56 *After the execution of the procedure Initialize all variables are well-valued and Invariants 1, 2, and 3 hold.*

Proof. We first consider the variables and the arrays of the algorithm one by one and then we discuss the correctness of the three invariants, one after another. By Definition 5.14 variables e, $\mathsf{e_s}$, Φ, and fi[l], for leaves l of Q, are always well-valued. Also, E and e both are set equal to $-\infty$ (line 85) and hence E becomes well-valued (Definition 5.7). Then, for every node v, W[v] is set equal to BEG (line 87). So, as $\mathsf{e} = -\infty$ by Definition 5.8 elements of the array W are well-valued, as well.

We now prove that after executing the loop of line 89 M[v] becomes well-valued, for every node v of $N(Q)$. For every leaf l in $N(Q)$, M[l] is set equal to μ(fi[l]) (line 91) and hence by Definition 5.6 it becomes well-valued. For an internal node v, depending on v being a union or an intersection node, M[v] is set equal to the minimum or the maximum of M[u], for children u of v in $N(Q)$, (lines 92 and 93) and so it becomes well-valued (Definition 5.6). Also, since the loop of line 89 considers nodes of Q in order of height, when executing the "switch" of line 90, this switch has already been run for children of v and hence after initializing M[v], M[u], for children u of v, is not going to change in this loop. Hence after executing the loop, depending on v being a union or an intersection node, M[v] is still equal to the minimum or the maximum of M[u], for all children u of v in $N(Q)$, and so by definition it is well-valued.

Figure 5.7: Initializing variables

```
   Procedure Initialize;
   begin
84     for every leaf l of Q do Step[l] := 1;
85     E := e := e_s := −∞;
86     Φ := the empty sequence;
87     for every node v of Q do W[v] := BEG;
88     for every leaf l of Q do fi[l] := the first object of l;
89     for every internal node v of N(Q), starting from nodes with lower heights do
90         switch type of the node v do
               case leaf node
91                 M[v] := μ(fi[l]);
               case union node
92                 M[v] := min_u M(u) over all children u of v in N(Q);
               case intersection node
93                 M[v] := max_u M(u) over all children u of v in N(Q);
94     for every intersection node v of Q do
           Counter[v] := the number of children of v;
       for every internal node v of Q do Build H[v];
       for every intersection node v of Q do Build G[v];
95     for internal node v do Credit[v]:=0;
   end
```

We now prove the rest of variables becomes well-valued. For every intersection node v, in the loop of line 94 $\mathsf{Counter}[v]$ is set equal to the degree of v and since for every child u of v, $\mu(\mathsf{W}[u]) = \mu(\text{BEG}) = -\infty = \mathsf{E}$, after this assignment $\mathsf{Counter}[v]$ shows the number of children u of v satisfying $\mu(\mathsf{W}[u]) = \mathsf{E}$. Hence, by Definition 5.9 elements of the array $\mathsf{Counter}$ becomes well-valued. Next, the heap $\mathsf{H}[v]$, for internal nodes v of $N(Q)$, and the heap $\mathsf{G}[u]$, for intersection nodes u of Q, are build from scratch at the end of the procedure and so they becomes well-valued. Consequently, all variables are well-valued at the end.

Now we prove that when exiting the procedure Invariant 1 is true by showing that at the end of the procedure C is an eliminating configuration satisfying the three conditions of Definition 3.5. As e and $\mathsf{e_s}$ are set to $-\infty$ (line 85), the inequality $\mathsf{e_s} \leq \mathsf{e}$ holds and thus by Definition 3.2 C is an eliminating configuration. Furthermore, for every value a which is in the value set of the root in I or is in the value set of the root in $N(I)$, $a \not< -\infty = \mathsf{e}$, $a \not< -\infty = \mathsf{e_s}$, and a is not the value of any object in the expansion of Φ as Φ is set equal to the empty sequence (line 86). Hence, the first two conditions of Definition 3.5 are met. Also, since by Definition 3.1 the empty sequence is a partial solution, the third condition is also satisfied. Therefore, C is valid and so Invariant 1 is satisfied.

Finally we prove the correctness of the other two invariants at the end. The correctness of Invariant 2 follows from the fact that for every leaf l $\mathfrak{fi}[l]$ is set equal to the first object of l (line 88) and hence by Definition 5.1 after this assignment there is no scanned object. Now we show that Invariant 3 also holds. At the end of the execution of the procedure, $\mathsf{e_s} = -\infty$ and thus if there is a speedy leaf, the first object of $sl(Q)$ is the smallest object of $sl(Q)$ with a value at least $\mathsf{e_s}$. So, by Lemma 3.1, $find(I, sl(Q), \mathsf{e_s})$ equals the first object of $sl(Q)$. Therefore, as $\mathfrak{fi}[sl(Q)]$ is set equal to the first object of $sl(Q)$, the invariant is satisfied when the procedure exits. □

To show the correctness of the algorithm, we prove that after initializing the variables, and before executing the "while" loop of line 82, after executing every single line of Figure 5.6 all variables remain well-valued and the three invariants hold. We first prove this claim for lines 75 and 76. Then using these results we prove the claim for other lines in Lemma 5.59.

Lemma 5.57 *Suppose before executing the "while" loop of line 74 all variables are well-*

valued and the three invariants hold. Then, after each execution of line 75 all variables are still well-valued and the three invariants still hold.

Proof. We first prove that if before executing line 75 the three invariants hold and all variables are well-valued, after executing line 75 all feature variables of the speedy leaf are well-valued and the three invariants will hold. Then we discuss the variables other than feature variables of the speedy leaf.

Now we present the first part of the proof, as explained. Since by assumption before executing line 75 the three invariants hold, when executing line 75 the precondition of Gallop($sl(Q)$) (Figure 5.2) is met. So, by Lemma 5.38 after executing Gallop($sl(Q)$) on line 75, its postconditions hold, that is, the three invariants are correct and $\mathsf{W}[sl(Q)]$ and $\mathsf{M}[sl(Q)]$ are well-valued. Also, $\mathrm{fi}[sl(Q)]$ and $\mathsf{Step}[sl(Q)]$ are always well-valued by Definition 5.14. So, by Definition 5.13 all feature variables of the speedy leaf are well-valued.

We first prove that after the execution of line 75 feature variables of all nodes except perhaps the root and the speedy leaf are well-valued. Next we discuss feature variables of the root and finally we prove the claim for the variable E. By Observation 5.37 by executing Gallop(l) only changing variables of Gallop($sl(Q)$), that is feature variables of $sl(Q)$ and $\mathsf{e_s}$ and Φ, may be modified. Hence, given a node v of Q other than the root and the speedy leaf, since $sl(Q)$ is not v nor a child of v, by executing line 75 feature variables of v nor any child of v change. So, as before executing line 75 all feature variables of v are well-valued, by Lemma 5.40 after the execution of this line all feature variables of v are still well-valued.

Now we show that by executing line 75 feature variables of the root also becomes well-valued. Since there is a speedy leaf, by Definition 2.23 the root is a union node. Also, a speedy leaf by definition is not in $N(Q)$. Hence, definitions of $\mathsf{M}[\text{root}]$ and $\mathsf{H}[\text{root}]$ being well-valued (Definitions 5.6 and 5.10) do not depend on any feature variable of the speedy leaf. So, the only feature variable of the root whose definition of being well-value depends on a variable of the speedy leaf is $\mathsf{W}[\text{root}]$.

We now prove that $\mathsf{W}[\text{root}]$ also remains well-valued. After checking the condition on line 73 and before entering the "while" loop of line 74, by the condition checked on line 73 $\mu(\mathsf{W}[\text{root}]) = \mathsf{e}$ and so as the root is a union node and $\mathsf{W}[\text{root}]$ is well-valued,

by Definition 5.8 one of the following three conditions hold: $\mathsf{E} \neq \mathsf{e}$, $\mathsf{e} \in \{-\infty, \infty\}$, or $\mu(\mathsf{W}[u]) = \mathsf{e}$, for a child u of v. If one of the first two conditions hold, then as e and E are not changing variables of the procedure Gallop, after each execution of line 75 these conditions still hold and hence W[root] remains well-valued, by definition. So we suppose these two conditions are false and thus the third one is true, that is, $\mathsf{e} = \mathsf{E}$, $\mathsf{e} \notin \{-\infty, \infty\}$, and $\mu(\mathsf{W}[u]) = \mathsf{e}$, for a child u of the root. We now show that u is not the speedy leaf. Since line 75 is executed, by the condition check on line 74 before entering the "while" loop of line 74 $\mu(\mathfrak{fi}[sl(Q)]) \neq \mathsf{e}$. Thus, as before entering the "while" loop of line 74 by assumption $\mathsf{W}[sl(Q)]$ was well-valued and by assumption $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$, by Definition 5.8 $\mu(\mathsf{W}[sl(Q)]) \neq \mathsf{e}$. Hence, $u \neq sl(Q)$ and thus $\mathsf{W}[u]$, as well as W[root], is not a changing variable of Gallop($sl(Q)$). Therefore, after each execution of Gallop($sl(Q)$), $\mu(\mathsf{W}[\text{root}]) = \mu(\mathsf{W}[u]) = \mathsf{e}$ and so W[root] is well-valued.

Since by Definition 5.14, variables other than feature variables of the nodes of Q and E are always well-valued, it just remains to prove that E is well-valued when exiting the procedure Gallop. The correctness of this claim follows from the facts that the definition of E being well-valued just depends on E and e and by Observation 5.37 these two variables are not modified by the procedure Gallop. Hence, the lemma is correct. □

Lemma 5.58 *Suppose before executing line 76 all variables are well-valued and the three invariants hold. Then, after executing this line all variables are still well-valued and the three invariants still hold.*

Proof. We first prove that when line 76 calls Update$\mathsf{C}$, the preconditions of Update$\mathsf{C}$ hold and then using the postconditions of Update$\mathsf{C}$ we show that after executing the procedure Update$\mathsf{C}$, the three invariants holds and all variables are well-valued. The first two preconditions of Update$\mathsf{C}$ hold by assumption. So let us prove the third precondition, that is, if $\mu(\mathsf{W}[\text{root}]) = \mathsf{E} = \mathsf{e}$ and a speedy leaf exists, the speedy leaf is satiated. As it is clear from the algorithm, before executing line 76, Block 19 is executed. We consider two cases based on the condition checked on line 73 being true or false when executing Block 19. First suppose that condition is false and so either there is no speedy leaf or the statement $\mu(\mathsf{W}[\text{root}]) = \mathsf{E} = \mathsf{e}$ is false. Then, after this condition check, line 76 is immediately executed and so when executing line 76 the third precondition is satisfied. Now consider

the other case, in which when executing line 73 its condition was true and so the "while" loop inside Block 19 is executed. Therefore, due to the condition checked on line 74, when finishing executing this "while" loop and starting executing line 76, $\mu(\mathfrak{fi}[sl(Q)]) \geq \mathsf{e}$. Also, by assumption just before executing line 76 the three invariants hold. Thus, by Lemma 5.4 the speedy leaf is satiated at that time. Hence, the third precondition of UpdateC holds in this case, as well.

Now we prove that after executing UpdateC the three invariants hold and all variables are well-valued. Since we proved before calling UpdateC its preconditions hold, by Lemma 5.29, after the execution of UpdateC its postconditions hold and thus the three invariants are true. Also, by the Lemma 5.29 only changing variables of UpdateC, that is C and $\mathfrak{fi}[sl(Q)]$, are modified by UpdateC. Except the variable E and elements of the arrays W and H, the definition of of being well-valued for no variable depends on C, that is on e, $\mathsf{e_s}$, or Φ (Table 5.1). Also, definitions of being well-valued for E and elements of the arrays W and H do not depend on $\mathsf{e_s}$ nor Φ. Thus, it suffices to discuss the situations in which e or $\mathfrak{fi}[sl(Q)]$ is modified.

We first prove it suffices to prove that if e changes then elements of the arrays W and H and the variable E remain well-valued. To prove this, let us first discuss the possible changes to $\mathfrak{fi}[sl(Q)]$. As Table 5.1 shows, the definition of being well-valued for only elements of M and W depend on the elements of the array fi. Also, by Definition 5.6, given two nodes v and u, the definition of $\mathsf{M}[v]$ being well-valued depends on $\mathfrak{fi}[u]$ only if $u = v$ and v is in $N(Q)$. So, as the speedy leaf is not $N(Q)$, the definition of being well-valued for no element of the array M depends on $\mathfrak{fi}[sl(Q)]$. Furthermore the only place in the procedure UpdateC that modifies $\mathfrak{fi}[sl(Q)]$ is in Block 4 and when this block is executed, due to line 6 e is also modified. Hence it suffices to show that if e changes then the elements of the arrays W and H and the variable E remain well-valued.

We now discuss elements of the arrays W and H, in that order. We first show that if e is modified, its value is increased. Having proved this fact, as before executing the procedure by Definition 5.7 the inequality $\mathsf{E} \leq \mathsf{e}$ holds and E is not a changing variable of UpdateC, if e is increased then after executing the procedure $\mathsf{e} > \mathsf{E}$ and thus by Definition 5.8 elements of the array W are well-valued. We then discuss the array H. To prove that UpdateC may only increase e, we consider the only two lines of UpdateC that may modify e. The first

line is line 2 which is executed only if $\mathsf{M}[\text{root}] > \mathsf{e}$ (by the condition checked on line 1) and then it sets $\mathsf{e} := \mathsf{M}[\text{root}]$. So by executing this line e is increased. The other one is line 6 which sets $\mathsf{e} := \mathsf{e}_+$ and hence increases e. So, as explained, after exiting the procedure elements of the array W are well-valued. Now consider the array H. Given an internal node v, if v is a union node, the definition of $\mathsf{H}[v]$ being well-valued (Definition 5.10) does not depend on e. So suppose v is an intersection node. We proved if e is modified the inequality $\mathsf{E} \neq \mathsf{e}$ will hold. Thus, in this case by Definition 5.11 the keys of all children of v in $\mathsf{H}[v]$ are up-to-date and hence $\mathsf{H}[v]$ is well-valued. So, by changing e the elements of the array H do not stop being well-valued.

Finally, we now prove that after exiting Update$\mathsf{C}$, E is well-valued, that is, $\mathsf{E} \leq \mathsf{e}$. The correctness of this claim follows form the facts by Definition 5.7 before executing Update$\mathsf{C}$ $\mathsf{E} \leq \mathsf{e}$, Update$\mathsf{C}$ does not change E, and, as we proved, if e is modified then its value is increased. □

Lemma 5.59 *After initializing the variables in line 71 and before executing the "while" loop of line 82, after executing every line in Figure 5.6 all variables are well-valued and Invariants 1, 2, and 3 hold.*

Proof. Since by Lemma 5.56 after executing line 71 the three invariants hold and all variables are well-valued, it suffices to show that for every line ℓ of Figure 5.6 executed after line 71 and before line 82, if before executing line ℓ the three invariants hold and all variables are well-valued, after executing line ℓ, these conditions still hold. Since a non-procedure call line in Figure 5.6 does not modify any variable except local variables, we prove the aforementioned property just for procedure call lines.

We now consider procedure call lines of Figure 5.6, except lines 71 and 83, one by one. The aforementioned properties for lines 75 and 76 are proved in Lemmas 5.57 and 5.58, respectively. Now consider line 78. It follows from Lemma 5.55 that after executing Update$\mathsf{E}$(root) on this line, the postconditions of Update$\mathsf{E}$(root) hold. So, by the second postcondition of Update$\mathsf{E}$(root), all feature variables of all nodes are well-valued and the three invariants hold. Moreover, by the first postcondition of Update$\mathsf{E}$(root), $\mathsf{E} = \mathsf{e}$ and so by Definition 5.7 E is also well-valued. Hence, it follows from Definition 5.14 that all variables are well-valued.

We now prove these properties for lines 80 and 81. Since we are considering the situations in which before executing these lines, which are calls to Visit$^-$(root) and Visit$^+$(root), Invariants 1, 2, and 3 hold and all variables are well-valued, the procedure calls are legal. Thus, as by Lemma 5.53 the root is well-behaved, by the first condition of Definition 5.15 after executing these lines the postconditions stated in Figures 5.3 hold. Hence, after the execution of these lines the three invariants hold and feature variables of all nodes are well-valued. Also, the definition of E being well-valued just depends on E and e and as these two variables are not changing variables of Visit$^-$(root) and Visit$^+$(root), they are not modified by these procedures, as proved in Lemma 5.43. Hence, after executing each of these procedures, E remains well-valued. In addition, the variables of the algorithm other than the feature variables of the nodes and E are always well-valued, according to Definition 5.14. So, all variables are well-valued after executing lines 80 and 81. □

Now we prove that the algorithm works correctly.

Theorem 5.60 *If the algorithm stops executing, the expansion of* Φ *is a solution to the instance.*

Proof. We prove that when the algorithm exits C is valid and all objects are C-eliminated. Then, the correctness of the theorem follows from Lemma 3.9. To prove these facts we first prove that when the procedure exits the three invariants are true.

We now prove that after the "while" loop of line 82, the three invariants hold and then we conclude that at that time C is valid. By Lemma 5.59 after executing the last line in Figure 5.6 before the 'while" loop of line 82, the three invariants hold and all variables are well-valued. Hence, the postconditions of Gallop($sl(Q)$) are true. Also, if before executing line 83 the postconditions of Gallop($sl(Q)$) hold, since the set of preconditions of Gallop($sl(Q)$) is a subset of the set of its postconditions, all preconditions of Gallop hold. Therefore, by Lemma 5.38 after the execution of Gallop($sl(Q)$) its postconditions still hold. Hence, after each execution of Gallop($sl(Q)$), its postconditions remain satisfied. Therefore, when the algorithm exits, the postconditions of Gallop($sl(Q)$) are true and thus the three invariants are correct. Now, the correctness of the claim that C is valid when the algorithm finishes its execution follows from the correctness of Invariant 1 at that time.

Now we prove that when the procedure exits all objects are C-eliminated. By the condition checked on line 72, when the algorithm stops the execution of the "while" loop of line 72, $\mathsf{e} = \infty$. Consequently, as e is not a changing variable of the procedure Gallop and so line 83 does not modify e, when the algorithm exits, $\mathsf{e} = \infty$ and hence values of all objects are less than e. Thus, as non-speedy leaves are in $N(Q)$, by Definition 3.3 all objects of non-speedy leaves are eliminated. So, it remains to prove that if there is a speedy leaf, all of its objects are eliminated. As we proved, when exiting the algorithm Invariant 3 holds and so $\mathrm{fi}[sl(Q)] = find(I, sl(Q), \mathsf{e_s})$. Moreover, by the condition checked on line 82, when the algorithm stops, if there is a speedy leaf then $\mathrm{fi}[sl(Q)] = \text{END}$. As a result, if there is any speedy leaf, when the algorithm exits, $find(I, sl(Q), \mathsf{e_s}) = \text{END}$. Therefore, if there is a speedy leaf, by Lemma 3.1 values of all of its objects are less than $\mathsf{e_s}$ and thus by Definition 3.3 all objects of the speedy leaf are eliminated. Hence, at the end all objects are C-eliminated and C is valid. So, by Lemma 3.9 the theorem is correct. □

Now we have proved that if the algorithm finishes its execution, the expansion of Φ is a solution to the instance and thus the algorithm has correctly solved the problem for the given instance. In the next chapter we show that the algorithm stops executing after the elapsing of at most an amount of the time that is close to the difficulty of the instance.

Chapter 6

The Scaled Running Time

In this chapter, given an instance $I = (Q, \omega, \mu)$ in which Q is a normalized query tree with at most one speedy leaf, we analyze the running time of the algorithm on I. We first in Section 6.1 show how we can approximate the running time of the algorithm using the number of times that the procedure Gallop is called. Then in Section 6.2 we compare the number of times that the procedure Gallop is called with the difficulty of the instance. In the expressions, we will use "root" to denote the root of Q.

6.1 Calls to Gallop

As the first step towards analyzing the running time of the algorithm, in this section we reduce the problem of evaluating the running time to the problem of counting the number of times that the procedure Gallop is called. We first discuss the time consumed by executing visit procedures and then we study the total time taken by the algorithm.

The number of times that the procedure Gallop is called during an execution of one of the visit procedures depends on, and in fact is controlled by, the contents of the elements of the array Credit. It is possible that during one execution of Visit$^-(v)$, for a union node v, after increasing Credit$[v]$, because of the smallness of the value stored in Credit$[v]$, the algorithm does not visit any child of v and exits immediately. On the other hand, it is also possible that during another execution of Visit$^-(v)$, for the same node v, due to the large value stored in Credit$[v]$ the algorithm has the chance to call the procedure Gallop

for a large number of times. So, to study the number of times that Gallop is called, when considering an execution of Visit$^-$, in addition to the number of times that the procedure Gallop is called during that execution, the amounts by which integers stored in elements of the array Credit are increased (or decreased) are our concerns, as well. At any time during the execution of the algorithm, we define the *credit of an internal node* v as the value of the variable Credit$[v]$. We define the *leaf-cost* of an execution of Visit$^-(v)$ or Visit$^+(v)$, for a node v of Q, as the number of times that Gallop is called during that execution plus the amount by which the sum of the credits of intersection and union nodes is increased (the amount can be negative). To present an equivalent definition, suppose at any time during the execution of the algorithm LeafCost denotes the number of times that the procedure Gallop is called up to that time plus $\sum_{v \in nodes(Q)} \mathsf{Credit}[v]$. Just like regular variables of the algorithm, let us use $\mathsf{LeafCost}^{(t)}$ to denote the value of LeafCost at a specific time t. Then, if *bef* and *aft* are the times just before and just after an execution of one of the two visit procedures, the leaf-cost of that execution is evaluated as $\mathsf{LeafCost}^{(aft)} - \mathsf{LeafCost}^{(bef)}$.

Now we define a number of definitions and notation. Given a leaf l, Gallop(l) only may be called directly by Visit$^-(l)$, Visit$^+(l)$, and the main algorithm. We denote the number of times that Gallop(l), for a leaf l, is called by Visit$^-(l)$, Visit$^+(l)$, and the main algorithm by g_v^-, g_v^+, and g_v^{sl}, respectively. Also we define $g^- = \sum_{v \in leaves(Q)} g_v^-$, $g^+ = \sum_{v \in leaves(Q)} g_v^+$, $g^{\text{sl}} = \sum_{v \in leaves(Q)} g_v^{\text{sl}}$, and $g = g^- + g^+ + g^{\text{sl}}$. Then, g is the total number of times that the procedure Gallop is called. Also, an execution of Visit$^-$(root) or Visit$^+$(root) is called a *round*. Thus, as the concept of the leaf-cost is already defined for executions of the visit procedures, we can talk about the leaf-cost of a round. Finally, given a node v of Q, we define $d(v)$, $\mathcal{U}_v$, and $\mathcal{I}_v$ as the number of children of v, the set of proper ancestors of v that are union nodes, and the set of proper ancestors of v that are intersection nodes, respectively.

As the first step, as explained, we study the number of times that Gallop is called by the two visit algorithms. Since after initialization the credits of the nodes are only modified by the visit procedures, it can be proved that the sum of leaf-costs of all rounds of the algorithm is the number of calls to the procedure Gallop by the visit procedures plus the amount by which the sum of credits of all nodes is increased throughout the algorithm. In the next lemma we prove that the sum of credits of all nodes is increased by a value only

depending on Q and then we discuss how to evaluate the sum of leaf-costs of all rounds of the algorithm. In this way, we will be able to evaluate the number of calls to the Gallop procedure by the two visit procedures. Our upper bound on the integer by which the credit of a node is increased is defined as follows. Suppose v is an internal node with m children $u_1, \ldots, u_m$. Then, we define $L^-_{\max}(v)$ and $L^+_{\max}(v)$ as follows. Recall the definition of workload functions (Definition 2.27).

$$\begin{aligned} L^-_{\max}(v) &= \max_{1\le i\le m} L^-(u_i) \\ L^+_{\max}(v) &= \max_{1\le i\le m} L^+(u_i) \end{aligned}$$

In the next lemma we prove that the two functions $L^-_{\max}$ and $L^+_{\max}$ evaluate an upper bound on credits of the union nodes and the intersection nodes, respectively.

Lemma 6.1 *If v is a node of Q not being visited, the credit of v is at most $L^-_{\max}(v)$ if v is a union node, and it is at most $L^+_{\max}(v)$, otherwise.*

Proof. We present the proof for union nodes. The proof for intersection nodes will be similar: It suffices to exchange all occurrences of "union", "Visit$^-$", "L^-" , and "$L^-_{\max}$" with "intersection", "Visit$^+$", "L^+" , and "$L^+_{\max}$", respectively. At the beginning of the algorithm, all credits are set equal to zero (line 95). After that, $\mathsf{Credit}[v]$ is only modified directly by Visit$^-(v)$ and Visit$^+(v)$. Also, Visit$^+(v)$ modifies the credit of v only if v is an intersection node. So we just consider calls to Visit$^-(v)$. As v is a union node, because of the condition checked on line 38, Visit$^-(v)$ does not exit unless the credit of v is less than $L^-(u)$, for a child u of v. Also, by definition $L^-(w) \le L^-_{\max}(v)$, for every child w of v. Hence, after each execution of Visit$^-(v)$ the credit of v is at most $L^-_{\max}(v)$. So, the lemma is correct. □

Now we can give an upper bound on the sum of credits of all nodes. The next observation follows immediately from Lemma 6.1.

Observation 6.2 *If no node is being visited, the sum of credits of all nodes is at most L_Σ where L_Σ is defined as follows.*

$$L_\Sigma = \sum_{v\in unions(Q)} L^-_{\max}(u) + \sum_{v\in intersections(Q)} L^+_{\max}(u)$$

Next we explain how the leaf-cost of a round is evaluated.

Lemma 6.3 *For every node v the leaf-cost of a call to Visit$^-(v)$ is $L^-(v)$.*

Proof. We use induction on the height of v to prove the lemma. If v is a leaf, the procedure calls Gallop once (line 27) and does not change the credit of any node. Thus, the leaf-cost of the call is one. Also, as v is a leaf, by Definition 2.27 $L^-(v)$ equals one and so the lemma is correct in this case. Now suppose v is an internal node and the lemma is true for all children of v.

First suppose v is an intersection node. Then, in the loop of line 28, for every child u of v the procedure Visit$^-(u)$ is called once. Suppose $u_1, \ldots, u_m$ are children of v in the order considered in this loop and for every i, $1 \leq i \leq m$, bef_i and aft_i are the times just before and just after executing Visit$^-(u_i)$. Also, suppose bef_0 and aft_{m+1} are the times just before and just after executing Visit$^-(v)$. When v is a union node, Visit$^-(v)$ does not change the array Credit or call the procedure Gallop directly. Hence, while running the procedure, before time bef_1, or after time aft_m, or between the times aft_{i-1} and bef_i, for any i, $2 \leq i \leq m$ the array Credit is not modified and the procedure Gallop is not called. Therefore, $\mathsf{LeafCost}^{(bef_1)} = \mathsf{LeafCost}^{(bef_0)}$, $\mathsf{LeafCost}^{(aft_m)} = \mathsf{LeafCost}^{(aft_{m+1})}$ and for every i, $2 \leq i \leq m$, $\mathsf{LeafCost}^{(bef_i)} = \mathsf{LeafCost}^{(aft_{i-1})}$. Also, by induction the leaf-cost of the call Visit$^-(u_i)$ is $L^-(u_i)$ and thus $\mathsf{LeafCost}^{(aft_i)} - \mathsf{LeafCost}^{(bef_i)} = L^-(u_i)$, for every i, $1 \leq i \leq m$. Hence,

$$\begin{aligned}
& \mathsf{LeafCost}^{(aft_{m+1})} - \mathsf{LeafCost}^{(bef_0)} \\
= \; & \mathsf{LeafCost}^{(aft_m)} - \mathsf{LeafCost}^{(bef_1)} \\
= \; & \mathsf{LeafCost}^{(aft_1)} - \mathsf{LeafCost}^{(bef_1)} + \textstyle\sum_{2 \leq i \leq m} \mathsf{LeafCost}^{(aft_i)} - \mathsf{LeafCost}^{(aft_{i-1})} \\
= \; & \mathsf{LeafCost}^{(aft_1)} - \mathsf{LeafCost}^{(bef_1)} + \textstyle\sum_{2 \leq i \leq m} \mathsf{LeafCost}^{(aft_i)} - \mathsf{LeafCost}^{(bef_i)} \\
= \; & \textstyle\sum_{1 \leq i \leq m} \mathsf{LeafCost}^{(aft_i)} - \mathsf{LeafCost}^{(bef_i)} \\
= \; & \textstyle\sum_{1 \leq i \leq m} L^-(u_i).
\end{aligned}$$

In addition, as v is an intersection node, the sum of $L^-(u_i)$, for all children u_i of v, equals $L^-(v)$ (Definition 2.27). Therefore, $\mathsf{LeafCost}^{(aft_{m+1})} - \mathsf{LeafCost}^{(bef_0)} = L^-(v)$ and so the leaf-cost of the call to Visit$^-(v)$ is $L^-(v)$.

Now suppose v is a union node and consider the times *bef* and *aft* just before and just after calling the procedure. At the beginning, the credit of v is increased by $L^-(v)$ (line 36) and thus after this setting $\mathsf{LeafCost} = \mathsf{LeafCost}^{(bef)} + L^-(v)$. Then, for a number of iterations, a child u is selected and Block 11 is executed. We claim that after each execution of this block, still $\mathsf{LeafCost} = \mathsf{LeafCost}^{(bef)} + L^-(v)$. So, suppose the algorithm has chosen a child u of v and is going to execute Block 11 while $\mathsf{LeafCost} = \mathsf{LeafCost}^{(bef)} + L^-(v)$. Then, in line 39 first Visit$^-(u)$ is executed and so as by induction the leaf-cost of this call is $L^-(u)$, after the execution of Visit$^-(u)$ $\mathsf{LeafCost} = \mathsf{LeafCost}^{(bef)} + L^-(v) + L^-(u)$. Then, the assignment on line 40 decreases the credit of v by $L^-(u)$ and hence after this line again $\mathsf{LeafCost} = \mathsf{LeafCost}^{(bef)} + L^-(v)$. Thus as the next lines of the block do not call any procedure or modify the credit of any node, after the execution of the block $\mathsf{LeafCost} = \mathsf{LeafCost}^{(bef)} + L^-(v)$. So, when the procedure exits $\mathsf{LeafCost} = \mathsf{LeafCost}^{(bef)} + L^-(v)$ and thus the leaf-cost equals $L^-(v)$. □

By exchanging all occurrences of the terms "union", "Visit$^-$", and "L^-" with the terms "intersection", "Visit$^+$", and "L^+", respectively, replacing line numbers 27, 28, 36, 39, and 40 with line numbers 46, 47, 52, 56, and 57, respectively, and replacing the block number 11 with 15 in the proof of the previous lemma, the following lemma is proved.

Lemma 6.4 *For every node v the leaf-cost of a call to Visit$^+(v)$ is $L^+(v)$.* □

Now to express the running time of the algorithm in terms of g, we first express the number of times that the visit procedures are called in terms of g and then we consider the cost of each call to the visit procedures. We first consider the negative visit procedure and first we count the number of times that the procedure is called for union nodes. At any time during the execution of the algorithm, we use *UnionCall* to denote the number of calls to the procedure Visit$^-(v)$ such that v is a union node, up to that time.

We now explain how we estimate the value of *UnionCall*. While executing the procedure Visit$^-$, Gallop is called directly only by Visit$^-(l)$, for leaves l of the query tree. Thus, we try to estimate the value of *UnionCall* in terms of the number of times that leaves are negative-visited. Whenever a union node is negative-visited, the credit of v is increased by $L^-(v)$, intuitively meaning that $L^-(v)$ times leaves of the subtree $Q[v]$ can be visited. Thus, given a leaf l of $Q[v]$, every visit to the leaf l corresponds to $\frac{1}{L^-(v)}$ visits to the node

v. Hence, given a leaf l of Q, as by definition $\mathcal{U}_l$ is the set of all proper ancestors of l that are union nodes, one negative-visit to l costs $\sum_{v\in\mathcal{U}(l)} \frac{1}{L^-(v)}$. So, given a node v, we define

$$\mathcal{C}^-(v)=\sum_{u\in\mathcal{U}(v)} \frac{1}{L^-(u)} \tag{6.1}$$

Due to the above explanations, intuitively, in order to have an upper bound on *UnionCall*, it suffices to evaluate the number of times that leaves are visited and then multiply this number by the maximum of $\mathcal{C}^-(l)$, over all leaves l of Q. Now we use the following technique to prove this fact formally. For every node u we consider a variable $\chi_l^-(u)$ which is initially zero. Whenever the algorithm attempts to call Visit$^-(u)$, for a node u other than the root, we increase $\chi_l^-(u)$ by $L^-(u)\mathcal{C}^-(u)$ and we decrease $\chi_l^-(v)$ by $L^-(u)\mathcal{C}^-(v)$, where v is the parent of u. In this way, as the next lemma shows, the sum of $\chi_l^-(v)$, for nodes v of Q, can be used to evaluate the number of times that Visit$^-$ is called for union nodes. Also, as our approach for updating the variables χ_l^- shows, as the algorithm proceeds, we decrease $\chi_l^-(v)$, for nodes v with greater heights, and we increase $\chi_l^-(v)$, for nodes v with smaller heights and so ultimately $\chi_l^-(v)$, for internal nodes, becomes almost zero. Hence, at the end of the algorithm we can use the sum of $\chi_l^-(l)$, for leaves l to estimate *UnionCall*. Also, given a leaf l, as our method for updating $\chi_l^-(l)$ shows, the value of $\chi_l^-(l)$ is the number of times that l is negative-visited multiplied by $\mathcal{C}^-(v)$. Thus, intuitively, in this way it is proved that in order to evaluate an upper bound on *UnionCall*, it suffices to evaluate the number of times that leaves are visited and then multiply this number by the maximum value of $\mathcal{C}^-(l)$, over all leaves l of Q. Recall the definition of the function *unions* from Section 2.2.1.

Lemma 6.5 *At the end of the execution of the algorithm, the following equality holds.*

$$\mathit{UnionCall} = \sum_{v\in \mathit{nodes}(Q)} \chi_l^-(v) + \sum_{v\in \mathit{unions}(Q)} \frac{\mathit{Credit}[v]}{L^-(v)}$$

Proof. We first discuss how the variables *UnionCall* and χ_l^- are changed and then we show that by changing these variables, the credit of nodes are also modified so that the equality remains satisfied. The credits of union nodes, after initialization in the procedure

Initialize, are only modified by the procedure Visit$^-$, as explained before. The value of each of the variables *UnionCall* and $\chi_l^-(v)$, for nodes v of Q, is only changed when the algorithm attempts to call the procedure Visit$^-$. Hence, while considering each of the above variables, we explain how that variables is modified for each call to the procedure Visit$^-$.

We start from the variables $\chi_l^-(w)$, for nodes w of Q, and we investigate the changes to the result of the expression $\sum_w \chi_l^-(w)$ by each attempt to call Visit$^-(u)$, for a node u of Q. Suppose that Visit$^-(u)$, for a node u of Q, is being called. We consider several cases for u. If u is the root no change happens to variables $\chi_l^-(w)$, for any node w of Q. If u is not root, and v is the parent of u, $\chi_l^-(u)$ is increased by $L^-(u)\mathcal{C}^-(u)$ and $\chi_l^-(v)$ is decreased by $L^-(u)\mathcal{C}^-(v)$. Therefore, $\sum_w \chi_l^-(w)$ is increased by $L^-(u)(\mathcal{C}^-(u) - \mathcal{C}^-(v))$. Now first consider the case in which v is a union node. Then, every proper ancestor of u except v is also a proper ancestor of v and so it follows from the definition that $\mathcal{U}_u = \mathcal{U}_v \cup \{v\}$. Thus, by Equation 6.1

$$\mathcal{C}^-(u) - \mathcal{C}^-(v) = \sum_{w \in \mathcal{U}(u)} \frac{1}{L^-(w)} - \sum_{w \in \mathcal{U}(v)} \frac{1}{L^-(w)} = \frac{1}{L^-(v)}.$$

So, $\sum_w \chi_l^-(w)$ is increased by

$$L^-(u)(\mathcal{C}^-(u) - \mathcal{C}^-(v)) = \frac{L^-(u)}{L^-(v)}.$$

Next suppose v is an intersection node. Then the sets of proper ancestors of v and u that are union nodes are the same and thus $\mathcal{U}_u = \mathcal{U}_v$. Hence, due to Equation 6.1, $\mathcal{C}^-(u) = \mathcal{C}^-(v)$ and so the result of $\sum_w \chi_l^-(w)$ remains unchanged. Consequently the result of the expression $\sum_w \chi_l^-(w)$ is modified only when Visit$^-(u)$, for a node u of Q other than the root, is called and the parent v of u is a union node.

Now let us study the situation when Visit$^-(u)$, for a node u of Q, is called and the parent v of u is a union node. The procedure Visit$^-$ is only called in lines 28, 39, and line 81. When line 28 calls Visit$^-(u)$, the parent v of u is an intersection node. Also, line 81 only calls Visit$^-$(root). But when line 39 is executed and Visit$^-(u)$ is called, the parent v of u is a union node. Thus the only procedure call that causes the result of the

expression $\sum_w \chi_l^-(w)$ to change is the procedure call Visit$^-(u)$ of line 39 that increases $\sum_w \chi_l^-(w)$ by $\frac{L^-(u)}{L^-(v)}$, where v is the parent of u, as explained.

Next we explore the changes to *UnionCall*. *UnionCall* by its definition is increased only when Visit$^-(v)$, for a union node v is called. When Visit$^-(v)$, for a union node v, is called line 36 is executed once. Also, line 36 is executed only after the algorithm starts to execute Visit$^-(v)$, for a union node v. Hence, at the end of the execution of the algorithm *UnionCall* is the number of times that line 36 is executed.

Having explored changes to the expressions $\sum_w \chi_l^-(w)$ and *UnionCall*, we now introduce a new variable Changes, which is initially zero, to record the changes to the expression $\mathit{UnionCall} - \sum_w \chi_l^-(w)$. Recall that, as we proved, the only possible change to the result of the expression $\sum_w \chi_l^-(w)$ is due to the execution of line 39. After every time that line 39 is executed and so, as we showed, $\sum_w \chi_l^-(w)$ is increased by $\frac{L^-(u)}{L^-(v)}$, we decrease Changes by $\frac{L^-(u)}{L^-(v)}$. Also, every time that line 36 is executed and so *UnionCall* is increased by one we also increase Changes by one. Therefore, at the end of the algorithm, the value of Changes shows the result of the expression $\mathit{UnionCall} - \sum_w \chi_l^-(w)$. Thus, to prove the lemma, it suffices to show that at the end of the algorithm Changes stores the value of $\sum_{w \in \mathit{unions}(Q)} \frac{\mathsf{Credit}[w]}{L^-(w)}$.

We now prove that at the end of the algorithm $\mathsf{Changes} = \sum_{w \in \mathit{unions}(Q)} \frac{\mathsf{Credit}[w]}{L^-(w)}$. The avriable Credit$[v]$, for a union node v, is only modified by Visit$^-(v)$ on lines 36 and 40. Line 36 increases Credit$[v]$ by $L^-(v)$, that is, increases $\sum_{w \in \mathit{unions}(Q)} \frac{\mathsf{Credit}[w]}{L^-(w)}$ by one and at the same time we are increasing Changes by one, as well. Also, line 40 decreases Credit$[v]$ by $L^-(u)$, for a child u of v, that is decreases $\sum_{w \in \mathit{unions}(Q)} \frac{\mathsf{Credit}[w]}{L^-(w)}$ by $\frac{L^-(u)}{L^-(v)}$, and just before executing line 39, we have decreased Changes by $\frac{L^-(u)}{L^-(v)}$. So, Changes and $\sum_{w \in \mathit{unions}(Q)} \frac{\mathsf{Credit}[w]}{L^-(w)}$ are always increased by the same amount. Also, both of these two are initially zero. Consequently, at the end of the execution of the algorithm,

$$\mathit{UnionCall} - \sum_w \chi_l^-(w) = \mathsf{Changes} = \sum_{w \in \mathit{unions}(Q)} \frac{\mathsf{Credit}[w]}{L^-(w)}$$

and thus

$$\mathit{UnionCall} = \sum_w \chi_l^-(w) + \sum_{w \in \mathit{unions}(Q)} \frac{\mathsf{Credit}[w]}{L^-(w)}.$$

Hence, the lemma is correct. □

Now we evaluate the value of $\chi_l^-(v)$, for nodes v of Q, at the end of the algorithm. We first discuss the case of union nodes v and then we consider intersection nodes v.

Lemma 6.6 *Given a union node v, the equality $\chi_l^-(v) = \mathsf{Credit}[v]\mathcal{C}^-(v)$ holds at the end of the algorithm.*

Proof. Let us track all possible changes to $\mathsf{Credit}[v]$ and $\chi_l^-(v)$ and see why the equation holds. Due to line 95, after initialization of the variables, $\mathsf{Credit}[v] = 0$ and thus the equation $\chi_l^-(v) = \mathsf{Credit}[v] = 0$ holds. Also, after initialization, the credits of the nodes only are modified by visit procedures and, as Figure 5.4 shows, Visit$^+$ only modifies credit of intersection nodes. Hence, since v is union node, a change to $\mathsf{Credit}[v]$ may only happen in the procedure Visit$^-$ by one of lines 40 or 36, that is right after Visit$^-(v)$ starts its execution or or right after Visit$^-(u)$, for a child u of v, finishes its execution. In addition, a change to $\chi_l^-(v)$ only happens when Visit$^-(v)$ or Visit$^-(u)$, for a child u of v, is called. Due to these facts we only consider effects of calls to Visit$^-(v)$ and Visit$^-(u)$, for children u of v, on these two variables.

We first consider the effect of a call to Visit$^-(v)$ in each of the two cases that v is the root and v is not the root, separately, and then we consider the effect of a call to Visit$^-(u)$, for children u of v. If v is the root, $\mathcal{U}_v = \emptyset$ and thus by definition $\mathcal{C}^-(v) = 0$. Hence, as $\chi_l^-(v)$ always increases by a factor of $\mathcal{C}^-(v)$, both sides of the equality claimed in the lemma always remain zero. If v is not root, by executing Visit$^-(v)$ the variable $\chi_l^-(v)$ is increased by $L^-(v)\mathcal{C}^-(v)$ and after starting the execution of Visit$^-(v)$, line 36 increases $\mathsf{Credit}[v]$ by $L^-(v)$. Hence, both of the two sides of the equality are increased by the same value. Now we consider the effect of a call to Visit$^-(u)$, for a child u of v. When calling Visit$^-(u)$, for a child u of v, we decrease $\chi_l^-(v)$ by $L^-(u)\mathcal{C}^-(v)$ and after the execution of Visit$^-(u)$, line 40 decreases $\mathsf{Credit}[v]$ by $L^-(u)$. So, again the two sides of the equality are decreased by the same amount. Hence, at the end of the algorithm the equality holds. □

Lemma 6.7 *Every time that the negative algorithm exits Visit$^-(v)$, where v is an intersection node, $\chi_l^-(v)$ is equal to zero.*

Proof. We consider two cases v = root and $v \neq$ root. If v is the root $\chi_l^-(v)$ is always changed by a factor of $\mathcal{C}^-(v)$ which is zero and so always $\chi_l^-(v) = 0$. Now suppose v is not the root and has m children $u_1, \ldots, u_m$. After calling Visit$^-(v)$, $\chi_l^-(v)$ is increased by $L^-(v)\mathcal{C}^-(v)$. Then, in the loop of line 28 each child u_i of v is visited once and when visiting u_i, $\chi_l^-(v)$ is decreased by $L^-(u_i)\mathcal{C}^-(v)$. Furthermore, since v is an intersection node, by Definition 2.27 $L^-(v) = \sum_{i=1}^m L^-(u_i)$. So $\chi_l^-(v)$ is changed by

$$L^-(v)\mathcal{C}^-(v) - \sum_{i=1}^{m} L^-(u_i)\mathcal{C}^-(v) = \mathcal{C}^-(v)\left(L^-(v) - \sum_{i=1}^{m} L^-(u_i)\right) = 0.$$

□

Now we count the number of calls to the procedure Visit$^-$. In the rest of the chapter, we will use h and k to denote the height of Q and the number of leaves of Q, respectively.

Lemma 6.8 *The number of times that the procedure Visit$^-$ is called is $O(g^-h + hL_\Sigma)$.*

Proof. We show the number of times that the procedure Visit$^-$ is called for union nodes and leaves is $O(g^-h + hL_\Sigma)$. Then, we will argue that the number of times that this procedure is called for non-intersection nodes is at least the number of times that Visit$^-$ is called for intersection nodes. So, the total number of times that nodes are negative-visited is $O(g^-h + hL_\Sigma)$.

Now we count the number of calls to Visit$^-$ for union nodes using the equation proved in Lemma 6.5. First we prove an upper bound on $\chi_l^-(v)$ for nodes v of Q. Given a leaf l, $\chi_l^-(l)$ is increased only when Visit$^-(l)$ is called. Also, each time that Visit$^-(l)$ is called, we increase $\chi_l^-(l)$ by $L^-(l)\mathcal{C}^-(l)$ and by Definition 2.27 $L^-(l) = 1$. Therefore, at the end of the algorithm $\chi_l^-(l)$ is the number of times that Visit$^-(l)$ is called multiplied by $L^-(l)\mathcal{C}^-(l) = \mathcal{C}^-(l)$. Moreover, the number of times that Visit$^-(l)$ is called is at most g_l^- because Visit$^-(l)$ calls Gallop(l). Consequently, at the end the algorithm $\chi_l^-(l) \leq g_l^-\mathcal{C}^-(l)$, for every leaf l. Furthermore, by Lemmas 6.7 and 6.6, at the end of the execution of the algorithm, for every intersection node v, $\chi_l^-(v) = 0$ and for every union node v,

$\chi_l^-(v) = \mathsf{Credit}[v]\mathcal{C}^-(v)$. Hence,

$$\begin{aligned}\sum_{v \in nodes(Q)} \chi_l^-(v) &= \sum_{l \in leaves(Q)} \chi_l^-(v) + \sum_{l \in unions(Q)} \chi_l^-(v) + \sum_{l \in intersections(Q)} \chi_l^-(v) \\ &\leq \sum_{l \in leaves(Q)} g_l^- \mathcal{C}^-(l) + \sum_{v \in unions(Q)} \mathsf{Credit}[v]\mathcal{C}^-(v).\end{aligned}$$

Now we can use Lemma 6.5 to conclude that at the end of the execution of the algorithm

$$\begin{aligned}\mathit{UnionCall} &= \sum_{v \in nodes(Q)} \chi_l^-(v) + \sum_{v \in unions(Q)} \frac{\mathsf{Credit}[v]}{L^-(v)} \\ &\leq \sum_{l \in leaves(Q)} g_l^- \mathcal{C}^-(l) + \sum_{v \in unions(Q)} \mathsf{Credit}[v]\mathcal{C}^-(v) + \sum_{v \in unions(Q)} \frac{\mathsf{Credit}[v]}{L^-(v)} \\ &= \sum_{l \in leaves(Q)} g_l^- \mathcal{C}^-(l) + \sum_{v \in unions(Q)} \mathsf{Credit}[v] \left(\mathcal{C}^-(v) + \frac{1}{L^-(v)}\right).\end{aligned}$$

Now if we consider the fact that, for every node v, $L^-(v)$ by definition is at least 1, it follows from the definition of the function $\mathcal{C}^-$ (Equation 6.1 on page 216) that for every node v, $\mathcal{C}^-(v) \leq |\mathcal{U}_v| \leq h$. Also, by Definition 2.27 $L^-(v) \geq 1$, for every union node v. Therefore, $\mathcal{C}^-(l) \leq h$, for every leaf l, and $\mathcal{C}^-(v) + \frac{1}{L^-(v)} \leq h + 1$, for every union node v. So, we can rewrite the previous inequality as follows.

$$\begin{aligned}\mathit{UnionCall} &\leq h \sum_{l \in leaves(Q)} g_l^- + (h+1) \sum_{v \in unions(Q)} \mathsf{Credit}[v] \\ &= hg^- + (h+1) \sum_{v \in unions(Q)} \mathsf{Credit}[v]\end{aligned}$$

Thus, since by Observation 6.2 the sum of credits of all nodes at the end of the algorithm

is at most L_Σ,

$$\mathit{UnionCall} \leq g^- h + (h+1)L_\Sigma.$$

Now we consider the number of calls to Visit$^-(v)$, for non-union nodes v. Given a leaf l, since Visit$^-(l)$ calls Gallop(l), the number of calls to Visit$^-(l)$ is at most g_l^-. Consequently, the total number of calls to Visit$^-(l)$ for all leaves l is at most $\sum_{l \in leaves(Q)} g_l^- = g^-$. Hence, the number of calls to union nodes and leaves, that is non-intersection nodes, is at most $\mathit{UnionCall} + g^- \leq g^-(h+1) + (h+1)L_\Sigma$. Now consider an intersection node v. The node v has at least one child u and as v is an intersection node, u is a non-intersection node. Also, every time that Visit$^-(v)$ is called, it directly calls Visit$^-(u)$ (line 28). Therefore, the number of calls to Visit$^-$ for intersection nodes is not more than the number of calls to Visit$^-$ for non-intersection ones. Hence, the total number of calls to Visit$^-$ is at most $2(g^-(h+1) + (h+1)L_\Sigma) = O(g^- h + hL_\Sigma)$. □

By exchanging all occurrences of "χ_l^-", "g^-", "g_l^-", "union", "Visit$^-$", "L^-", and "negative-visit" with "χ_l^+", "g^+", "g_l^+", "intersection", "Visit$^+$", "L^+", and "positive-visit", respectively and also by replacing line numbers 28, 36, 39, 40, and 81 with line numbers 47, 52, 56, 57, and 80, respectively, in the previous definitions, lemmas, and arguments, the next lemma is proved.

Lemma 6.9 *The number of times that procedure Visit$^+$ is called is $O(g^+ h + hL_\Sigma)$.* □

To evaluate the time consumed by the algorithm we consider the time consumed by an execution of a single line of the algorithm as the *cost* of that execution and then the sum of costs of all executions of all lines of the algorithm will be the total time taken by the algorithm. Note that in this definition we consider an $O(1)$ cost for an execution of a procedure call line because the costs of executions of the lines inside that procedure are considered separately. Also, the cost of an execution of a block is the total cost of all executions of lines of that block occurred during that execution of that block. In the following argument, whenever we talk about lines, blocks, or commands executed directly by an execution of a procedure, we are considering the lines, blocks, or commands that are executed during the execution of that procedure and are not executed due to a procedure call in that procedure, that is, we are considering each procedure call line ℓ in that procedure

as a black-box and we are ignoring lines executed during the execution of the procedure call line ℓ. Similarly, by a direct execution of a line, a block, or a command by a procedure we mean an execution of that line, block, or command that is happened during the execution of that procedure but is not occurred during the execution of lines of a procedure called inside that procedure.

We now evaluate the total cost of all executions of all lines of the visit procedures. We first evaluate this total cost for all lines of the visit procedures except lines of the "if" blocks of lines 42, 61, and 62 and then we consider the total cost of all executions of these "if" blocks, separately.

Lemma 6.10 *Consider an execution of Visit$^-(v)$ during which the number of times that the procedure selects a child u of v and executes Visit$^-(u)$ is m, for an integer m. The cost of the executions of lines executed directly by the procedure Visit$^-(v)$ during that execution of Visit$^-(v)$, except the "if" block of line 42, is $O(m+1)$.*

Proof. We consider three cases based on v being a leaf, an intersection node, or a union node. If v is a leaf, $m = 0$ because v has no child. Moreover, the cost of executions of lines executed directly by Visit$^-(v)$ (during that single execution of Visit$^-(v)$) is $O(1)$ since Visit$^-(v)$ directly executes just one line, which is a procedure call. Therefore, the lemma is correct in this case.

Next suppose v is an intersection node. Then, the procedure calls Visit$^-(u)$, for every child u of v, and thus $m = d(v)$. Also, the procedure builds the two heaps $\mathsf{H}[v]$ and $\mathsf{G}[v]$, each containing $d(v)$ elements, from scratch which is a well-known standard heap operation and takes $O(d(v)+1)$ time [CSRL01]. So, the cost of executions of all lines directly executed by Visit$^-(v)$ is $O(d(v)+1) = O(m+1)$.

Finally, suppose v is a union node. Then the procedure each time selects a child u of v, calls Visit$^-(u)$, and updates some variables. Thus, ignoring the "if" block of line 42, selecting the child u and and updating the variables are all done in $O(1)$ time. So, ignoring the "if" block of line 42, corresponding to each of these m selections of a child u of v and calling Visit$^-(u)$ the procedure consumes $O(1)$ time. Hence, the cost of all executions of lines executed directly by Visit$^-(v)$ (during that single execution of Visit$^-(v)$) except line 42 is $O(m+1)$. Therefore, the lemma in all cases is true. □

By replacing occurrences of "Visit$^-$" with "Visit$^+$" and occurrences of "the 'if' block of line 42" with "the 'if' blocks of lines 61 and 62" in the proof of Lema 6.10 the following lemma is proved.

Lemma 6.11 *Consider an execution of Visit$^+(v)$ during which the number of times that the procedure selects a child u of v and executes Visit$^+(u)$ is m, for an integer m. The cost of the executions of lines executed directly by the procedure Visit$^+(v)$, except the "if" blocks of lines 61 and 62, is $O(m+1)$.* □

Lemma 6.12 *The sum of costs of all executions of all lines of the visit procedures except the "if" blocks of lines 42, 61, and 62 is $O(gh + hL_\Sigma)$.*

Proof. We claim that the sum of costs of all executions of all lines of the visit procedures except the "if" blocks of lines 42, 61, and 62 is proportional to the number of calls to the visit procedures. Having proved this claim, since by Lemmas 6.8 and 6.9 that the number of calls to the visit procedures Visit$^+$ and Visit$^-$ is $O((g^-h+hL_\Sigma)+(g^+h+hL_\Sigma)) = O(gh+hL_\Sigma)$, we may conclude that the lemma is correct.

We now prove the above claim. We assign a unique ID to each execution of Visit$^+$ or Visit$^-$ and we consider the set $\mathcal{S}$ of all these IDs. Then, we create a sequence Γ of members of the set $\mathcal{S} \times \mathcal{S}$ as follows. Initially we set Γ equal to the empty sequence. Each time that an execution of Visit$^+(v)$ or Visit$^-(v)$ with an ID i starts, for a node v, we add the pair (i, i) to the end of Γ. Also, each time that in an execution of Visit$^+(v)$ or Visit$^-(v)$ with an ID i, for a node v, the algorithm attempts to start an execution of Visit$^-(u)$ or Visit$^+(u)$ with an ID j, for a child u of v, we add the pair (i, j) to the end of Γ. In this, way, given an execution of Visit$^+(v)$ (or Visit$^-(v)$) with an ID i, if n is the number of times that that execution of Visit$^+(v)$ (Visit$^-(v)$, respectively) has attempted to visit its children, we have added $n + 1$ items of the form (i, j), for integers j, to the sequence Γ. Thus, due to Lemmas 6.10 and 6.11 the sum of costs of all executions of all lines of the visit procedures except the "if" blocks of lines 42, 61, and 62 is within a constant factor of the number of items added to the sequence Γ. Now one can observe that for each execution of Visit$^+(v)$ (or Visit$^-(v)$) with an ID j, the number of pairs of the form (i, j), for integers i, in Γ is at most two: one of the form (i, j), for some i, which is added when another execution of the procedure Visit$^+(w)$ (or Visit$^-(w)$) with the ID i has attempted to call Visit$^+(v)$ (or

Visit$^-(v)$), where w is the parent of v, and one of the form (j, j) which is added when the execution with the ID j has been started. Therefore, the number of items added to Γ is at most two times the total number of executions of the two visit procedures. Hence, the sum of costs of all executions of all lines of the visit procedures except the "if" blocks of lines 42, 61, and 62 is within a constant factor the total number of executions of the two visit procedures. So, the lemma is correct, as argued before. □

We now evaluate the cost of executions of the "if" blocks of lines 42, 61, and 62. Consider the time right after finishing a direct execution of line 39 (line 56) by Visit$^-(v)$ (Visit$^+(v)$, respectively), for a node v. In such a situation the algorithm has just finished an execution of Visit$^-(u)$ (Visit$^+(u)$, respectively) for a child u of v and will execute the "if" block of line 42 (the "if" blocks of lines 61 and 62) to update the key of u in $\mathsf{H}[v]$ (the keys of u in $\mathsf{H}[v]$ and $\mathsf{G}[v]$), if necessary. We prove that, in such situations, when executing the "if" block of line 42 (the "if" blocks of lines 61 and 62) the algorithm performs a heap operation only if during the execution of Visit$^-(u)$ (Visit$^+(u)$, respectively) at least one time the procedure Gallop has been called. To prove this fact in Lemma 6.14 we show that if Gallop has not been called then $\mathsf{M}[u]$ remains unchanged and then in Lemma 6.15 we prove that if v is an intersection node and Gallop has not been called, the key of u of $\mathsf{H}[v]$ remains up-to-date. Then we conclude that, due to the conditions checked before updating the heaps in the aforementioned three "if" blocks, no heap operation is performed. Before proving these two lemmas, we first need to prove that the calls to Visit$^-(u)$ and Visit$^+(u)$ are legal so that we can use the correctness of preconditions and postconditions of these procedures before and after their executions.

Lemma 6.13 *Every time that the algorithm calls one of the two visit procedures, before calling that visit procedure all of its preconditions hold.*

Proof. We first prove that every execution of the visit procedures is during an execution of one of the lines 80 and 81. Then we show that every procedure called during an execution of these lines is legal and so the lemma is correct.

The main algorithm only calls Initialize, Visit$^-$(root), Visit$^+$(root), Gallop($sl(Q)$) (if there is a speedy leaf), Update$\mathsf{E}$, and Update$\mathsf{C}$. Also, the procedures Initialize, Gallop,

UpdateE, and UpdateC do not call any of the two visit procedures, directly nor indirectly. So, every call to the visit procedures is during executions of one of the commands Visit$^-$(root) or Visit$^+$(root) on lines 80 and 81. By Lemma 5.59, every time that one of these two lines is executed, all variables are well-valued and Invariants 1, 2, and 3 hold. Hence, every time that the main algorithm calls Visit$^-$(root) or Visit$^+$(root) on lines 80 and 81, all preconditions of these procedures hold. Thus, as by Lemma 5.53 the root is well-behaved, it follows from Definition 5.15 that every procedure call that occurred during the executions of lines 80 and 81 is legal. Therefore, as we proved that all calls to the visit procedures are during executions of lines 80 and 81, every call to one of the visit procedures is legal and so always before the execution of one of the visit procedures, its preconditions hold. □

Lemma 6.14 *Given an internal node v, if during an execution of Visit$^-(v)$ or Visit$^+(v)$ no leaf is visited, then $\mathsf{M}[v]$ is not modified by that execution.*

Proof. We first prove that during the visit to v *maxmin*(v) is not modified and then we show that this means that $\mathsf{M}[v]$ is not modified neither. There is no line in the visit procedures directly modifying the array fi, and the visit procedures only call the visit procedures and the procedure Gallop. Also, the procedure Gallop is called only when a leaf is being visited and hence as by assumption no leaf is visited, Gallop is not called. Thus, elements of the array fi are not modified. Therefore, given a sub-union tree T, as the definition of *min-wait*(T) depends only on elements of the array fi (Definition 5.3), *min-wait*(T) remains unchanged. Hence, it follows from Definition 5.4 that *maxmin*(v) does not change.

Now we prove that $\mathsf{M}[v]$ is not modified. Since by Lemma 6.13 before visiting v the preconditions of the visit procedures held, before visiting v, all feature variables of nodes in $Q[v]$ were valid and so by Lemma 5.14 the equation $\mathsf{M}[v] =$ *maxmin*(v) held. Also, since before visiting v the preconditions of the visit procedures held, the call to Visit$^-(v)$ or Visit$^+(v)$ was legal and thus as by Lemma 5.53 v is well-behaved, by Definition 5.15 after visiting v the postconditions of the visit procedures hold. Consequently, after visiting v all feature variables of nodes in $Q[v]$ are well-valued and so by Lemma 5.14 the equation

$\mathsf{M}[v] = \mathit{maxmin}(v)$ still holds. Hence, as we proved by visiting v $\mathit{maxmin}(v)$ is not changed and before visiting v the equation $\mathsf{M}[v] = \mathit{maxmin}(v)$ held, $\mathsf{M}[v]$ is not modified. □

Given an intersection node v, to prove that if during a visit to a child u of v no leaf is visited then the key of u in $\mathsf{H}[v]$ remains up-to-date, due to Definition 5.11, we show that when $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$, after the visit to u the equality $\mu(\mathsf{W}[u]) = \mathsf{e}$ holds if and only if before visiting u this equation holds. Then, will argue that since before executing u, as we will show, the key of u in $\mathsf{H}[v]$ was up-to-date, after visiting v the key of u in $\mathsf{H}[v]$ is still up-to-date. Since e and E are not changing variables of the visit procedures and thus they are not modified by the visit procedures (Lemma 5.43), we do not use any time stamp for these variables in the next lemma.

Lemma 6.15 *Consider the times bef and aft just before and just after an execution of* $\mathit{Visit}^{-}(v)$ *or* $\mathit{Visit}^{+}(v)$ *during which no leaf is visited. Also, suppose during the visit to* v*, the statements* $\mathsf{e} = \mathsf{E}$ *and* $\mathsf{e} \notin \{-\infty, \infty\}$ *hold. Then,* $\mu(\mathsf{W}^{(\mathrm{aft})}[v]) = \mathsf{e}$ *if and only if* $\mu(\mathsf{W}^{(\mathrm{bef})}[v]) = \mathsf{e}$*.*

Proof. We prove the correctness of the following three statements in order. First, there is an eyewitness for v at time *bef* if and only there is one at time *aft*. Second, $\mu(\mathsf{W}^{(\mathit{bef})}[v]) = \mathsf{e}$ if and only if there is an eyewitness for v at time *bef*. Third, $\mu(\mathsf{W}^{(\mathrm{aft})}[v]) = \mathsf{e}$ if and only if there is an eyewitness for v at time *aft*. Having proved these three claims, the correctness of the lemma follows trivially.

Now we first prove the first claim. Since during the visit to v no leaf is visited, an argument the same as that used in the proof of Lemma 6.14 shows that elements of the array fi are not modified. Also, as mentioned, e is not modified. Hence, as the definition of eyewitnesses depends only on e and the elements of fi (Definition 5.5), there is an eyewitness for v at time *bef* if and only there is one at time *aft*. So, the first claim is true.

Now we prove the second and the third claims. An argument the same as that used in the proof of Lemma 6.14 shows that just before and just after visiting v all feature variables of v are well-valued. Thus, at the times *bef* and *aft* $\mathsf{W}[u]$ is well-valued, for every node u in $Q[v]$. Also, by assumption, at both times *bef* and *aft*, $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$. Hence, due to Lemma 5.15, at time *bef* (at time *aft*), $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if at time *bef* (at

time *aft*), there is an eyewitness for v. So, the second and the third claims are true. Hence, the lemma is correct. □

We now explain how the total time taken by all executions of lines 42, 61, and 62 is evaluated. Given a leaf l, after visiting l, it is possible that $\mathsf{fi}[l]$ is modified and so, $\mathsf{W}[v]$ and $\mathsf{M}[v]$ are changed, for ancestors v of l. Then, due to these changes, the algorithm might perform heap operations to update keys of v in the heaps of the parent of v, for ancestors v of l. Also, as we will see, the time consumed for updating the key of a child of a node w in one of the heaps $\mathsf{H}[w]$ or $\mathsf{G}[w]$ is roughly proportional to $\log d(w)$. Motivated by these facts, we define the *heap-cost of a leaf* l as the sum of $\log(d(w))$ over all ancestors w of l, that is, $\sum_{w \in \mathcal{U}_l \cup \mathcal{I}_l} \log d(w)$. Also, we define the maximum heap-cost of all leaves as the *heap-factor of* Q, denoted by $\mathcal{H}$. As by assumption Q is normalized, w has at least two children and so $\log d(w) \geq 1$, for every ancestor w of every leaf. Consequently, it follows from the definition of the heap-cost that, given a leaf l, the heap-cost of l is at least the number of ancestors of l, that is the height of l. Hence, as by definition $\mathcal{H}$ is the maximum of heap-costs of leaves, the following observation is correct.

Observation 6.16 *The inequality $\mathcal{H} \geq h$ holds.*

We now give an upper bound on the total time taken by lines 42, 61, and 62.

Lemma 6.17 *The cost of all executions of "if" blocks of lines 42, 61, and 62 is $O(g\mathcal{H} + hL_\Sigma)$.*

Proof. We first prove a number of facts showing that specific feature variables of nodes before or after executing lines 39 and 56 are well-valued. Then, we prove that, given a node v, when the "if" block of lines 42 is (the "if" blocks of lines 61 and 62 are) executed directly by $\text{Visit}^-(v)$ ($\text{Visit}^+(v)$, respectively), a heap operation is performed only if during the last direct execution of $\text{Visit}^-(u)$ on line 39 by $\text{Visit}^-(v)$ (execution of $\text{Visit}^+(u)$ on line 56 by $\text{Visit}^+(v)$), at least one leaf is visited. Next, using this fact we show that the cost of all executions of the three aforementioned "if" blocks is roughly at most the number of times that leaves are visited multiplied by $\mathcal{H}$. Then we prove that this result yields the correctness of the lemma.

Now, we show that before and after visiting u in line 39 (executing Visit$^+(u)$ in line 56), $\mathsf{M}[u]$ (if u is in $N(Q)$) and $\mathsf{W}[u]$ are well-valued and also that before executing Block 11 (Block 15) $\mathsf{H}[v]$ and $\mathsf{G}[v]$ (if v is an intersection node) are well-valued. By Lemma 6.13 before visiting u the preconditions of the visit procedures hold and hence $\mathsf{W}[u]$ and $\mathsf{M}[u]$ (if u is in $N(Q)$) are well-valued. As a result, since by Lemma 5.53 u is well-behaved, by Definition 5.15 after visiting u the postconditions of the visit procedures hold and so $\mathsf{W}[u]$ and $\mathsf{M}[u]$ (if u is in $N(Q)$) are well-valued. Moreover, by precondition 1 of the visit procedures, before visiting v feature variables of v are well-valued and by Lemma 5.49 (Lemma 5.52) after each execution of Block 11 (Block 15) feature variables of v are well-valued. Hence, before each execution of Block 11 (Block 15) feature variables of v are well-valued. Thus since before visiting u in Block 11 (Block 15) only elements of the array Credit and Counter may change, just before visiting u, $\mathsf{H}[v]$ and $\mathsf{G}[v]$ (if it is defined) are still well-valued.

Now let us consider an execution of Visit$^-(v)$ (of Visit$^+(v)$), for a node v, and prove that after a direct execution of line 39 (line 56) during which no leaf is visited, the 'if" block of line 42 does not (the "if" blocks of lines 61 and 62 do not) perform any heap operation. We consider two cases based on v being a union node or v is an intersection node. First suppose v is a union node. In this case, we only discuss the "if" block of line 42 as the "if" blocks of lines 61 and 62 are in the "**case** *intersection node:* " block and so they are executed only when v is an intersection node. Also we suppose when executing this "if" block the variable u of the procedure Visit$^-(v)$ is a node in $N(Q)$ as otherwise by the condition checked on line 42 no heap operation is performed. Since before the execution of Visit$^-(u)$ on line 39 $\mathsf{H}[v]$ is well-valued, by Definition 5.10 the key of u in $\mathsf{H}[v]$ is $\mathsf{M}[u]$ and since by assumption while visiting u on line 39 no leaf is visited, by Lemma 6.14 by executing Visit$^-(u)$ $\mathsf{M}[u]$ is not modified. Also, $\mathsf{H}[v]$ is not a changing variables of Visit$^-(u)$ and so by Lemma 5.43 $\mathsf{H}[v]$ is not modified. Hence, when reaching line 42 the key of u in $\mathsf{H}[v]$ is still $\mathsf{M}[u]$. Therefore, due to the condition checked on line 42, no heap operation is performed.

Next we consider the case in which v is an intersection node. Then, the "if" block of line 42 is not executed as that line is in the "**case** *union node:* " block. Also, the same argument as above shows that when reaching line 62 no heap operation is performed: it

suffices to replace occurrences of "union", "$\mathsf{M}[u]$", and "$\mathsf{H}[v]$" with "intersection" , "$-\mathsf{M}[u]$, and "$\mathsf{G}[v]$", respectively. Now prove that the "if" block of line 61 also dose not execute any heap operation. We suppose $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$ as otherwise by Definition 5.11 the key of u in $\mathsf{H}[v]$ is up-to-date and so due to the condition checked on line 61 the proof is trivial. Since, as we proved, before the execution of Visit$^+(u)$ on line 56 $\mathsf{H}[v]$ was well-valued, the key of u in $\mathsf{H}[v]$ was up-to-date and so the key of u in $\mathsf{H}[v]$ was e if $\mu(\mathsf{W}[u]) = \mathsf{e}$; otherwise the key of u in $\mathsf{H}[v]$ was less than e (Definition 5.11). Also, by Lemma 6.15 after execution of Visit$^+(u)$ $\mu(\mathsf{W}[u]) = \mathsf{e}$ if and only if before the execution of Visit$^+(u)$ $\mu(\mathsf{W}[u]) = \mathsf{e}$. Furthermore, by the same argument as in the previous case, by visiting u on line 56 $\mathsf{H}[v]$ is not modified. Hence, after the execution of Visit$^+(u)$, the key of u in $\mathsf{H}[v]$ still is e if $\mu(\mathsf{W}[u]) = \mathsf{e}$; otherwise the key of u in $\mathsf{H}[v]$ is less than e. So, the key of u in $\mathsf{H}[v]$ is up-to-date (Definition 5.11). Thus, by the condition checked on line 61 no heap operation is performed.

Now we evaluate the total cost of all executions of the above three "if" blocks. For every node v we consider a variable $\mathsf{ChildCost}[v]$ and for every leaf l we consider a variable $\mathsf{LeafCost}[l]$ and we initialize all these variables to zero. Consider an execution of Visit$^-(v)$ or Visit$^+(v)$, for a node v, in which after executing Visit$^-(u)$ or Visit$^+(u)$, for a child u of v, the algorithm is going to execute one of the three aforementioned "if" blocks. If during that execution of Visit$^-(u)$ or Visit$^+(u)$ no leaf is visited, the cost of the execution of the "if" block is constant since no heap operation is performed. In this situation, we increase $\mathsf{ChildCost}[u]$ by 1. Otherwise at least one leaf l is visited and so because of the probable heap operations where each one is updating (deleting and reinserting) the key of an element in a heap with at most $d(v)$ elements, the cost of executing the aforementioned "if" blocks of Visit$^-(v)$ or Visit$^+(v)$ is $O(1 + \log d(v))$. So, in addition to increasing $\mathsf{ChildCost}[u]$ by 1, we increase $\mathsf{LeafCost}[l]$ by $\log d(v)$. In this way, at the end of the algorithm the result of the expression

$$\sum_{w \in nodes(Q)} \mathsf{ChildCost}[w] + \sum_{w \in leaves(Q)} \mathsf{LeafCost}[w] \tag{6.2}$$

is proportional to the total cost of all executions of the three "if" blocks.

Now let us evaluate the result of the expression of Equation 6.2. After executing Visit$^-(u)$ or Visit$^+(u)$, for a node u which has a parent, we are increasing $\mathsf{ChildCost}[u]$ by 1.

Hence, $\sum_{w \in nodes(Q)} \mathsf{ChildCost}[w]$ is at most the number of times that the visit procedures are executed, that is, $O((g^- h + hL_\Sigma) + (g^+ h + hL_\Sigma)) = O(gh + hL_\Sigma)$ (Lemmas 6.8 and 6.9). Also, after exiting Visit$^-(l)$ or Visit$^+(l)$, for a leaf l, when returning to Visit$^-(v)$ or Visit$^+(v)$, for every ancestor v of l, we might increase $\mathsf{LeafCost}[l]$ by $O(\log d(v))$. So, because of this execution of Visit$^-(l)$ or Visit$^+(l)$, $\sum_{w \in leaves(Q)} \mathsf{LeafCost}[w]$ is increased by at most the sum of $\log(d(v))$ over all ancestors v of l, that is, the heap-cost of l and by definition the heap-cost of l is at most $\mathcal{H}$. Moreover during every visit to every leaf the procedure Gallop is called and thus the number of times that leaves are visited is at most g. Therefore, at the end of the algorithm $\sum_{w \in leaves(Q)} \mathsf{LeafCost}[w]$ is at most $g\mathcal{H}$. Thus, at the end of the algorithm the result of the expression of Equation 6.2 is $O(gh + hL_\Sigma + g\mathcal{H})$ which equals $O(g\mathcal{H} + hL_\Sigma)$ as by Observation 6.16 $\mathcal{H} \geq h$. Hence, as we proved the total cost of all executions of the "if" blocks of lines 42, 61, and 62 is the result of the expression of Equation 6.2, the lemma is correct. □

As by Observation 6.16 $\mathcal{H} \geq h$, the following lemma follows from Lemmas 6.12 and 6.17.

Lemma 6.18 *The sum of costs of all executions of all lines of the visit procedures is* $O(g\mathcal{H} + hL_\Sigma)$. □

We next investigate the total time taken by all executions of the procedure Gallop. Since, aside from line 17 of the procedure Gallop, an execution of the rest of the procedure takes $O(1)$ time, we first evaluate the sum of costs of all executions of line 17. Given a leaf l, we call an execution of Gallop(l) an *l-gallop*. Also, if during an l-gallop, for a leaf l, line 17 is executed, that l gallop is *slow*; otherwise it is a *fast l-gallop*. Given a leaf l, to analyze the total time taken by l-gallops, we divide the time taken by the execution of line 17 in a slow l-gallop among the fast l-gallops executed just before that slow l-gallop. For this purpose, given a leaf l, we consider the sequence of all l-gallops in order of the starting time and we define every fast l-gallop to be *associated with* its next slow l-gallop (if any exists).

We now give a sketch of the proof. Consider a slow l-gallop executed at time t. In Lemma 6.21 we will prove that when the first l-gallop associated with this l-gallop starts, $\mathsf{Step}[l]$ equals 1. Also, we prove that in each l-gallop associated with a slow l-gallop $\mathsf{Step}[l]$ is increased by a factor of at most two. Then, we conclude that when executing the binary

search of the l-gallop of time t, $\mathsf{Step}[l]$ is at most 2^m+1 where m is the number of the l-gallops associated with the l-gallop of time t. This yields the fact that the binary search takes at most $O(m+1)$ time, as we will see. To prove that each fast l-gallop associated with a slow l-gallop increases $\mathsf{Step}[l]$ by at most a factor of two, we must prove that in such a fast l-gallop, the condition checked on line 12 is false as otherwise the "if" block of line 12 may increase $\mathsf{Step}[l]$ much more than a factor of two. Hence, in the next lemma we prove that if the condition checked on line 12 is true then the l-gallop sets $\mathfrak{fi}[l]$ equal to END, then we show that the index of $\mathfrak{fi}[l]$ never is decreased and so after $\mathfrak{fi}[l]$ is set equal to END, $\mathfrak{fi}[l]$ remains equal to END, and finally in Lemma 6.21 we show that if when starting an l-gallop $\mathfrak{fi}[l] = \text{END}$, the l-gallop is fast. We then conclude that after an l-gallop in which the condition checked on line 12 is true there is no slow l-gallop and so that l-gallop is not associated with any slow l-gallop.

Lemma 6.19 *Consider an l-gallop in which when executing line 12, the condition checked on this line is true. Then, the l-gallop is fast and sets $\mathfrak{fi}[l] = \text{END}$.*

Proof. We prove the lemma by investigating how variables $\mathsf{Step}[l]$, i, and j, are modified. As by assumption the condition checked on line 12 is true, the "if" block of line 12 sets $\mathsf{Step}[l]$ equal to $\text{length}(\omega(l))-i+1$. Then line 13 sets j equal to $i+\mathsf{Step}[l]-1 = \text{length}(\omega(l))$ and hence since by the condition checked on line 12 $\mu(\omega(l)[j]) = \mu(\omega(l)[\text{length}(\omega(l))]) < \mathsf{e}$, by the condition checked on line 14 Block 5 is not executed. So, the binary search is not executed and thus the l-gallop is fast. After that, since by the condition checked on line 12 $i \leq \text{length}(\omega(l))$ and we proved that j is set equal to $\text{length}(\omega(l))$, the condition checked on line 19 is true and hence line 20 sets $\mathfrak{fi}[l]$ equal to $\omega(l)[j+1] = \omega(l)[\text{length}(\omega(l))+1] = \text{END}$. Consequently, the lemma is correct. □

Lemma 6.20 *Consider the times bef and aft just before and just after executing a line of the algorithm modifying $\mathfrak{fi}[l]$, for a leaf l. Then, $\mu(\mathfrak{fi}^{(\text{bef})}[l]) \leq \mu(\mathfrak{fi}^{(\text{aft})}[l])$.*

Proof. We consider the only three lines of the algorithm modifying the array $\mathfrak{fi}$, one by one. The first one is line 20. This line sets $\mathfrak{fi}[l] := \omega(l)[j+1]$ where $j \geq i$ (the condition checked on line 19). Also, due to the assignment on line 11, $i = \mathit{index}(I, l, \mathfrak{fi}^{(\text{bef})}[l])$ and

hence by Definition 2.8 $\mathfrak{fi}^{(bef)}[l] = \omega(l)[i]$. Therefore, as we showed $j \geq i$, $j+1 > i$ and so $\mu(\mathfrak{fi}^{(aft)}[l]) = \mu(\omega(l)[j+1]) > \mu(\omega(l)[i]) = \mu(\mathfrak{fi}^{(bef)}[l])$. Hence, the lemma is correct in this case. Now, we consider the other two lines modifying an element of the array fi, that is, lines of Block 3 of Figure 5.1. This block either sets $\mathfrak{fi}[sl(Q)]$ equal to END or sets $\mathfrak{fi}[sl(Q)]$ equal to the object after $\mathfrak{fi}[sl(Q)]$. In the first case, $\mu(\mathfrak{fi}^{(aft)}[l]) = \mu(\text{END}) = \infty \geq \mu(\mathfrak{fi}^{(bef)}[l])$ and in the second case, trivially $\mu(\mathfrak{fi}^{(aft)}[l]) > \mu(\mathfrak{fi}^{(bef)}[l])$ as μ is ordered. So, the lemma is correct in all cases. □

Lemma 6.21 *The total time taken by all executions of line 17 is $O(g)$.*

Proof. By investigating the changes to the variable $\mathsf{Step}[l]$, we prove that before starting a slow l-gallop with m associated fast l-gallops $\mathsf{Step}[l]$ is at most 2^m and then we conclude that the binary search executed during that slow l-gallop takes $O(m+1)$ time. Finally, we argue that this result yields the correctness of the lemma.

Let us first see how each of the fast and the slow l-gallops modify $\mathsf{Step}[l]$. Suppose at time t the algorithm starts a slow l-gallop and the number of fast l-gallops associated with this slow l-gallop is m. Also, consider time *bef* just before executing the first fast l-gallop associated with the slow l-gallop being talked about. We prove that $\mathsf{Step}^{(bef)}[l] = 1$ and then we show that $\mathsf{Step}^{(t)}[l] \leq 2^m$. First suppose there is no l-gallop before time *bef*. Then, since the only line outside Gallop(l) modifying $\mathsf{Step}[l]$ is in the procedure Initialize and after executing Initialize, due to line 84, $\mathsf{Step}[l] = 1$, $\mathsf{Step}^{(bef)}[l] = 1$. Now consider the other case, in which there is an l-gallop before time *bef*. Then the last l-gallop before time *bef* is slow because otherwise that l-gallop was associated with the l-gallop of time t. Hence, because of line 18 executed in that slow l-gallop just after executing line 17 the equality $\mathsf{Step}^{(bef)}[l] = 1$ holds.

We now prove that $\mathsf{Step}^{(t)}[l] \leq 2^m$ and then we conclude that the cost of the execution of line 17 in the l-gallop of time t is $O(m+1)$. To prove this fact we first show that in any l-gallop before time t, when executing line 12, the condition checked on this line is false. Suppose in an l-gallop, when executing line 12 the condition checked on this line is true. Then, by Lemma 6.19 that l-gallop sets $\mathfrak{fi}[l]$ equal to END and since $\mu(\text{END}) = \infty$ is the maximum possible, it follows from Lemma 6.20 that after this setting $\mu(\mathfrak{fi}[l])$ remains equal to ∞ and thus $\mathfrak{fi}[l]$ remains equal to END. Hence, when executing the l-gallop of time t

fi$[l]$ = END. As a result, if Block 5 is executed, when checking the condition on line 15, this condition is true and so the binary search is not executed, contradicting the fact that the l-gallop of time t is slow. So, we proved that in all l-gallops associated with the l-gallop of time t, when checking the condition checked on line 12, this condition is false.

Now we complete the proof of the claim $\mathsf{Step}^{(t)}[l] \leq 2^m$. In every l-gallop associated with the l-gallop of time t the "if" block of line 12 doubles $\mathsf{Step}[l]$ (because we proved the condition checked on line 12 is false) and then possibly line 18 sets $\mathsf{Step}[l] := 1$. Hence, in each l-gallop associated with the l-gallop of time t, $\mathsf{Step}[l]$ is set equal to 1 or is doubled. Consequently, as we proved that just before the first l-gallop associated with the l-gallop of time t, $\mathsf{Step}[l] = 1$, after executing the mth fast l-gallop associated with the l-gallop of time t, $\mathsf{Step}[l] \leq 2^m$. So, since by definition after the last l-gallop associated with the l-gallop of time t and before time t there is no l-gallop, at time t the inequality $\mathsf{Step}[l] \leq 2^m$ holds.

Next we prove that the binary search of the l-gallop of time t takes at most $O(m + 1)$ time. It follows from Lemma 6.19 that in this l-gallop, when executing line 12, the condition checked on this line is false and so the "if" block of this line doubles $\mathsf{Step}[l]$. So, after executing this "if" block $\mathsf{Step}[l] \leq 2^{m+1}$. Hence, when executing line 17, due to the assignment on line 13, $j - i + 1 = \mathsf{Step}[l] \leq 2^{m+1}$ and thus since the binary search on line 17 seeks a k between i and $\min(j, \text{length}(\omega(l)))$, the binary search is a search among at most 2^{m+1} members. Consequently, it takes at most $O(m + 1)$ time.

Now we can prove the claim of the lemma. Suppose there are n_f fast l-gallops and n_s slow l-gallops and m_i is number of the fast l-gallops associated with the ith slow l-gallop, for every i, $1 \leq i \leq n_s$. Then, since each slow l-gallop is associated with most one fast l-gallop, $\sum_{i=1}^{n_s} m_i \leq n_f$. Also, we proved the cost of an execution of line 17 in a slow l-gallop with m_i associated fast l-gallops is $O(m_i + 1)$, for every i, $1 \leq i \leq n_s$. Thus, the total cost of all executions of line 17 is within a constant factor of $\sum_{i=1}^{n_s}(1 + m_i) = n_s + \sum_{i=1}^{n_s} m_i \leq n_s + n_f = g$. So, the lemma is correct. □

We now can evaluate the total cost of all executions of all lines of the procedure Gallop. This procedure is executed g times, in each execution every line of the procedure is executed at most once, and the cost of each execution of each line of the procedure except line 17 is $O(1)$. Hence, the total cost of all executions of all lines of the procedure Gallop except

line 17 is $O(g)$. Also, by Lemma 6.21 the total cost of all executions of line 17 is $O(g)$. Hence, the following lemma is obtained.

Lemma 6.22 *The total cost of all executions of all lines of the procedure Gallop is equal to* $O(g)$. □

We now present an upper bound on the total time taken by all executions of the procedure UpdateE. We first give a sketch of the proof of the upper bound. In Lemma 6.26 we will prove that at the end of the execution of the algorithm CBalance will store the difference between a constant factor of the time taken by all executions of the procedure UpdateE and the sum of leaf-costs of all rounds. Also, in Lemma 6.25 we show that after every execution of the body of the loop of line 72, $|\mathsf{CBalance}|$ is not more than k where, as we defined before, $k = |\mathit{leaves}(Q)|$. In this way, it is proved that the total time taken by all executions of the procedure UpdateE is within a constant factor of the sum of leaf-costs of all rounds and k. Finally, in Lemma 6.27 we give an upper bound on the sum of leaf-costs of all rounds in terms of g. Before proving that after every execution of the body of the "while" loop of line 72, $|\mathsf{CBalance}| \leq k$, since the body of the "while" loop may decrease CBalance by $L^-(\text{root})$ or $L^+(\text{root})$, we first prove that neither $L^-(\text{root})$ nor $L^+(\text{root})$ is more than k.

Lemma 6.23 *Given a node* v *of* Q, $L^-(v) \leq |\mathit{leaves}(Q[v])|$.

Proof. We prove the lemma using induction on the height of v. If v is a leaf, the lemma is true as $Q[v]$ has one leaf and by Definition 2.27 $L^-(v) = 1$. Now suppose v is an internal node with m children $u_1, \ldots, u_m$ and the lemma is true for every child u_i of v. By Definition 2.27 if v is an intersection node then $\sum_{i=1}^{m} L^-(u_i) = L^-(v)$; otherwise $\sum_{i=1}^{m} L^-(u_i) \geq L^-(v)$. Hence, in either $\sum_{i=1}^{m} L^-(u_i) \geq L^-(v)$. Also, by induction $|\mathit{leaves}(Q[u_i])| \geq L^-(u_i)$, for every child u_i of v. Thus,

$$|\mathit{leaves}(Q[v])| = \sum_{i=1}^{m} |\mathit{leaves}(Q[u_i])| \geq \sum_{i=1}^{m} L^-(u_i) \geq L^-(v).$$

□

By exchanging all occurrences of the terms "union" and "L^-" with the terms "intersection" and "L^+", respectively, in the proof of the previous lemma, the following lemma is also proved.

Lemma 6.24 *Given a node v of Q, $L^+(v) \leq |leaves(Q[v])|$.* □

Lemma 6.25 *After and before each execution of the body of the "while" loop of line 72, $|\mathsf{CBalance}| \leq k$.*

Proof. We first prove that before the first execution of the body of this "while" loop $|\mathsf{CBalance}| \leq k$ and then we prove that after each execution of this body the above inequality remains satisfied. Before starting the "while" loop of line 72 due to initialization of the variable CBalance just before line 72, $|\mathsf{CBalance}| = 0 \leq k$.

Now we consider the times *bef* and *aft* just before and just after an execution of the body of this "while" loop, we suppose $|\mathsf{CBalance}^{(bef)}| \leq k$, and we prove that $|\mathsf{CBalance}^{(aft)}| \leq k$. We consider two cases $\mathsf{CBalance}^{(bef)} > 0$ and $\mathsf{CBalance}^{(bef)} \leq 0$. In the first case, by the condition checked on line 77 UpdateE(root) is executed and after this execution the algorithm decreases CBalance by $|leaves(Q)| = k$. Hence, as by assumption $0 < \mathsf{CBalance}^{(bef)} \leq k$, $-k < \mathsf{CBalance}^{(aft)} \leq 0$ and so $|\mathsf{CBalance}^{(aft)}| \leq k$. Now consider the case $\mathsf{CBalance}^{(bef)} \leq 0$. Then, one of the two visit procedures is executed and then the algorithm increases CBalance by $L^+(\text{root})$ or $L^-(\text{root})$. Also, by Lemmas 6.23 and 6.24 both $L^-(\text{root})$ and $L^+(\text{root})$ are at most $|leaves(Q[\text{root}])| = k$. Therefore, $\mathsf{CBalance}^{(bef)} \leq \mathsf{CBalance}^{(aft)} \leq \mathsf{CBalance}^{(bef)} + k$. Hence, as by assumption $-k \leq \mathsf{CBalance}^{(bef)} \leq 0$, $-k \leq \mathsf{CBalance}^{(aft)} \leq k$ and so $|\mathsf{CBalance}^{(aft)}| \leq k$. □

Lemma 6.26 *The cost of all executions of all lines of the procedure UpdateE is proportional to the sum of leaf-costs of all rounds plus k.*

Proof. We first evaluate the time taken by one execution of UpdateE and then we consider the total time consumed by all executions of this procedure.

Considering an execution of UpdateE(v), for a node v, we now prove that that the cost of all lines executed directly by UpdateE(v) is $O(1 + d(v))$. To prove this, one can observe that in addition to the lines executed just once directly by UpdateE(v), the procedure

might execute two "for" loops, one for calling UpdateE(u), for all children u of v, and one for updating Counter[v] (the first loop is always executed). Since the cost of one time execution of the body of each of these loops is $O(1)$, the cost of execution of each of these two "for" loops is $O(d(v)+1)$. Also, the procedure might build a heap (the heap H[v] on line 70) from scratch and since each of these two heaps has at most $d(v)$ elements, the cost of each execution of these lines is also $O(1+d(v))$. The cost each execution of the other lines of the procedure is also $O(1)$. Hence, the overall cost of all executions of all lines executed directly by UpdateE(v) during one execution of this procedure is $O(1+d(v))$.

Now we evaluate the cost of all executions of all lines executed during one time execution of UpdateE(root). When UpdateE(v), for a node v is called, it recursively calls UpdateE(u) once, for each child u of v. Hence, by one execution of UpdateE(root), UpdateE(u) is executed once, for every node u of Q. Therefore, as we proved the total cost of execution of all lines executed directly by UpdateE(v), for a node v, is $O(1+d(v))$, the overall cost is $O(\sum_{u\in nodes(Q)} 1+d(u)) = O(|nodes(Q)|) = O(|leaves(Q)|) = k$.

Finally we evaluate the total cost of all executions of all lines of UpdateE. As Figure 5.6 shows, after each execution of Visit$^-$(root) (of Visit$^+$(root)), the algorithm increases CBalance by $L^-(v)$ (by $L^+(v)$) and by Lemma 6.3 (by Lemma 6.4) we know the leaf-cost of a round started by calling Visit$^-$(root) (by calling Visit$^+$(root)) is $L^-(v)$ (by $L^+(v)$). Also, after each execution of UpdateE(root) the algorithm decreases CBalance by k. Therefore, at the end of the execution of the algorithm, |CBalance| is the difference between the number of calls to UpdateE(root) multiplied by k and the total leaf cost of all rounds. In addition, we proved that the total cost of all lines executed, directly or indirectly, by one time execution of UpdateE(root) is $O(k)$. Hence, at the end of the algorithm, the total cost of all executions of all lines of the procedure UpdateE is within a constant factor of the sum of |CBalance| and the total leaf-cost of all rounds. So, as by Lemma 6.25 at the end the algorithm |CBalance| $\leq k$, the lemma is proved. □

Now we prove an upper bound on the total leaf-cost of all rounds. It follows from the definition of the leaf-cost that the total leaf-costs of all rounds is the amount by which the sum of credits of all nodes are increased by rounds of the algorithm plus the number of times of the procedure Gallop is called by rounds. The number of times that the procedure Gallop is called by rounds is at most g. Also, since right after initializing the variables

credits of all nodes are zero and after initialization credits of nodes are only modified during the executions of rounds, the amount by which credits of all nodes is increased by all rounds equals the sum of credits of all nodes at the end of the execution of the algorithm. The sum of credits of all nodes at the end of the execution of the algorithm is by Observation 6.2 at most L_Σ. Hence, the correctness of the next lemma is proved.

Lemma 6.27 *The total leaf-cost of all rounds is at most* $g + L_\Sigma$. □

Now we evaluate the time taken by the Initialize procedure and then we compute the time consumed by the whole algorithm.

Lemma 6.28 *The procedure Initialize is executed in* $O(k^2)$ *time.*

Proof. We consider lines of the procedure one by one. It is clear from the code of the procedure that each line before line 89 is executed in $O(k)$ time. Although this is not the most efficient way, to execute the "for" loop of line 89 the algorithm can simply find the heights of nodes in $O(k^2)$ (each node in $O(k)$) time, sort nodes in the order of their heights in $O(k \log k)$ time, and then execute the body of the loop for every node. Each execution of the body of the loop takes at most $O(|nodes(Q)|) = O(k)$ time. Hence, the total time taken by this loop is $O(k^2)$. After the "for" loop of line 89 there are four "for" loops where each execution of the body of each "for" loop takes $O(|nodes(Q)|) = O(k)$ time. So, the execution of each of these "for" loops take $O(k^2)$ time. Consequently, the total time taken by the procedure is $O(k^2)$. □

Lemma 6.29 *The algorithm is executed in* $O(g\mathcal{H} + hk^2)$ *time.*

Proof. We discuss the total cost of all executions of all lines of every procedure and also the cost of all lines of the main algorithm, separately. The procedure Initialize is executed once and by Lemma 6.28 it takes $O(k^2)$ time to be executed. So, the total cost of all executions of all lines of this procedure is $O(k^2)$.

Next we consider the total cost of all executions of lines of the "while" loop of line 72. Suppose the body of the "while" loop is executed m times. Since line 75 executes the procedure Gallop, this line is executed at most g times and hence the total cost of all

executions of this line is $O(g)$. As a result, as Block 19 is executed once in each execution of the body of the "while" loop of line 72, the total cost of all executions of this block is $O(g+m)$. Also, except lines of Block 19, by each execution of the body of the "while" loop of line 72, each line of this body is executed at most once and so the total cost of all executions of all these lines is $O(m)$. Therefore, the total cost of all executions of all lines of the "while" loop of line 72 is $O(g+m)$.

Now we consider the total cost of all executions of other procedures. Since the procedure Update$\mathsf{C}$ is executed exactly m times and each time this procedure takes $O(1)$ time, the total cost of all executions of lines of this procedure is $O(m)$. Also, by Lemma 6.18 the total cost of all executions of all lines of the two visit procedures is $O(g\mathcal{H}+hL_\Sigma)$ and it follows from Lemmas 6.26 and 6.27 that the total cost of all executions of lines of the procedure Update$\mathsf{E}$ is $O(g+L_\Sigma+k)$. Furthermore by Lemma 6.22 the total cost of all executions of all lines of the procedure Gallop is $O(g)$. So, the total cost of all executions of lines of the procedures Update$\mathsf{C}$, Gallop, Visit$^-$, Visit$^+$, and Update$\mathsf{E}$ is $O(m+g+(g\mathcal{H}+hL_\Sigma)+(g+L_\Sigma+k))=O(m+g\mathcal{H}+hL_\Sigma+k)$.

Finally we consider the "while" loop of line 82. The body of this loop calls Gallop and so it is executed at most g times. Therefore, the total cost of all executions of lines of this loop is $O(1+g)$.

Now we give an upper bound on m and summarize the results. Since in each execution of the body of the "while" loop of line 72 one of procedures Visit$^-$, Visit$^+$, or Update$\mathsf{E}$ is executed, m is within a constant factor of the total cost of all executions of all lines of these procedures, that is $O((g\mathcal{H}+hL_\Sigma)+(g+L_\Sigma+k))=O(g\mathcal{H}+hL_\Sigma+k)$. So, the total time taken by the algorithm is $O(k^2+(g+m)+(m+g\mathcal{H}+hL_\Sigma+k)+(1+g))$ where $m=O(g\mathcal{H}+hL_\Sigma+k)$. So, the total time is $O(k^2+g\mathcal{H}+hL_\Sigma)$. Also, given an internal node v, $L^-_{\max}(v)=L^-(u)$, for a child u of v and by Lemma 6.23 $L^-(u)\leq|\mathit{leaves}(Q[u])|\leq k$. So, $L^-_{\max}(v)\leq k$, for every internal node v. Similarly it can be proved that $L^+_{\max}(v)\leq k$, for every internal node v, and so it follows from the definition of L_Σ that $L_\Sigma\leq k^2$. Consequently, the algorithm is executed in $O(hk^2+g\mathcal{H})$ time. □

6.2 The Running Time

In this part we compare the sum of leaf-costs of all rounds with the difficulty of the instance and then we use Lemma 6.29 to give an upper bound on the running time of the algorithm in terms of the difficulty of the instance. Since the difficulty of an instance by definition is the minimum L-gap cost of all of its proofs (defined in Equation 2.4), we consider a proof $\mathcal{P}$ of I and we compare the sum of leaf-costs all rounds with the L-gap cost of $\mathcal{P}$. For this purpose, we consider an eliminating proof $\Pi = \pi_1, \ldots, \pi_n$ for I such that $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$ (Theorem 4.28). Since by Definition 3.13 Π is valid, the following observation follows from Lemma 3.22.

Observation 6.30 *$\Pi^{(i)}$ is valid, for any i, $0 \leq i \leq n$.*

We now establish some connections between the execution of the algorithm and the steps of Π. At any moment, we say a prefix $\Pi^{(i)}$ of Π, $0 \leq i \leq n$, is *incomplete* if e is less than the e-value of the result of $\Pi^{(i)}$ on I; otherwise $\Pi^{(i)}$ is *complete*. If there is an incomplete prefix of Π, we define *cur* as an integer such that $\Pi^{(cur)}$ is the shortest incomplete prefix of Π; otherwise we define $cur = n + 1$. Note that as $\Pi^{(0)}$ is empty, by Definition 3.13 the e-value of its result is $-\infty \leq \mathsf{e}$ and hence $\Pi^{(0)}$ is always complete. Thus, at any moment $cur \geq 1$. Also, by definition, if $cur = n + 1$ then all prefixes of Π are complete; otherwise a prefix $\Pi^{(i)}$ of Π is complete if $i < cur$. Hence, in either case, a prefix $\Pi^{(i)}$ of Π is complete if $i < cur$.

Observation 6.31 *At any moment, $cur \geq 1$ and a prefix $\Pi^{(i)}$ of Π is complete if $i < cur$.*

The main idea is that, when $cur \neq n + 1$, we prove an upper bound on the number of times that the procedure Gallop is called before $\Pi^{(cur)}$ becomes complete. Once all steps of Π are complete, as we will prove, $\mathsf{e} = \infty$ and the algorithm finishes its execution. When $cur \neq n + 1$, defining e_{goal} as the e-value of the result of $\Pi^{(cur)}$ on I, the inequality $\mathsf{e} < \mathsf{e}_{\text{goal}}$ holds as otherwise $\Pi^{(cur)}$ is complete. Also, when $cur \neq n+1$, defining e_{old} as the e-value of the result of $\Pi^{(cur-1)}$ on I, by Observation 6.31 $\Pi^{(cur-1)}$ is complete and so by the definition of a prefix of Π being complete $\mathsf{e}_{\text{old}} \leq \mathsf{e}$. Therefore, the following observation is true.

Observation 6.32 *At any moment that $cur \neq n + 1$, $\mathsf{e}_{\text{old}} \leq \mathsf{e} < \mathsf{e}_{\text{goal}}$.*

We now distinguish several categories of rounds and we evaluate the total leaf-cost of rounds in each category, separately. Let us first explain the main idea of defining these categories. Suppose $cur \neq n+1$. It can be proved that when π_{cur} is of the first type, every value at least e and less than e_{goal} is not in the value set of the root in $N(I)$ and when π_{cur} is of the second type the value e is in the value set of the root in I. Also, as explained in Section 5.1, the procedures Visit$^-$ and Visit$^+$ are optimized for when e is not in the value set of the root in $N(I)$ and for when e is in the value set of the root in I, respectively. Hence, when π_{cur} is of the first (second) type, we only rely on the effect of the procedure Visit$^-$ (the procedure Visit$^+$) on e and we consider the time consumed by the procedure Visit$^+$ (the procedure Visit$^-$) as overhead. Also, when π_{cur} is of the second type but $\mathsf{E} \neq \mathsf{e}$ it is not guaranteed that W[root] shows the existence of an eyewitness for the root correctly and so in such situations we will not be able to give an upper bound on the number of times that Gallop should be called before e is increased. So, when when π_{cur} is of the second type but $\mathsf{E} \neq \mathsf{e}$, every visit is considered as a "wasted" visit.

We now define the categories of rounds, formally. As the definition of a prefix of Π being complete is based on e and e is not modified during a round (because e is not a changing variable of the visit procedures), *cur* is not modified during a round. If during a round r $cur \neq n+1$, π_{cur} is defined as the *active step of the round* r. Also, as we will prove in Lemma 6.35, if when starting a round r $cur = n+1$, the round is the last round. Now, to define the categories that we talked about, we first define the state of a round as follows. Consider a round r such that during r, $cur \neq n+1$ and thus r has an active step. Then, r has a *negative state* if the active step of r is of the first type; otherwise, if the active step of r is of the second type then r has a *positive state*. A *negative round* is a round with the negative state that is started by calling Visit$^-$(root). A round r with the positive state that is started by calling Visit$^+$(root) is a *positive round* if $\mathsf{E} = \mathsf{e}$ when r starts; otherwise r is a *wasted round*. A round that is not placed in any of categories negative, positive, and wasted, is a *null round*. As explained above, we will only rely on the effect of negative and positive rounds and the time taken by wasted and null rounds is considered as an overhead. Due to our definitions, when the state of a round starting by calling Visit$^+$(root) (by calling Visit$^-$(root)) is positive (negative), the round is either a positive or a wasted round (is a negative round) and thus it is not null. Also, a round with

a positive (negative) state that is started by calling Visit$^-$(root) (by calling Visit$^+$(root)) is not negative, wasted, or positive, and so it is null. In addition, Visit$^-$(root) (Visit$^+$(root)) is called only by line 81 (by line 80). So, the following observation is true.

Observation 6.33 *A round with a positive (negative) state is a null round only if and only if it is started by an execution of line 81 (of line 80).*

We now prove before the last round $cur \neq n+1$ and π_{cur} is of the first or of the second type. To prove this result, we prove that every time that the algorithm reaches line 77, if $cur = n+1$ or π_{cur} is of the third type, $\mathsf{e} = \infty$. Then, we conclude that on the next time that the algorithm checks the condition of line 72, the algorithm exits the "while" loop of line 72 and so no more rounds will occur.

Lemma 6.34 *After an execution of line 76, if $cur = n+1$ or π_{cur} is of the third type then $\mathsf{e} = \infty$.*

Proof. We prove the lemma in the case $cur = n+1$ and in the case in which $cur \neq n+1$ but π_{cur} is of the third type, separately. First suppose $cur = n+1$. Then, by Observation 6.31 $\Pi^{(n)}$ is complete and hence e is at least as large as the e-value of the result of $\Pi^{(n)} = \Pi$ on I. But as Π is an eliminating proof, by Definition 3.13 the e-value of the result of Π on I is ∞. So, $\mathsf{e} \geq \infty$ and thus $\mathsf{e} = \infty$.

Now suppose $cur \neq n+1$ but π_{cur} is of the third type and so $\pi_{cur} = \pi^\infty$. Then, since by Lemma 3.22 $\Pi^{(cur)}$ is valid, by Definition 3.13 $\pi_{cur}(\mu, C) = \pi^\infty(\mu, C)$ is defined where C is the result of $\Pi^{(cur-1)}$ on I. So, by Definition 3.11 the e-value of $C = result(I, \Pi^{(cur-1)})$ is ∞, that is, $\mathsf{e}_{\text{old}} = \infty$. Hence, as by Observation 6.32 $\mathsf{e} \geq \mathsf{e}_{\text{old}}$, $\mathsf{e} = \infty$. Thus, in both cases $\mathsf{e} = \infty$. □

Now we can prove that if a round r is not the last round, during r $cur \neq n+1$ and π_{cur} is of the first or of the second type. Since a round is started by one of lines 80 or 81, just before starting r line 76 is executed. If during r $cur \neq n+1$ or π_{cur} is of the third type, by Lemma 6.34 after the execution of line 76 and just before starting the round r $\mathsf{e} = \infty$. Hence, as e does not change during the rounds, e remains ∞ and so before the next execution of the body of the "while" loop of line 72, the condition checked on this line

is false. Consequently, the algorithm exists the "while" loop and hence there is no round after r. So, if there is a round after r, $cur \neq n+1$ and π_{cur} is of the first type or of the second type.

Lemma 6.35 *If a round r is not the last round, during r $cur \neq n+1$ and π_{cur} is of the first type or of the second type.* □

We now discuss the leaf-costs of rounds of each category, separately. First, we explain the idea of the proof of the upper bound on the sum of leaf-costs of all null rounds. Considering a maximal sequence Γ of consecutive rounds of the same state, by Observation 6.33 either all null rounds in Γ are started by calling Visit$^-$(root) or all are started by calling Visit$^+$(root). So, as we will prove, the amount by which VBalance is increased during the execution all rounds in Γ is the difference between the total leaf-cost of null rounds and the total leaf-cost of other rounds of Γ. We first show that the value of e is never decreased and we conclude that the value of *cur* is not decreased. Then, prove an upper bound on the number of different values that can be assigned to *cur*, that is, the number of steps of Π that might become active. These results will be used to estimate the number of maximal sequences that we talked about. Then, in Lemma 6.38 we prove an upper bound on |VBalance| after each round and we obtain an upper bound on the amount by which VBalance is increased during the execution of rounds in each of the aforementioned maximal sequences.

Lemma 6.36 *After initializing the variables, neither* e *nor* cur *is ever decreased.*

Proof. We first prove that e is not decreased and then we conclude that *cur* is not decreased either. Since after initialization e is modified only by lines 2 and 6, we prove that by executing each of these two lines e is not decreased. First we consider line 2. Just before an execution of this line, by the condition checked on line 1 M[root] > e and this line sets e := M[root]. Hence, line 2 increases e. Also, line 6 sets e := e$_+$ and thus it increases e, as well. So, both lines increase e. Therefore, e is never decreased and thus a complete prefix of Π does not become incomplete. Hence every time that *cur* changes, it accepts a bigger value until all prefixes of Π are complete and *cur* is set equal to its maximum value $n+1$. □

Lemma 6.37 *The number of different integers less than $n+1$ assigned to cur throughout the execution of the algorithm is at most $2|Visited(\mathcal{P})|+2$.*

Proof. Defining $start(x_1), \ldots, start(x_m)$ to be the sequence of all distinct integers less than $n+1$ assigned to *cur* in increasing order and e_i to be the e-value of the result of $\Pi^{(x_i)}$ on I, for every $1 \leq i \leq m$, we prove that the e_i's are distinct. Then, we use Lemma 3.26 to prove that there are not more than $2|Visited(\mathcal{P})|+2$ possible values for the e_i's and in this way the lemma is proved. We define t_i as the first time such that $cur^{(t_i)} = x_i$, for every $1 \leq i \leq m$. Since e_i is the result of $\Pi^{(x_i)}$ on I and $cur^{(t_i)} = x_i$, $\mathsf{e}_{\text{goal}}^{(t_i)} = \mathsf{e}_i$ and so by Observation 6.32 $\mathsf{e}^{(t_i)} < \mathsf{e}_i$, for every i, $1 \leq i \leq m$. Also, as $cur^{(t_i)} = x_i$, at time t_i $\Pi^{(x_i)}$ is the shortest incomplete prefix of Π, for every i, $1 \leq i \leq m$. Consequently, as by assumption $x_{i-1} < x_i$, at time t_i $\Pi^{(x_{i-1})}$ is complete, for every i, $1 < i \leq m$. Hence, as e_{i-1} is the result of $\Pi^{(x_{i-1})}$ on I, $\mathsf{e}_{i-1} \leq \mathsf{e}^{(t_i)}$, for every i, $1 < i \leq m$. So, $\mathsf{e}_{i-1} \leq \mathsf{e}^{(t_i)} < \mathsf{e}_i$, for every i, $1 < i \leq m$. Therefore, $\mathsf{e}_1 < \mathsf{e}_2 < \ldots < \mathsf{e}_m$ and thus the e_i's are distinct. Since e_i is the e-value of the result of $\Pi^{(x_i)}$ on I, by Lemma 3.26 e_i is in the set $\{-\infty, \infty\}$ or there is an object of value $\langle \mathsf{e}_i \rangle$ in $Visited(I, \Pi)$, for every i, $1 \leq i \leq m$. Hence, e_i is in the set $\mathcal{S} = \{-\infty, \infty\} \cup \{o \in Visited(I, \Pi) \mid \mu(o), \mu(o)+\}$, for every i, $1 \leq i \leq m$. As a result, since the e_i's are distinct, $m \leq |\mathcal{S}| = 2|Visited(I, \Pi)| + 2$ and so as by choice of Π $Visited(I, \Pi) \subseteq Visited(\mathcal{P})$, $m \leq 2|Visited(\mathcal{P})| + 2$. □

Before proving our upper bound on $|\mathsf{VBalance}|$, we remind the reader that k was defined as the number of leaves of Q.

Lemma 6.38 *Before and after each round, $|VBalance| \leq k$.*

Proof. The proof is similar to the proof of Lemma 6.25. Since in each execution of the body of the "while" loop of line 72 at most one round is executed and $\mathsf{VBalance}$ is only modified after that round, it suffices to prove that before and after each execution of the body of this "while" loop, $|\mathsf{VBalance}| \leq k$. We first prove that before the first execution of the body of this "while" loop $|\mathsf{VBalance}| \leq k$ and then we prove that after each execution of this body the above inequality remains satisfied. Before starting the "while" loop of line 72 due to initialization of the variable $\mathsf{VBalance}$ just before line 72, $|\mathsf{VBalance}| = 0 \leq k$.

Now we consider the times *bef* and *aft* just before and just after an execution of the body of the "while" loop, we suppose $|\mathsf{VBalance}^{(bef)}| \leq k$, and we prove that $|\mathsf{VBalance}^{(aft)}| \leq k$.

We consider two cases $\mathsf{VBalance}^{(bef)} > 0$ and $\mathsf{VBalance}^{(bef)} \leq 0$. In the first (second) case, by the condition checked on line 79 Visit$^+$(root) (Visit$^-$(root)) is executed and after this execution the algorithm decreases (increases) VBalance by L^+(root) (L^-(root), respectively). Hence, as by assumption $0 < \mathsf{VBalance}^{(bef)} \leq k$ ($-k \leq \mathsf{VBalance}^{(bef)} \leq 0$, respectively) and by Lemmas 6.23 and 6.24 L^-(root) and L^+(root) are at most $|\mathit{leaves}(Q[\text{root}])| = k$, $-k \leq \mathsf{VBalance}^{(aft)} \leq k$ and so $|\mathsf{VBalance}^{(aft)}| \leq k$. Therefore, the lemma is correct. □

Before evaluating the total leaf-cost of null rounds, we note that by Lemma 6.35, every round except the last round has an active step of the first or of the second type and thus every round except the last round has a negative or a positive state.

Lemma 6.39 *The total leaf-cost of null rounds except the last round is at most the total leaf-cost of non-null rounds plus* $4k(|\mathit{Visited}(\mathcal{P})| + 1)$.

Proof. We consider the sequence of all rounds except the last round and we put all consecutive rounds with the same state in one group so that the state of rounds in each group differs from the state of rounds in its next group. Then, we evaluate the total leaf-cost of null rounds in each group and finally we find the sum of these totals. Suppose $R_1, \ldots, R_m$ is the sequence of these groups and x_i and y_i are the total leaf-cost of all null rounds and the total leaf-cost of all non-null rounds of the group R_i, respectively, for every i, $1 \leq i \leq m$.

Given an i, $1 \leq i \leq m$, we now prove that $x_i \leq y_i + k$. We consider two cases depending on whether all rounds in R_i have a negative state or they all have a positive state. First we consider the case in which all rounds of R_i have a negative state. Then, by Observation 6.33 null rounds (non-null rounds) of the group R_i are those run by line 80 (line 81) of the algorithm. After every execution of line 80 the algorithm decreases VBalance by $L^+(v)$, that is, by the leaf-cost of the round started by line 80 (Lemma 6.4). Also, after every execution of line 81 the algorithm increases VBalance by $L^-(v)$, that is, by the leaf-cost of the round started by line 81 (Lemma 6.3). Hence, defining *bef* and *aft* as the time just before the first round of R_i and the time just after the last round of R_i, $\mathsf{VBalance}^{(aft)} - \mathsf{VBalance}^{(bef)}$ is the difference between the total leaf-cost of rounds of R_i started by executions of line 81 and the total leaf-cost of rounds of R_i started by executions of line 80, that is, the difference between total leaf-costs of null and non-null rounds of R_i.

Therefore, $x_i - y_i \leq |\mathsf{VBalance}^{(aft)} - \mathsf{VBalance}^{(bef)}|$. So, as by Lemma 6.38 $|\mathsf{VBalance}^{(aft)}|$ and $|\mathsf{VBalance}^{(bef)}|$ both are at most k, $x_i - y_i \leq 2k$ and thus $x_i \leq y_i + 2k$.

Now we evaluate the total leaf-cost of all null-rounds except the last round. According to what we proved, the total leaf-cost of all null rounds of all R_i's is $\sum_{i=1}^{m} x_i \leq \sum_{i=1}^{m}(y_i+2k) = 2km + \sum_{i=1}^{m} y_i$. Also, $\sum_{i=1}^{m} y_i$ is the total leaf-cost of all non-null rounds. Hence, if we prove that $m \leq 2(|\mathit{Visited}(\mathcal{P})| + 1)$ the lemma is proved. Consider the first round r_i in R_i and suppose a_i is the value of *cur* during r_i, for every i, $1 \leq i \leq m$. Since we are considering the rounds before the last round, by Lemma 6.35 $a_i \neq n+1$ and π_{a_i} is either of the first type or of the second type. As a result, π_{a_i} is the active step of r_i, for every i, $1 \leq i \leq m$. Also, by assumption the state of r_i differs from the state of r_{i+1} and hence the state of one of the rounds r_i and r_{i+1} is positive and the state of the other one is negative, for every i, $1 \leq i < m$. So, one of π_{a_i} and $\pi_{a_{i+1}}$ is of the first type and the other one is of the second type and thus $a_i \neq a_{i+1}$, for every i, $1 \leq i < m$. Moreover, as by Lemma 6.36 *cur* is never decreased, $a_i \leq a_{i+1}$ and hence as we proved $a_i \neq a_{i+1}$, $a_i < a_{i+1}$, for every i, $1 \leq i < m$. So, the a_i's are distinct integers. But during r_i, $a_i = \mathit{cur}$, for every i, $1 \leq i < m$. Therefore, as the a_i's are distinct, by Lemma 6.37 they number is at most $2|\mathit{Visited}(\mathcal{P})| + 2$. So, $m \leq 2|\mathit{Visited}(\mathcal{P})| + 2$ and hence the lemma is correct, as explained. □

Next we evaluate the total leaf-cost of all wasted rounds. We first in the next lemma discuss the situation in which the state of a round is positive and then we discuss the situation in which a non-null round with a positive state is wasted.

Lemma 6.40 *If $\mathit{cur} \neq n+1$ and π_{cur} is of the second type, then $\mathsf{e} \notin \{-\infty, \infty\}$, $\mathsf{e} = \mathsf{e}_{\text{old}}$, and $\mathsf{e}_{\text{goal}} = \mathsf{e}+$.*

Proof. We first prove that $\mathsf{e}_{\text{goal}} = \mathsf{e}_{\text{old}}+$, then we show that $\mathsf{e} = \mathsf{e}_{\text{old}}$, and finally we prove that $\mathsf{e} \notin \{-\infty, \infty\}$. Since π_{cur} is of the second type, $\pi_{\mathit{cur}} = \pi_T^+$, for a sub-intersection tree T of Q. Also, as by Observation 6.30 $\Pi^{(\mathit{cur})}$ is valid and π_{cur} is the last step of $\Pi^{(\mathit{cur})}$, by Definition 3.13 $\pi_{\mathit{cur}}(\mu, C)$ is defined where C is the result of $\Pi^{(\mathit{cur}-1)}$ on I. Moreover, by Definition 3.13 the result of $\Pi^{(\mathit{cur})}$ on I equals $\pi_{\mathit{cur}}(\mu, C) = \pi_T^+(\mu, C)$ and so e_{goal} is the e-value of $\pi_T^+(\mu, C)$. Furthermore, since C is the result of $\Pi^{(\mathit{cur}-1)}$ on I, by definition e_{old} is the e-value of C. Consequently, by Definition 3.10 $\mathsf{e}_{\text{goal}} = \mathsf{e}_{\text{old}}+$. Since by Observation 6.32,

$\mathsf{e} < \mathsf{e}_{\text{goal}} = \mathsf{e}_{\text{old}}+$, by Observation 2.30 $\mathsf{e} \leq \langle \mathsf{e}_{\text{old}}+ \rangle = \mathsf{e}_{\text{old}}$. Hence, since by Observation 6.32 $\mathsf{e} \geq \mathsf{e}_{\text{old}}$, $\mathsf{e} = \mathsf{e}_{\text{old}}$. Also, since we proved $\pi_{cur}(\mu, C) = \pi_T^+(\mu, C)$ is defined, by Definition 3.10 every leaf of T has an object of value $\mathsf{e}_{\text{old}} = \mathsf{e}$. So, by Definition 2.5 $\mathsf{e} \notin \{-\infty, \infty\}$. □

Now we prove that if π_{cur} is of the second type, once UpdateE is executed and E is assigned the value e, E remains equal e until *cur* changes and so there is no wasted round until *cur* changes. In this way we will be able to evaluate an upper bound on the total leaf-cost of a number of wasted rounds with the same active step.

Lemma 6.41 *Consider two times t_1 and t_2, $t_2 > t_1$, such that at time t_1 $cur \neq n+1$ and π_{cur} is of the second type. Also, we suppose t_2 is not during an execution of line 78, $cur^{(t_1)} = cur^{(t_2)}$, and $\mathsf{e}^{(t_1)} = \mathsf{E}^{(t_1)}$. Then, $\mathsf{e}^{(t_2)} = \mathsf{E}^{(t_2)}$.*

Proof. We consider the first time *aft* after time t_1 such that $\mathsf{e}^{(aft)} \neq \mathsf{e}^{(t)}$ (or ∞ if such a time does not exist). First we prove that $t_2 < aft$ and then we show the correctness of the lemma.

We now prove that at time *aft* and after time *aft* $cur \neq cur^{(t_1)}$ and so $t_2 < aft$. At time t_1, by Lemma 6.40 $\mathsf{e}_{\text{goal}} = \mathsf{e}+$. Also, after this time if e changes e is increased (Lemma 6.36). Hence, $\mathsf{e}^{(aft)} > \mathsf{e}^{(t_1)} \geq \langle \mathsf{e}^{(t_1)} \rangle = \langle \mathsf{e}^{(t_1)}+ \rangle$ and so $\mathsf{e}^{(aft)} \not\leq \langle \mathsf{e}^{(t_1)}+ \rangle$. Consequently, by Observation 2.30 $\mathsf{e}^{(aft)} \not< \mathsf{e}^{(t_1)}+ = \mathsf{e}_{\text{goal}}^{(t_1)}$ and so $\mathsf{e}^{(aft)} \geq \mathsf{e}_{\text{goal}}^{(t_1)}$. Therefore, since $\mathsf{e}_{\text{goal}}^{(t_1)}$ is the e-value of the result of $\Pi^{(cur^{(t_1)})}$ on I, at time *aft* $\Pi^{(cur^{(t_1)})}$ is complete. Hence as by definition $cur^{(aft)}$ is such that at time *aft* $\Pi^{(cur^{(aft)})}$ is incomplete, $cur^{(aft)} \neq cur^{(t_1)}$. So, as by Lemma 6.36 *cur* is never decreased, $cur^{(aft)} > cur^{(t_1)}$ and after time *aft* *cur* always remains greater than $cur^{(t_1)}$. Hence, $t_2 < aft$ because by assumption $cur^{(t_1)} = cur^{(t_2)}$.

We now prove that $\mathsf{e}^{(t_2)} = \mathsf{E}^{(t_2)}$. Since $t_2 < aft$, at any time between the times t_1 and t_2 (inclusive), $\mathsf{e} = \mathsf{e}^{(t_1)}$. So, as t_2 is not during an execution of line 78, it suffices to show that E is only modified during executions of line 78 and after each execution of line 78 the equality $\mathsf{E} = \mathsf{e}$ holds. The variable E is modified only by the procedure UpdateE and every execution of UpdateE is during an execution of line 78. Also, every time that line 78 is executed, by Lemma 5.59 all invariants are true and all variables are well-valued. Hence, before each execution of line 78 the preconditions of UpdateE hold. Therefore, after each execution of line 78, by Lemma 5.55 the postconditions of UpdateE hold and thus $\mathsf{e} = \mathsf{E}$.

Hence, except the times during executions of line 78, at any moment between t_1 and t_2 (inclusive) $\mathsf{e} = \mathsf{E}$. So, as t_2 is not during an execution of line 78, $\mathsf{e}^{(t_2)} = \mathsf{E}^{(t_2)}$. □

Lemma 6.42 *The total leaf-cost of all wasted rounds except the last round is at most* $4k(|\mathit{Visited}(\mathcal{P})| + 1)$.

Proof. For every integer i less than $n+1$ assigned to *cur* before the last round, we give an upper bound on the total leaf-cost of wasted rounds during which $\mathit{cur} = i$. Then, we use Lemma 6.37 to prove the upper bound claimed by the lemma. For every integer i assigned to *cur*, we define x_i as the sum of leaf-costs of wasted rounds before the last round during which $\mathit{cur} = i$.

We consider an integer i less than $n+1$ assigned to *cur* before the last round such that $x_i > 0$ and we prove $x_i \leq 2k$. As $x_i > 0$, there is at least one round before the last round during which $\mathit{cur} = i$. Consider time t just before starting the first round during which $\mathit{cur} = i$. We first prove that π_i is of the second type. Then, we show that the sum of leaf-costs of rounds after time t and before the first execution of line 78 after time t is at most $2k$. Finally we use Lemma 6.41 to complete the proof of the lemma.

Now let us prove that π_i is of the second type. Since by choice of i there is at least one round other than the last round of the algorithm during which $\mathit{cur} = i$ and the round of time t is the first such round, the round of time t (meaning the round starting just after time t) is not the last round. Hence, as during the round of time t $\mathit{cur} = i$, by Lemma 6.35 π_i is of the first or the second type. Also, if π_i is of the first type, by definition there is no wasted round during which $\mathit{cur} = i$ and hence $x_i = 0$. So, as by assumption $x_i > 0$, π_i is not of the first type and thus π_i is of the second type.

Now we give an upper bound on the total leaf-cost of rounds after time t and before the first execution of line 78 after t and then we prove that $x_i \leq 2k$. By Lemma 6.25 $\mathsf{CBalance}^{(t)} \geq -k$. Also, after every negative-visit (positive-visit) to the root in line 81 (line 80), CBalance is increased by $L^-(\text{root})$ (by $L^+(\text{root})$) and by Lemma 6.3 (Lemma 6.4) the leaf-cost of a negative-visit (positive-visit) to the root is $L^-(\text{root})$ (is $L^+(\text{root})$). Hence, after every round the algorithm increases CBalance by the leaf-cost of that round. Also, the algorithm does not decrease CBalance unless UpdateE is executed by line 78. Consequently, if the leaf-cost of rounds after time t and before the first execution of line 78 after t is greater

than $2k$, after an execution of line 81 or line 80 CBalance becomes greater than k. But by Lemma 6.25 this is not possible. So, if there is no execution of line 78 after time t, $x_i \leq 2k$. Now suppose this is not the case and consider time *aft* just after the first execution of line 78 after time t.

We now consider two cases $i = cur^{(\text{aft})}$ and $i \neq cur^{(\text{aft})}$ and in each case we prove that there is no wasted round with the active step $\pi_{cur^{(t)}}$ after time *aft*. Then, we conclude that $x_i \leq 2k$. Recall that by choice of t $cur^{(t)} = i$. If $i \neq cur^{(\text{aft})}$, as by Lemma 6.36 *cur* is never decreased, after time *aft* *cur* never becomes equal to i again. Hence, there is no round with the active step $\pi_{cur^{(t)}}$ after time *aft*. Now consider the other case, in which $i = cur^{(\text{aft})}$. In this case, since by the postcondition of UpdateE at time *aft* $\mathsf{e} = \mathsf{E}$, by Lemma 6.41 there is no round after time *aft* during which $cur = cur^{(t)}$ and $\mathsf{e} \neq \mathsf{E}$. Hence, there is no wasted round with the active step $\pi_{cur^{(t)}}$ after time *aft*. So, in either case we proved there is no wasted round with the active step $\pi_{cur^{(t)}}$ after time *aft*. Hence, as by choice of t the round starting just after time t is the first round with the active step $\pi_i = \pi_{cur^{(t)}}$, x_i is at most the sum of leaf-costs of rounds between the times t and *aft* and we proved this sum is at most $2k$. So, $x_i \leq 2k$.

Defining $i_1, \ldots, i_m$ as all distinct integers assigned to *cur* before the last round, to find the sum of leaf-costs of all wasted rounds, we evaluate $\sum_{j=1}^{m} x_{i_j}$. If $x_{i_j} > 0$, by definition there is a round before the last round during which $cur = i_j$ and so by Lemma 6.35 $i_j < n + 1$, for every j, $1 \leq j \leq m$. Thus, if $x_{i_j} > 0$, as we proved, $x_{i_j} \leq 2k$, for every j, $1 \leq j \leq m$. In addition, it follows from Lemma 6.37 that the number of the i_j's less than $n + 1$ is at most $2|\mathit{Visited}(\mathcal{P})| + 2$. So, the total leaf-cost of all wasted rounds is at most $2k(2|\mathit{Visited}(\mathcal{P})| + 2) = 4k(|\mathit{Visited}(\mathcal{P})| + 1)$. □

Now we evaluate the sum of the leaf-costs of negative and positive rounds. We use an idea similar to the idea of defining the variables $\chi_l^-(v)$ and $\chi_l^+(v)$, for nodes v in Section 6.1. For every node v we define three variables $\chi_{\text{vis}}^-(v)$, $\chi_{\text{vis}}^+(v)$, and $\chi_e(v)$. Given a node v, $\chi_{\text{vis}}^-(v)$ and $\chi_{\text{vis}}^+(v)$ are called the χ_{vis}-*variables of* v, and $\chi_e(v)$ is the χ_e-*variable of* v. Also, for every $\mathcal{P}$-visited object o we define a variable $\chi_{\text{obj}}(o)$, called the χ_{obj}-*variable of* o. All these new variables are called *charging variables* and all are initially zero. We will change these variables such that when a negative (a positive) round r other the last round starts, $\chi_{\text{vis}}^-(\text{root})$ ($\chi_{\text{vis}}^+(\text{root})$, respectively) is increased by the leaf-cost of that round. After

this, we change values of charging variables so that their sum does not change. Hence, at the end of the execution of the algorithm the sum of charging variables is the sum of leaf-costs of all positive and negative rounds except the last round.

Similar to what we did for updating variables $\chi_l^-(v)$ and $\chi_l^+(v)$, for nodes v, while visiting nodes v, we decrease χ_{vis}-variables of v and we increase χ_{vis}-variables of children of v. Here we want to know how many times leaves should be visited before $\Pi^{(cur)}$ becomes complete. We now explain how we modify charging variables so that we can find this number of times in each of the two cases in which π_{cur} is of the first type and π_{cur} is of the second type.

First consider the case in which π_{cur} is of the form π_T^-, for some T. If $\mathsf{M}[\text{root}] \geq \mathsf{e}_{\text{goal}}$, as we will show in Lemma 6.43, the procedure Update$\mathsf{C}$ sets e equal to $\mathsf{M}[\text{root}]$ and hence $\Pi^{(cur)}$ becomes complete. So, we give an upper bound on the number of times that leaves should be visited before $\mathsf{M}[\text{root}]$ becomes at least as large as e_{goal}. It can be proved that if $\mathsf{M}[l] \geq \mathsf{e}_{\text{goal}}$, for every leaf l of T, then $\mathsf{M}[v] \geq \mathsf{e}_{\text{goal}}$, for every node v of T, and hence $\mathsf{M}[\text{root}] \geq \mathsf{e}_{\text{goal}}$. Motivated by this fact, whenever $\pi_{cur} = \pi_T^-$, for some T, T is called *the active tree* and we say a node v of Q is an *active node* if v is in T and $\mathsf{M}[v] < \mathsf{e}_{\text{goal}}$; otherwise v is *inactive*. For each visit to an active node v, we hope that the algorithm increases $\mathsf{M}[v]$ and in this way $\mathsf{M}[u]$, for all ancestors u of v, is increased and ultimately $\mathsf{M}[\text{root}]$ becomes as large as e_{goal}. In any negative round before the last round, if the algorithm starts visiting an active child u of v when visiting a node v, we decrease $\chi_{\text{vis}}^-(v)$ by an amount and we increase $\chi_{\text{vis}}^-(u)$ by the same amount and in this way ultimately almost all amounts stored in χ_{vis}-variables of internal nodes are moved to χ_{vis}-variables of active leaves of T. Also we will give an upper bound on the number of times that an active leaf might be visited before it becomes inactive. In this way, we will have an upper bound on the sum of leaf-costs of all rounds except the last round.

The approach for the case in which $\pi_{cur} = \pi_T^+$, for some T, is similar. In such cases, the states of all rounds are positive and so by definition there are no negative rounds. Hence we are considering just positive rounds as we are evaluating the total leaf-cost of just negative and positive rounds. So, by the definition of positive rounds, we may suppose $\mathsf{E} = \mathsf{e}$. If $\mu(\mathsf{W}[\text{root}]) = \langle \mathsf{e}_{\text{goal}} \rangle$, as we will show in Lemma 6.44, the procedure Update$\mathsf{C}$ updates e so that $\Pi^{(cur)}$ becomes complete. Also, one can prove that if for every

leaf l of T, $\mu(\mathsf{W}[l]) = \langle \mathsf{e}_{\text{goal}} \rangle$ then for every node v of T, $\mu(\mathsf{W}[v]) = \langle \mathsf{e}_{\text{goal}} \rangle$ and hence $\mu(\mathsf{W}[\text{root}]) = \langle \mathsf{e}_{\text{goal}} \rangle$. So, as in the case of the negative steps, whenever $\pi_{cur} = \pi_T^+$, for some T, T is called *the active tree* and we say a node v of Q is an *active node* if v is in T and $\mu(\mathsf{W}[v]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$; otherwise v is *inactive*. Then, whenever in a positive round before the last round, while visiting a node v, the algorithm starts visiting an active child u of v, we decrease $\chi^+_{\text{vis}}(v)$ by an amount and we increase $\chi^+_{\text{vis}}(u)$ by the same amount. We now explain more precisely our method for changing values of charging variables.

Definition 6.1 *A* visit modification *is applied when in a negative (a positive) round other than the last round an active node starts being visited. In this modification, we increase $\chi^-_{\text{vis}}(v)$ by $W^-_L(v)L^-(v)$ (increase $\chi^+_{\text{vis}}(v)$ by $W^+_L(v)L^+(v)$) and if v is not the root we decrease $\chi^-_{\text{vis}}(u)$ by $W^-_L(v)L^-(v)$ (we decrease $\chi^+_{\text{vis}}(u)$ by $W^+_L(v)L^+(v)$), where u is the parent of v.*

Now we formally prove that if we apply all visit modifications to χ_{vis}-variables of nodes, we can use the sum of charging variables to evaluate the sum of leaf-costs of positive and negative rounds. We will also introduce some other changes to values of charging variables that do not change their sum and hence at the end of the algorithm the sum of these variables is the sum of leaf-costs of negative and positive rounds. A modification of a number of charging variables that does not modify the sum of integers stored in charging variables is called a *sum-preserving modification*. As is clear from the definition of visit modifications, the only modifications that increase χ_{vis}-variables of an internal node v, are those that are applied at the times just after starting a visit to v. After that, during a visit to v, other visit modifications only may decrease χ_{vis}-variables of v.

We now prove that when a negative or a positive round other than the last round starts, the root is active and then using this fact we will prove that when the algorithm starts a negative or a positive round other than the last round, a visit modification is applied and the sum of charging variables is increased by the leaf-cost of the round being started.

Lemma 6.43 *When starting a negative round the root is active.*

Proof. We first prove that a negative round r has an active tree containing the root and after that we show that when r starts $\mathsf{M}[\text{root}] < \mathsf{e}_{\text{goal}}$. Then, by definition it is proved that

the root is active. As r is a negative round, by definition when r starts $cur \neq n+1$ and π_{cur} is of the first type, that is, $\pi_{cur} = \pi_T^-$, for a sub-union tree T of $N(Q)$. Then T is the active tree and thus as by Definition 2.18 T contains the root, the root is in the active tree.

Assume to the contrary that $\mathsf{M}[\text{root}] \geq \mathsf{e}_{\text{goal}}$. Then, as by Observation 6.32 $\mathsf{e} < \mathsf{e}_{\text{goal}}$, $\mathsf{e} < \mathsf{M}[\text{root}]$. So, as $\mathsf{M}[\text{root}]$ is not a changing variable of UpdateC and by Lemma 6.36 e is not decreased, before executing UpdateC on line 76 the inequality $\mathsf{e} < \mathsf{M}[\text{root}]$ held. So, due to the condition checked on line 1, the assignment on line 2 is executed and e is set equal to $\mathsf{M}[\text{root}]$ and after that e is not decreased. Consequently, as $\mathsf{M}[\text{root}] \geq \mathsf{e}_{\text{goal}}$ after executing UpdateC $\mathsf{e} \geq \mathsf{e}_{\text{goal}}$ and this contradicts Observation 6.32. So, the lemma is correct. □

Lemma 6.44 *When starting a positive round before the last round, the root is active.*

Proof. Using an argument similar to that used in the previous lemma we prove the lemma by showing that a positive round r other than the last round has an active tree containing the root and that when r starts $\mu(\mathsf{W}[\text{root}]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$. By definition of the positive rounds, when r starts $cur \neq n+1$ and $\pi_{cur} = \pi_T^+$, for a sub-intersection tree T of Q. Then, by Definition 2.18 T contains the root and by definition T is the active tree.

Now we prove that $\mu(\mathsf{W}[\text{root}]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$. Assume to the contrary that $\mu(\mathsf{W}[\text{root}]) = \langle \mathsf{e}_{\text{goal}} \rangle$. Then, as by Lemma 6.40 $\mathsf{e} \mathrel{+}= \mathsf{e}_{\text{goal}}$, $\mu(\mathsf{W}[\text{root}]) = \mathsf{e}$. We now discuss the situation while executing UpdateC before starting the round. Since $\mathsf{e} \mathrel{+}= \mathsf{e}_{\text{goal}}$, e is a main value. Hence, Block 4 is not executed as otherwise line 6 would set e equal to a skirted value. Thus, while executing line 4 the condition checked on this line was false and so at that time one of the statements $\mu(\mathsf{W}[\text{root}]) = \mathsf{E} = \mathsf{e}$ or $\mathsf{e} \notin \{-\infty, \infty\}$ was false. Since Block 4 is not executed, after checking the condition checked on line 4 UpdateC has not changed any variable. Thus, before starting the round one of the statements $\mu(\mathsf{W}[\text{root}]) = \mathsf{E} = \mathsf{e}$ or $\mathsf{e} \notin \{-\infty, \infty\}$ is false. We now show that both these statements are true and obtain a contradiction: As the round is positive, the equality $\mathsf{E} = \mathsf{e}$ holds. Also, by assumption $\mu(\mathsf{W}[\text{root}]) = \mathsf{e}$ and by Lemma 6.40 $\mathsf{e} \notin \{-\infty, \infty\}$. Hence, both statements are true and so the assumption that $\mu(\mathsf{W}[\text{root}]) = \langle \mathsf{e}_{\text{goal}} \rangle$ was false. Therefore, as the root is in the active tree, the root is active. □

In the next lemma we formally prove that the sum of charging variables at the end of the algorithm can be used to evaluate the sum of leaf-costs of negative and positive rounds.

Lemma 6.45 *Suppose we apply all visit modifications that we have defined and also suppose except these changes every change that we make to values of charging variables is sum-preserving. Then, at the end of the algorithm, the total leaf-cost of all negative rounds and positive rounds except the last round is the sum of the charging variables.*

Proof. We first consider the effect of visit modifications and then we discuss other modifications. By definition, every visit modification is during a negative or during a positive round. At the beginning of every negative round before the last round by Lemma 6.43 the root is active and hence we increase $\chi^-_{\text{vis}}(\text{root})$ by $W^-_L(\text{root})L^-(\text{root})$. So, as by Definition 2.28 $W^-_L(\text{root}) = 1$, at the beginning of every negative round before the last round we increase $\chi^-_{\text{vis}}(\text{root})$ by $L^-(\text{root})$. Also, a negative round is started by calling $\text{Visit}^-(\text{root})$ and so by Lemma 6.3 its leaf-cost is $L^-(\text{root})$. Therefore, just after starting a negative round other than the last round we are increasing the sum of charging variables by the leaf-cost of that round. A similar argument, by replacing the occurrences of "negative", "Lemma 6.43", "$\chi^-_{\text{vis}}(\text{root})$", "$W^-_L(\text{root})$", "Lemma 6.3", and "$L^-(\text{root})$" with "positive", "Lemma 6.44", "$\chi^+_{\text{vis}}(\text{root})$", "$W^+_L(\text{root})$", "Lemma 6.4", and "$L^+(\text{root})$", respectively, proves that just after starting a positive round before the last round we increase the sum of charging variables by the leaf-cost of that round. After these changes, during a round, every time that an active node v is visited, v is not the root and so v has a parent u. Then, we might increase $\chi^-_{\text{vis}}(v)$ (or $\chi^+_{\text{vis}}(v)$) by an amount and decrease $\chi^-_{\text{vis}}(u)$ (or $\chi^+_{\text{vis}}(u)$) by the same amount. So, overall, by applying all visit modifications the sum of charging variables is increased by the sum of leaf-costs of all negative and positive rounds except the last round. Moreover, aside from visit modifications, all other modifications are sum-preserving and so they do not change the sum of charging variables. Hence, the lemma is correct. □

Now we define a set of sum-preserving modifications that helps us to evaluate the sum of χ_{vis}-variables of the leaves. Given a leaf l, an object o of l is called a *$\mathcal{P}$-touched object* if o is not the last object of l and the object after o in $\omega(l)$ is $\mathcal{P}$-visited. Given a leaf l, a waiting object o of l with a value less than e is *chargeable* if o is $\mathcal{P}$-visited or $\mathcal{P}$-touched. By

applying the following sum-preserving modifications we will keep χ_{vis}-variables of leaves non-positive, as we will prove.

Definition 6.2 *A* leaf modification *is applied when in a negative (a positive) round other than the last round an active leaf l is visited and has a chargeable object. Then, we increase $\chi_{\text{obj}}(o)$ by $W_L^-(l)$ (by $W_L^+(l)$), where o is the smallest chargeable object of l, and we decrease $\chi_{\text{vis}}^-(l)$ (decrease $\chi_{\text{vis}}^+(l)$) by the same amount.*

The next lemma proves an upper bound on the number of times that $\chi_{\text{obj}}(o)$, for a node o, is increased.

Lemma 6.46 *For every $\mathcal{P}$-visited or $\mathcal{P}$-touched object o of a leaf l, the number of times that Gallop(l) is called and o is the first chargeable object of l is at most $1+\log len(o)$ where $len(o)$ is the length of the $\mathcal{P}$-gap containing o.*

Proof. In the first step of the proof, we show that once o becomes scanned o is non-chargeable until the end of the algorithm. In the second step, defining *bef* as the time at which o becomes the first chargeable object of l, we prove in the one of the first $(1 + \log len(o))$ l-gallops starting after *bef* one of these three events occurs: First, Block 5 is executed. Second, when executing line 12, the condition checked on this line is true. Third, when starting the l-gallop, $\mathsf{Step}[l] \geq len(o)$. Finally, in the third step, we show that when each of these three events occurs, o becomes scanned. In this way the lemma is proved.

We first prove that after o becomes scanned o is non-chargeable until the end of the algorithm. Once o becomes scanned, by Definition 5.1 the inequality $\mu(\mathfrak{fi}[l]) > \mu(o)$ is satisfied and so it follows from Lemma 6.20 that the inequality $\mu(\mathfrak{fi}[l]) > \mu(o)$ remains satisfied until the end of the algorithm. Hence, once o becomes scanned, by Definition 5.1 o remains scanned and thus since by definition o being waiting is a necessary condition for o being chargeable, o does not become chargeable again.

Next we take the second step, as explained at the beginning of the proof. Suppose in each of the first $\log len(o)$ l-gallops starting after *bef* when executing line 12, the condition checked on this line is false and also Block 5 is not executed. Hence, in each of these l-gallops, in the "if" block of line 12 $\mathsf{Step}[l]$ is doubled and then as Block 5 is not executed

$\mathsf{Step}[l]$ is not set equal to 1. Consequently, when executing the $(1 + \log len(o))$th l-gallop starting after *bef*, $\mathsf{Step}[l] \geq len(o)$, that is, the third event occurs.

Finally, as the last step, we show that if when starting an l-gallop after time *bef* o is not scanned and in the l-gallop one of the three events stated above occurs, o becomes scanned. Since after time *bef* by Lemma 6.36 e has been not decreased, the inequality $\mu(o) < \mathsf{e}$ still holds and thus o is still chargeable. We consider the three events, one by one. First we suppose the first event occurs, that is, Block 5 is executed. We first prove that line 17 is executed. Then, we prove that line 17 assigns the index of the biggest object of l with a value at most e to j, and finally we conclude that when exiting the procedure o is scanned. As o is chargeable, $\mu(o) < \mathsf{e}$ and o is waiting, that is, $\mu(o) \geq \mu(\mathfrak{fi}[l])$ (Definition 5.1). Hence, $\mathsf{e} > \mu(\mathfrak{fi}[l])$ and so the condition checked on line 15 is false. Therefore, the binary search of line 17 is executed.

Now we show that after line 17, j is the index m of the biggest object of l with a value less than e. We first prove that $i \leq m$ and $m \leq j$. By the condition checked on line 14, either $j > length(\omega(l))$ or $\mu(\omega(l)[j]) \geq \mathsf{e}$. In the first case l has no object with an index more than j and in the second case as μ is ordered, every object of l with an index more than j has value greater than e. Therefore, in either case, as m is the index of an object less than e, $m \leq j$. Also, since i is the index of $\mathfrak{fi}[l]$ (line 11) and we proved $\mathsf{e} > \mu(\mathfrak{fi}[l])$, it follows from the definition of m that m is not less than i. Now, it follows from the definition of m that m is the biggest integer such that $1 \leq m \leq \text{length}(\omega(l))$ and $\mu(\omega(l)[m]) < \mathsf{e}$ and we proved $m \leq j$ and $i \leq m$. So, m is the biggest integer such that $\max(1, i) \leq m \leq \min(j, \text{length}(\omega(l)))$ and $\mu(\omega(l)[m]) < \mathsf{e}$. Also, since after line 11, i does not change, by Lemma 5.32 $1 \leq i$. Hence, m is the biggest integer such that $i \leq m \leq \min(j, \text{length}(\omega(l)))$ and $\mu(\omega(l)[m]) < \mathsf{e}$. So, as the code of line 17 shows, the binary search of line 17 finds m correctly and assigns it to j.

Now we can prove that o becomes scanned. As $\mu(o) < \mathsf{e}$ and we proved line 17 assigns to j the index of the biggest object of l with a value less than e, after line 17 j is at least the index of o in $\omega(l)$ and thus $\mu(\omega(l)[j]) \geq \mu(o)$. Also, as we proved $m \geq i$, the value assigned to j is at least i. So, the condition checked on line 19 is true and hence line 20 sets $\mathfrak{fi}[l]$ equal to $\omega(l)[j + 1]$. Hence, after line 20 $\mu(\mathfrak{fi}[l]) = \mu(\omega(l)[j + 1]) > \mu(\omega(l)[j]) \geq \mu(o)$. So, after executing line 20 by Definition 5.1 o is scanned.

Now consider the second event, that is, when executing line 12, the condition checked on this line is true. Then by Lemma 6.19 after this l-gallop $\mathrm{fi}[l] = \text{END}$ and hence $\mu(\mathrm{fi}[l]) = \infty > \mu(o)$. So, by Definition 5.1 o becomes scanned during the execution of the l-gallop.

Finally we suppose the third event occurs and so when starting the l-gallop, $\mathsf{Step}[l] \geq \mathit{len}(o)$. If when executing line 12 the condition checked on this line is true then the second event occurs and as we proved o becomes scanned. So suppose this is not the case and so the "if" block of line 12 doubles $\mathsf{Step}[l]$ and sets $\mathsf{Step}[l]$ equal to an integer at least $2\mathit{len}(o)$. Now we first prove that line 13 sets j equal to an integer at least $\mathit{Index}(I, l, o)$ and then using this fact we prove that o becomes scanned.

Now let use prove that after line 13 $j \geq \mathit{Index}(I, l, o)$. Suppose Γ is the $\mathcal{P}$-gap containing o. Then, it follows from Definition 2.26 that if Γ is the first $\mathcal{P}$-gap of l there is no object before Γ; otherwise, the object of l just before Γ is visited by $\mathcal{P}$. Also, since o is chargeable, its value is less than e and hence the value of every object before o in l is less than e. So, by definition every $\mathcal{P}$-visited waiting object before o in $\omega(l)$ is chargeable. Therefore, as o is the first chargeable object of l, every $\mathcal{P}$-visited object before o in $\omega(l)$ is scanned. Thus, as we proved the biggest object of $\omega(l)$ before the first object of Γ (if there is any) is $\mathcal{P}$-visited, the biggest object of l before Γ (if there is any) is scanned. Hence, the number of waiting objects of l before o is at most the number of objects of Γ before o, that is, $\mathit{len}(o) - 1$. Also, due to the assignment on line 11, every object with an index at least i has value of at least $\mu(\mathrm{fi}[l])$. Consequently, by Definition 5.1 every object with an index at least i is waiting. Moreover, every object p with an index m, $i \leq m < \mathit{Index}(I, l, o)$ has an index at least i and is before o. Therefore, any object with an index m, $i \leq m < \mathit{Index}(I, l, o)$ is waiting and is before o. Thus, there are at least $\mathit{Index}(I, l, o) - i$ waiting objects before o in $\omega(l)$. So, as we proved the number of waiting objects of l before o is at most $\mathit{len}(o) - 1$ and $\mathsf{Step}[l]$ is set equal to an integer at least $2\mathit{len}(o)$, $\mathit{Index}(I, l, o) - i \leq \mathit{len}(o) - 1 \leq \mathsf{Step}[l] - 1$. Hence, as line 13 sets $j := i + \mathsf{Step}[l] - 1$, after this line $j = i + \mathsf{Step}[l] - 1 \geq \mathit{Index}(I, l, o)$.

Now, having proved after line 13 $j \geq \mathit{Index}(I, l, o)$, we consider two cases based on Block 5 being executed or not. If this block is executed, the first event occurs and thus, as we proved, o becomes scanned. So, suppose this is not true and thus j is not modified until line 19. Now again we consider two possibilities depending on whether Block 7 is executed or not. First suppose this block is not executed. Then, by the condition checked

on line 19, $j < i$ and so as we proved $j \geq \mathit{Index}(I, l, o)$, $\mathit{Index}(I, l, o) < i$. But, due to line 11, $i = \mathit{Index}(I, l, \mathfrak{fi}[l])$. So, $\mathit{Index}(I, l, o) < \mathit{Index}(I, l, \mathfrak{fi}[l])$ and thus $\mu(o) < \mu(\mathfrak{fi}[l])$. Hence, by Definition 5.1 o is scanned. Now consider the other possibility, that is, Block 7 is executed. Then, line 20 sets $\mathfrak{fi}[l] := \omega(l)[j+1]$. So, after this line $\mathit{Index}(I, l, \mathfrak{fi}[l]) = j + 1 > j \geq \mathit{Index}(I, l, o)$ and hence $\mu(o) < \mu(\mathfrak{fi}[l])$. Therefore, by Definition 5.1 o has become scanned.

Now, we have proved that if in an l-gallop after time *bef* o is not scanned and one of the three events occurs then o becomes scanned. Also, we proved that once o becomes scanned o remains scanned and so o will be not chargeable. Thus, as we proved in one of the first $1 + \log \mathit{len}(o)$ l-gallops after time *bef* one of the three events occurs, the lemma is true. □

We now prove that when visiting an active leaf l, l has a chargeable object and so a leaf modification is applied and χ_{vis}-variables of l become zero. To prove this fact, we first show that when π_{cur} is of the first type, every object of l with a value of at least e_{old} has value of at least e_{goal}. Then, using this fact we prove that if l has no chargeable object $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}_{\text{goal}}$ and this will contradict the fact that l is active, as we will see.

Lemma 6.47 *Suppose $\pi_{cur} = \pi_T^-$, for a sub-union tree T of $N(Q)$, and l is a leaf of T. Then, the value of every object of l with a value of at least e_{old} is at least e_{goal}.*

Proof. To prove the lemma it suffices to suppose that l has an object with a value of at least e_{old} and to prove that the value of the smallest object o of l with a value of at least e_{old} is at least e_{goal}. We first show that $\mathit{hmin}(\mu, C, T) \leq \mu(o)$ and then we prove that $\mathsf{e}_{\text{goal}} = \mathit{hmin}(\mu, C, T)$. As l has an object of value at least e_{old} and o is the smallest of such objects, by Lemma 3.1 $\mathit{find}(I, l, \mathsf{e}_{\text{old}}) = o \neq \text{END}$. The leaf l is in $N(Q)$ because l is in T and T is a sub-union tree of $N(Q)$. Also, by the definition of o, $\mu(o) \geq \mathsf{e}_{\text{old}}$ and by definition e_{old} is the e-value of the result C of $\Pi^{(cur-1)}$ on I. Therefore, by Definition 3.3 o is C-remaining and so by Definition 3.7 $\mathit{hmin}(\mu, C, T)$ is the minimum of $\mu(\mathit{find}(I, v, \mathsf{e}_{\text{old}}))$ over all leaves v of T with $\mathit{find}(I, v, \mathsf{e}_{\text{old}}) \neq \text{END}$. Hence, as we proved $\mathit{find}(I, l, \mathsf{e}_{\text{old}}) \neq \text{END}$, $\mathit{hmin}(\mu, C, T) \leq \mu(\mathit{find}(I, l, \mathsf{e}_{\text{old}})) = \mu(o)$.

Now we prove that $\mathsf{e}_{\text{goal}} = \mathit{hmin}(\mu, C, T)$ and we conclude the correctness of the lemma. As $\pi_{cur} = \pi_T^-$ is the last step of $\Pi^{(cur)}$, by Definition 3.13 $\mathit{result}(I, \Pi^{(cur)}) = \pi_T^-(\mu, \mathsf{C})$

and thus e_{goal} is the e-value of $\pi_T^-(\mu, \mathsf{C})$. So, by Definition 3.8 $\mathsf{e}_{\text{goal}} = \mathit{hmin}(\mu, C, T)$. Consequently, $\mathsf{e}_{\text{goal}} \leq \mu(o)$. □

Lemma 6.48 *Suppose π_{cur} is of the first type. Then, just before visiting an active leaf l, l has a chargeable object.*

Proof. We assume to the contrary that l does not have a chargeable object and we prove that l is not active. We first prove that l does not have any object of value e_{old}. Next, we show that all objects of l with values less than e_{old} are scanned. Afterward, we prove that the inequality $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}_{\text{goal}}$ holds and then we conclude that l is not active. In the following C is the result of $\Pi^{(cur-1)}$ on I and hence its e-value is e_{old}. Also, as π_{cur} is of the first type, $\pi_{cur} = \pi_T^-$, for a sub-union tree T of $N(Q)$ and thus T is the active tree. Furthermore, since $\pi_{cur} = \pi_T^-$ is the last step of $\Pi^{(cur)}$, by Definition 3.13 the result of $\Pi^{(cur)}$ on I is $\pi_T^-(\mu, C)$ and so e_{goal} is the e-value of $\pi_T^-(\mu, C)$.

We now prove that l does not have any object of value e_{old}. If l has an object o of value e_{old}, o is the smallest object of l with a value of at least e_{old} and hence by Lemma 6.47, its value is at least e_{goal}. Thus, as by assumption $\mu(o) = \mathsf{e}_{\text{old}}$, $\mathsf{e}_{\text{goal}} \leq \mathsf{e}_{\text{old}}$, contradicting Observation 6.32 which says $\mathsf{e}_{\text{goal}} > \mathsf{e}_{\text{old}}$. So, l does not have any object of value e_{old}.

We next prove that all objects of l with values less than e_{old} are scanned. We suppose l has an object of value less than e_{old} as otherwise the claim is trivially correct. We first prove that the biggest object o of l with a value at most $\langle \mathsf{e}_{\text{old}} \rangle$ is $\mathcal{P}$-visited. By Definition 3.15, the boundary-set of $(I, l, \langle \mathsf{e}_{\text{old}} \rangle)$ is a subset of $\mathit{Visited}_{cur}(I, \Pi)$ and so it is a subset of $\mathit{Visited}(I, \Pi)$ as by Definition 3.15 $\mathit{Visited}(I, \Pi) = \bigcup_{1 \leq i \leq n} \mathit{Visited}_i(I, \Pi)$. Also, as we supposed l has an object p with a value less than e_{old}, by Observation 2.30 $\mu(p) \leq \langle \mathsf{e}_{\text{old}} \rangle$ and so by Definition 3.14 the biggest object o of l with a value at most $\langle \mathsf{e}_{\text{old}} \rangle$ is in the boundary-set of $(I, l, \langle \mathsf{e}_{\text{old}} \rangle)$. Hence, o is in $\mathit{Visited}(I, \Pi)$ and thus as by the choice of Π $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$, $o \in \mathit{Visited}(\mathcal{P})$.

Now we prove that all objects of l with values less than e_{old} are scanned. Every object with a value at most $\langle \mathsf{e}_{\text{old}} \rangle$ has value at most e_{old} as $\mathsf{e}_{\text{old}} \geq \langle \mathsf{e}_{\text{old}} \rangle$. Also, given an object p of l such that $\mu(p) \leq \mathsf{e}_{\text{old}}$, as we proved there is no object of value e_{old} in $\omega(l)$, $\mu(p) < \mathsf{e}_{\text{old}}$ and thus by Observation 2.30 $\mu(p) \leq \langle \mathsf{e}_{\text{old}} \rangle$. So, the sets of objects of l with values at most e_{old} and with values at most $\langle \mathsf{e}_{\text{old}} \rangle$ are the same. Therefore, as o is the biggest object of l with

a value at most $\langle \mathsf{e}_{\text{old}} \rangle$, o is the biggest object of l with a value at most e_{old}. In addition, $\mu(o) < \mathsf{e}_{\text{old}}$ because we proved l does not have an object of value e_{old}. Furthermore, as we proved, o is $\mathcal{P}$-visited. Consequently, o is a $\mathcal{P}$-object of value less than e_{old}. So, as there is no chargeable object in l, o is not waiting and thus by Definition 5.1 $\mu(o) < \mu(\mathfrak{fi}[l])$. As a result, as o is the biggest object of l with a value at most e_{old}, all objects of l with values less than e_{old} have values less than $\mu(\mathfrak{fi}[l])$ and hence they are are scanned.

Next we show that $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}_{\text{goal}}$ by considering two cases based on l having an object with a value of at least e_{old} or not. Since we proved all objects of l with values less than e_{old} are scanned, if l does not have any object with a value of at least e_{old}, there is no waiting object in $\omega(l)$. Consequently, $\mathfrak{fi}[l]$ is not an object of l as otherwise by Definition 5.1 $\mathfrak{fi}[l]$ is waiting. So, by Definition 5.1 $\mathfrak{fi}[l] = \text{END}$ and hence in this case, $\mu(\mathfrak{fi}[l]) = \infty \geq \mathsf{e}_{\text{goal}}$. Now suppose l has an object of value at least e_{old} and consider the smallest such an object o. Then, by Lemma 6.47 $\mathsf{e}_{\text{goal}} \leq \mu(o)$ and so as by Observation 6.32 $\mathsf{e}_{\text{goal}} > \mathsf{e}$, $\mu(o) > \mathsf{e}$. Also, by Lemma 6.13 before visiting l the precondition of the visit procedures, including Invariant 2, hold. So, as $\mu(o) > \mathsf{e}$, by Invariant 2 o is waiting. Moreover, as we proved all objects with values less than e_{old} are scanned and o is the smallest object with a value of at least e_{old}, all objects smaller than o are scanned. Hence o is the smallest waiting object and thus by Lemma 5.1 $\mathfrak{fi}[l] = o$. Therefore, $\mu(\mathfrak{fi}[l]) = \mu(o) \geq \mathsf{e}_{\text{goal}}$.

We now can observe the following contradiction. Since by precondition 2 of the visit procedures $\mathsf{M}[l]$ is well-valued, by Definition 5.6 $\mathsf{M}[l] = \mu(\mathfrak{fi}[l])$ and thus as we proved $\mu(\mathfrak{fi}[l]) \geq \mathsf{e}_{\text{goal}}$, $\mathsf{M}[l] \geq \mathsf{e}_{\text{goal}}$, contradicting the fact that l is active. Hence, the assumption that l does not have a chargeable object is wrong and so the lemma is correct. □

Before proving that when visiting an active leaf l in a positive round l has a chargeable object, to avoid repetitive arguments, we first prove that whenever the algorithm executes line 75, the speedy leaf has a chargeable object and thus we can use Lemma 6.46 to obtain an upper bound on the number of times that this line is executed. The proof will consider several possibilities and in one of them it is proved that $\mathit{find}(I, \mathit{sl}(Q), \mathsf{e})$ is a $\mathcal{P}$-visited object. In the next lemma, we show that in such situations if the speedy leaf does not have any chargeable object, the condition checked on line 74 is false and so line 75 is not executed.

Lemma 6.49 *Given a leaf l without any chargeable object, if $\mathit{find}(I, l, \mathsf{e})$ is a $\mathcal{P}$-visited object then $\mu(\mathit{fi}[l]) \geq \mathsf{e}$.*

Proof. Defining $o = \mathit{find}(I, l, \mathsf{e})$, after stating some basic facts, we consider two cases based on o being the first object of l or not. As by assumption $\mathit{find}(I, l, \mathsf{e})$ is an object, $\mathit{find}(I, l, \mathsf{e}) \neq \text{END}$ and hence by Lemma 3.1 $o = \mathit{find}(I, l, \mathsf{e})$ is the smallest object of l with a value of at least e. So, $\mu(o) \geq \mathsf{e}$. Also, we suppose $\mathrm{fi}[l] \neq \text{END}$ as otherwise $\mu(\mathrm{fi}[l]) = \infty \geq \mathsf{e}$ and thus the lemma is trivial. Hence, by Definition 5.1 $\mathrm{fi}[l]$ is an object of l.

Now we discuss the two cases listed above. First consider the case in which o is the first object. Then, as $\mathrm{fi}[l]$ is an object of l, by Definition 5.1 $\mu(\mathrm{fi}[l]) \geq \mu(o) \geq \mathsf{e}$ and so the claim is true. Now consider the other case, in which o is not the first object. Consider the object p prior to o in $\omega(l)$. Since, as we proved, o is the smallest object of l with a value of at least e, the value of p is less than e. Hence, since p is the object prior to the $\mathcal{P}$-visited object $o = \mathit{find}(I, l, \mathsf{e})$, p is $\mathcal{P}$-touched and thus as $\mu(p) < \mathsf{e}$, p is chargeable unless p is scanned. So, as by assumption there is no chargeable object, p in scanned and hence by Definition 5.1 $\mu(\mathrm{fi}[l]) > \mu(p)$. Therefore, as by assumption $\mathrm{fi}[l]$ is an object and o is the first object after p, $\mu(\mathrm{fi}[l]) \geq \mu(o) \geq \mathsf{e}$. Thus, the claim of the lemma is correct in this case, as well. □

Lemma 6.50 *Just before an execution of line 75 $\mathit{sl}(Q)$ has a chargeable object.*

Proof. We first prove that when executing line 75 $\mathit{cur} \neq n + 1$ and π_{cur} is not of the third type. Then we show that π_{cur} is not also of the first type and so π_{cur} is of the second type. After that we assume to the contrary that $\mathit{sl}(Q)$ has no chargeable object and we prove that the condition checked on line 74 is not true. This contradiction will prove the lemma.

First we prove that $\mathit{cur} \neq n+1$ and π_{cur} is not of the third type. Since by the condition checked on line 74 $\mu(\mathrm{fi}[\mathit{sl}(Q)]) < \mathsf{e}$, $\mathsf{e} \neq -\infty$ and so e has been changed after initialization. Thus, as after initialization e only is changed by Update$\mathsf{C}$, called only by line 76, at least one time line 76 was executed and after its last execution e was not modified. Hence, as cur is only changed by modifying e, after the last execution of line 76 cur was not changed. Now, assume to the contrary that $\mathit{cur} = n + 1$ or π_{cur} is of the third type. Then, by Lemma 6.34 just after the last execution of line 76 $\mathsf{e} = \infty$. So, as e only changes by line 76,

after that execution e has remained equal to ∞ and hence due to the condition checked on line 72 the procedure should exit the "while" loop. This contradicts the fact that after the last execution of line 76 the algorithm has reached line 75. So, $cur \neq n+1$ and π_{cur} is not of the third type.

We now prove some basic facts and then we show that π_{cur} is not of the first type. Since the procedure Gallop does not modify e, by the condition checked on lines 72 and 74 $\mu(\mathfrak{fi}[sl(Q)]) < \mathsf{e} < \infty$ and so $\mathsf{e} \notin \{-\infty, \infty\}$. Also, by the condition checked on line 73 $\mu(\mathsf{W}[\text{root}]) = \mathsf{e} = \mathsf{E}$. Hence, as by Lemma 5.59 all feature variables of all nodes are well-valued, by Lemma 5.15 there is an eyewitness U for v. Therefore by Definition 5.5 $\mu(\mathfrak{fi}[l]) = \mathsf{e}$, for every leaf l of U. So, since by the condition checked on line 74 $\mu(\mathfrak{fi}[sl(Q)]) < \mathsf{e}$, the speedy leaf is not a leaf of U. Thus, U is a sub-intersection tree of $N(Q)$, as well. We now assume to the contrary that π_{cur} is of the first type and so, $\pi_{cur} = \pi_T^-$, for a sub-union tree T of $N(Q)$. Then, by Lemma 2.19 $|leaves(U) \cap leaves(T)| = 1$. So, consider the member l of $leaves(U) \cap leaves(T)$. Then, as l is a leaf of U, $\mu(\mathfrak{fi}[l]) = \mathsf{e}$. Thus, as by Observation 6.32 $\mathsf{e}_{\text{goal}} > \mathsf{e} \geq \mathsf{e}_{\text{old}}$, l has an object with a value of at least e_{old} and less than e_{goal}. This contradicts Lemma 6.47. So, π_{cur} is not of the first type and hence it is of the second type.

Now we assume to the contrary that the speedy leaf does not have a chargeable object and we obtain a contradiction to the condition checked on line 74 by showing that $\mu(\mathfrak{fi}[sl(Q)]) \geq \mathsf{e}$. Considering the facts that by Definition 3.15 the boundary-set of $(I, l, \mathsf{e}_{\text{old}})$ is a subset of $Visited_{cur}(I, \Pi)$, by Definition 3.15 $Visited_{cur}(I, \Pi) \subseteq Visited(I, \Pi)$, by choice of Π $Visited(I, \Pi) \subseteq Visited(\mathcal{P})$, and by Lemma 6.40 $\mathsf{e} = \mathsf{e}_{\text{old}}$, we may conclude that the boundary-set of $(I, sl(Q), \mathsf{e})$ is a subset of $Visited(\mathcal{P})$.

Now, we consider two cases based on $sl(Q)$ having an object with value of at least e or not and in each case we prove the claim $\mu(\mathfrak{fi}[sl(Q)]) \geq \mathsf{e}$. First suppose there is such an object. Then by Definition 3.14 the smallest object o of $sl(Q)$ with a value of at least e is in the boundary-set of $(I, sl(Q), \mathsf{e})$ and hence o is in $Visited(\mathcal{P})$. But by Lemma 3.1 o equals $find(I, sl(Q), \mathsf{e})$. So, $find(I, sl(Q), \mathsf{e})$ is $\mathcal{P}$-visited and thus by Lemma 6.49 the claim $\mu(\mathfrak{fi}[sl(Q)]) \geq \mathsf{e}$ is true. Now consider the other case, in which $sl(Q)$ does not have any object with a value of at least e. So, all objects of the speedy leaf have values less than e. Therefore, the biggest object p of $sl(Q)$ is the biggest object of $sl(Q)$ with a value at most e and so by Definition 3.14 p is in the boundary-set of $(I, sl(Q), \mathsf{e})$. Hence p is in

Visited($\mathcal{P}$). So, as $\mu(p) < \mathsf{e}$ (because all objects of *sl*(Q) have values less than e) and there is no chargeable object, p is scanned. Therefore, by Definition 5.1 $\mu(\mathrm{fi}[\mathit{sl}(Q)]) > \mu(p)$ while p is the biggest object of *sl*(Q). Hence, $\mathrm{fi}[\mathit{sl}(Q)]$ is not an object of the speedy leaf and so by Definition 5.1 $\mathrm{fi}[\mathit{sl}(Q)]$ equals END. Thus, $\mu(\mathrm{fi}[\mathit{sl}(Q)]) = \infty \geq \mathsf{e}$. Hence in both cases $\mu(\mathrm{fi}[\mathit{sl}(Q)]) \geq \mathsf{e}$ which is a contradiction to the condition checked on line 74. Therefore, the assumption that *sl*(Q) does not have a chargeable object is false. □

We now prove that when visiting an active leaf l in a positive round, l has a chargeable object and hence a leaf modification is applied and decreases $\chi^{+}_{\mathrm{vis}}(l)$.

Lemma 6.51 *Just before visiting an active leaf l in a positive round, l has a chargeable object.*

Proof. We assume to the contrary that l does not have any chargeable object and after proving some basic facts, we show that $\mu(\mathrm{fi}[l]) = \mathsf{e}$. Then, we conclude that l is not active and in this way we obtain a contradiction. In the following C is the result of $\Pi^{(\mathit{cur}-1)}$ on I. Hence, by assumption $\mathsf{e}_{\mathrm{old}}$ is the e-value of C. Since the round is positive, $\mathit{cur} \neq n+1$ and $\pi_{\mathit{cur}} = \pi^{+}_{T}$, for a sub-intersection tree T of Q, and thus T is the active tree.

Now we prove that $\mu(\mathrm{fi}[l]) = \mathsf{e}$. Since $\Pi^{(\mathit{cur})}$ is valid (Observation 6.30) and $\pi_{\mathit{cur}} = \pi^{+}_{T}$ is the last step of $\Pi^{(\mathit{cur})}$, by Definition 3.13 $\pi^{+}_{T}(\mu, C)$ is defined. Also, since l is active l is a leaf of T. Therefore, by Definition 3.10 l has an object o of value $\mathsf{e}_{\mathrm{old}}$. Hence, o is in the smallest object of value at least $\mathsf{e}_{\mathrm{old}}$ in $\omega(l)$ and thus by Lemma 3.1 $\mathit{find}(I, l, \mathsf{e}_{\mathrm{old}}) = o$. Consequently, by Definition 3.15 o is in $\mathit{Visited}_{\mathit{cur}}(I, \Pi)$. So, as by Definition 3.15 $\mathit{Visited}(I, \Pi) = \bigcup_{1 \leq i \leq n} \mathit{Visited}_i(I, \Pi)$ and by assumption $\mathit{Visited}(I, \Pi) \subseteq \mathit{Visited}(\mathcal{P})$, o is in *Visited*($\mathcal{P}$). Also, we proved $\mu(o) = \mathsf{e}_{\mathrm{old}}$ and thus since by Lemma 6.40 $\mathsf{e}_{\mathrm{old}} = \mathsf{e}$, $\mu(o) = \mathsf{e}$. Moreover, by Lemma 6.13 when visiting l the preconditions of the visit procedures hold and so Invariant 2 is true. Hence, as $\mu(o) \not< \mathsf{e}$, by Invariant 2, o is waiting. So by Definition 5.1 $\mu(o) \geq \mu(\mathrm{fi}[l])$. Also, since $o = \mathit{find}(I, l, \mathsf{e}_{\mathrm{old}}) = \mathit{find}(I, l, \mathsf{e})$ and o is $\mathcal{P}$-visited, by Lemma 6.49 $\mu(\mathrm{fi}[l]) \geq \mathsf{e} = \mathsf{e}_{\mathrm{old}} = \mu(o)$. Therefore as we proved $\mu(o) \geq \mu(\mathrm{fi}[l])$, $\mu(\mathrm{fi}[l]) = \mu(o) = \mathsf{e}$.

Now we complete the proof by showing that l is not active. Considering the facts that by the definition of positive rounds $\mathsf{e} = \mathsf{E}$, by Lemma 6.40 $\mathsf{e} \notin \{-\infty, \infty\}$, by precondition 2 of the visit procedure $\mathsf{W}[l]$ is well-valued, and we proved that $\mu(\mathrm{fi}[l]) = \mathsf{e}$, we can conclude from

Definition 5.8 that $\mu(\mathsf{W}[l]) = \mathsf{e}$. So, as by Lemma 6.40 $\mathsf{e}_{\text{goal}} = \mathsf{e}_+$, $\mu(\mathsf{W}[l]) = \mathsf{e} = \langle \mathsf{e}_{\text{goal}} \rangle$ and thus l is not active, contradicting the assumption of the lemma. Therefore, the assumption that l does not have any chargeable object is false. □

Now, let us explain how we evaluate the sum of χ_{vis}-variables after finishing the execution of the algorithm. Using the lemmas we proved above, we will show that χ_{vis}-variables of leaves remain non-positive. Now we introduce a number of sum-preserving modifications to values of charging variables so that we can evaluate the sum of χ_{vis}-variables of internal nodes at the end of the execution of the algorithm. Given an internal node v, by *discharging* v we mean to make both of the following two changes to values of the charging variables: First, if $\chi^-_{\text{vis}}(v) > 0$, to increase $\chi_e(v)$ by $\chi^-_{\text{vis}}(v)$ and then to set $\chi^-_{\text{vis}}(v) := 0$. Second, if $\chi^+_{\text{vis}}(v) > 0$, to increase $\chi_e(v)$ by $\chi^+_{\text{vis}}(v)$ and then to set $\chi^+_{\text{vis}}(v) := 0$. By applying these modifications once in a while, we always will have an upper bound on the values stored in χ_{vis}-variables of internal nodes. Also, it is clear that discharging a node is a sum-preserving modification. When we discharge a node v we say we have performed a *useful discharging* or we have *discharged v usefully* if just before discharging v, one of $\chi^-_{\text{vis}}(v)$ or $\chi^+_{\text{vis}}(v)$ were positive. Now let us explain when we discharge a node of Q.

Definition 6.3 Internal modifications *are defined to be applied in the following way.*

1. *Before the last round we apply the following modifications.*
   (a) *Just before finishing an execution of Visit$^-(v)$, Visit$^+(v)$, for an internal node v, if before this execution v was active and now v is not active any more, we discharge v.*
   (b) *Just before finishing an execution of Block 20, for every internal node v, if before this execution v was active and now v is not active any more, we discharge v.*
   (c) *Every time that* cur *changes, just before changing* cur *we discharge all internal nodes.*
2. *Just after starting the last round we discharge all internal nodes.*

Let us first explain how a χ_{vis}-variable can be increased and then we prove some upper-bounds on values of these variables. Among the modifications we defined so far, it follows

from their definitions that leaf modifications and internal modifications always reduce χ_{vis}-variables. Also, the only visit modifications that may increase χ_{vis}-variables of a node v are those that are applied just after starting a visit to v. Hence, the next observation is true.

Observation 6.52 *The only modifications increasing χ_{vis}-variables of a node v are the visit modifications that are applied just after starting a visit to v.*

We now prove that whenever a node v becomes inactive as soon as the algorithm is not executing Block 20 or visiting v, χ_{vis}-variables of v become zero. To prove this fact, we first discuss the situations in which a node can become inactive.

Lemma 6.53 *An active node v may become inactive only when cur changes or during an execution of one of Visit$^-(v)$, Visit$^+(v)$, or line 78.*

Proof. The definition of a node v being active only depends on the active tree, on the value of e_{goal}, on *cur*, and on variables $\mathsf{W}[v]$ and $\mathsf{M}[v]$ (if v is in $N(Q)$). Also, the active tree and the value of e_{goal} are determined by the value of *cur*. Hence, v may become inactive only when one of *cur*, $\mathsf{W}[v]$, or $\mathsf{M}[v]$ are modified. Moreover, $\mathsf{W}[v]$ and $\mathsf{M}[v]$ are only modified during an execution of Visit$^-(v)$, Visit$^+(v)$, or Update$\mathsf{E}$ and Update$\mathsf{E}$ is called only during an execution of line 78. Consequently, v can become inactive only when *cur* changes or the algorithm is executing Visit$^-(v)$, Visit$^+(v)$, or line 78. □

Now we discuss the situation in which χ_{vis}-variables of nodes become non-positive. According to Lemma 6.53, an active node v can become inactive only when *cur* changes or the algorithm is visiting v or executing line 78 where line 78 is a part of Block 20. Also, before the last round, whenever *cur* changes, just before changing *cur* all internal nodes are discharged and hence χ_{vis}-variables of all nodes become non-positive. In addition, every time that the algorithm finishes an execution of Visit$^-(v)$, Visit$^+(v)$, or Block 20, just before finishing that execution if v has become inactive we discharge v and thus χ_{vis}-variables of v become non-positive. So, once v becomes inactive before the last round, as soon as the algorithm is not visiting v or executing Block 20, χ_{vis}-variables of v become non-positive. Moreover, by Observation 6.52 the only modification increasing χ_{vis}-variables of a

node v is the visit modification that is applied when v starts being visited and by definition that modification is applied only if v is active. Thus, at any moment before the last round that Block 20 is not being executed, χ_{vis}-variables of every inactive internal node that is not being visited are non-positive. Also, whenever a new step becomes active, *cur* changes and, just before changing *cur*, we discharge all nodes. Then, if π_{cur} is not of the first type (second type) the visit modifications do not increase $\chi^+_{\text{vis}}(v)$ ($\chi^-_{\text{vis}}(v)$, respectively), for nodes v, and hence due to Observation 6.52 $\chi^+_{\text{vis}}(v)$ ($\chi^-_{\text{vis}}(v)$, respectively) remains non-negative. So, the next lemma is proved.

Lemma 6.54 *Suppose we apply all visit modifications, leaf modifications, and internal modifications that we introduced before. Also suppose aside from these modifications, no change is applied to charging variables. Then, at any moment before the last round that the algorithm is not executing Block 20, for every internal node v that is not being visited and is not active, $\chi^-_{\text{vis}}(v) \leq 0$ and $\chi^+_{\text{vis}}(v) \leq 0$. Moreover, at any moment before the last round if π_{cur} is of the first type $\chi^+_{\text{vis}}(v) \leq 0$; otherwise, $\chi^-_{\text{vis}}(v) \leq 0$, for every internal node v.* □

To prove an upper bound on $\chi^-_{\text{vis}}(v)$ and $\chi^+_{\text{vis}}(v)$, for internal nodes v, in the cases in which an upper bound is not inferred from Lemma 6.54, we will need to show that if at some point during a round r, a node v is active, v has been active from the beginning of r and then we can conclude that during a visit to a child of v a visit modification has reduced a χ_{vis}-variable of v. Since the definition of a node being active in a negative round depends on whether $\mathsf{M}[v] < \mathsf{e}_{\text{goal}}$ or not, we prove that $\mathsf{M}[v]$ is not decreased by the algorithm and then we conclude that an inactive node does not become active unless the active step changes. Likewise, as the definition of a node being active in a positive round depends on whether $\mu(\mathsf{W}[v]) = \langle \mathsf{e}_{\text{goal}} \rangle$ or not and by Lemma 6.40 in positive rounds $\mathsf{e} = \langle \mathsf{e}_{\text{goal}} \rangle$, we prove that once $\mathsf{W}[v]$ is set equal to an object of value e, it remains an object of value e unless the active step changes.

Lemma 6.55 *Given a node v, if at some time bef after initialization the variable $\mathsf{M}[u]$ is well-valued, for every node u of $N(Q)[v]$, and at some time aft after time bef $\mathsf{M}[u]$ is well-valued again, for every node u of $N(Q)[v]$, then $\mathsf{M}^{(\text{bef})}[v] \leq \mathsf{M}^{(\text{aft})}[v]$.*

Proof. We first prove that after initialization $maxmin(v)$ is not decreased and then we use Lemma 5.14 to prove the lemma. Given a leaf l, by Lemma 6.20, after initialization the

result of the expression $\mu(\mathfrak{fi}[l])$ is never decreased. So, given a sub-union tree T, as by Definition 5.3 $\mathit{min\text{-}wait}(T) = \min_{l \in \mathit{leaves}(T)} \mu(\mathfrak{fi}[l])$, $\mathit{min\text{-}wait}(T)$ is never decreased. Consequently, since by Definition 5.4, $\mathit{maxmin}(v)$ is the maximum of $\mathit{min\text{-}wait}(T)$ over all sub-union trees T of $N(Q)[v]$, $\mathit{maxmin}(v)$ is not decreased. As a result $\mathit{maxmin}^{(bef)}(v) \leq \mathit{maxmin}^{(aft)}(v)$. Also, as by assumption at the times *bef* and *aft* $\mathsf{M}[u]$ is well-valued, for any node u of $N(Q)[v]$, by Lemma 5.14 $\mathsf{M}^{(bef)}[v] = \mathit{maxmin}^{(bef)}(v)$ and $\mathsf{M}^{(aft)}[v] = \mathit{maxmin}^{(aft)}(v)$. So, $\mathsf{M}^{(bef)}[v] \leq \mathsf{M}^{(aft)}[v]$. □

Lemma 6.56 *Consider a round r during which $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$ and a node v, such that at some time bef during r $\mathsf{W}[u]$ is well-valued, for every node u of $Q[v]$, and at a time aft, during r and after time bef, $\mathsf{W}[u]$ is well-valued again, for every node u of $Q[v]$. Also, suppose at neither of times aft or bef the procedure Gallop is being executed. Then, if $\mu(\mathsf{M}^{(aft)}[v]) \neq \mathsf{e}$, $\mu(\mathsf{M}^{(bef)}[v]) \neq \mathsf{e}$.*

Proof. We first prove that if at time *bef* there is an eyewitness T for v, at time *aft* T is still an eyewitness for v and then using Lemma 5.15 we conclude that if $\mu(\mathsf{M}^{(bef)}[v]) = \mathsf{e}$ then $\mu(\mathsf{M}^{(aft)}[v]) = \mathsf{e}$.

Considering an eyewitness T for v at time *bef*, we now prove that at time *aft* T is an eyewitness for v. The definition of T being an eyewitness depends only on elements of the array $\mathfrak{fi}$ and the variable e. Also, e is not modified during r. Moreover, the elements of $\mathfrak{fi}$ only may be modified during r by the Gallop procedure. Hence, since *aft* and *bef* are not during an execution of Gallop, it suffices to prove that every time that Gallop is called between the times *bef* and *aft*, if before that call T is an eyewitness, after that call T is still an eyewitness. We now prove this fact. Since the round r starts by executing one of lines 80 and 81, by Lemma 5.59 before the round r the three invariants are satisfied and all variables are well-valued. Hence, before r the preconditions of the visit procedures hold and so the round is started by a legal call. Thus, as the root is well-behaved, by Definition 5.15 every call to Gallop during r is legal. Therefore by Lemma 5.39 if before a call to Gallop T is an eyewitness after that call T is still an eyewitness.

Now we suppose $\mu(\mathsf{M}^{(bef)}[v]) = \mathsf{e}$ and we prove that $\mu(\mathsf{M}^{(aft)}[v]) = \mathsf{e}$. Since at both times *bef* and *aft* $\mathsf{W}[u]$ is well valued, for all nodes u of $Q[v]$ and during r $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$, by Lemma 5.15, in each of these two times, $\mu(\mathsf{W}[v]) = \mathsf{e}$ if and only if there

is an eyewitness for T. Thus as we supposed $\mu(\mathsf{M}^{(bef)}[v]) = \mathsf{e}$, there is an eyewitness T for v at time *bef*. So, as we proved above, T is an eyewitness for v at time *aft*. Consequently, $\mu(\mathsf{M}^{(aft)}[v]) = \mathsf{e}$. Hence, the lemma is correct. □

We will prove that χ_{vis}-variables of internal nodes are bounded by a factor of credits of nodes. To prove this fact, we first show that the credits of nodes are non-negative. Then using this result and Lemma 6.54 we can prove that our upper bound holds for inactive nodes.

Lemma 6.57 *After initialization, the credit of an internal node v never becomes negative.*

Proof. Since right after initialization the credits of nodes are zero and the only lines decreasing credits of nodes are lines 40 and 57, it suffices to prove that after that one of these two lines modifies the credit of a node v, the credit of v remains non-negative. Due to the condition checked on line 38 (on line 54), Block 10 (Block 13) is executed only if the credit of v is at least $L^-(u)$ (is $L^+(u)$), where u is the local variable of the procedure, storing a child of v. Block 10 first visits u and hence as $\mathsf{Credit}[v]$ is not a changing variable of Visit$^-(u)$ (of Visit$^+(u)$,), after this call still the credit of v is at least $L^-(u)$ (at least $L^+(u)$). Thus, after executing line 40 (line 57) the credit of v remains non-negative. So, the lemma is correct. □

Lemma 6.58 *At any time before the last round, given an internal node v that is not being visited, $\chi^-_{\text{vis}}(v) \leq 0$ if v is an intersection node and $\chi^-_{\text{vis}}(v) \leq \text{Credit}[v]W^-_L(v)$, otherwise.*

Proof. We first prove that it suffices to prove the lemma only for the times right after the executions of Visit$^-(v)$. After that, we prove that it suffices to discuss executions of Visit$^-(v)$ that occur in the negative rounds before the last round and then we show the correctness of the lemma for such executions of Visit$^-(v)$. In the proof we will use the fact that as Definition 2.28 shows, $W^-_L(v) \geq 0$, for every node v.

Now let us first prove that it suffices to prove the lemma for the times right after executions of Visit$^-(v)$. Since both $\chi^-_{\text{vis}}(v)$ and the credit of v are initialized to zero, before starting the first round the lemma is true. Also, the credit of v changes only when visiting v and χ_{vis}-variables are increased only during visits to v (Observation 6.52). Hence, it

suffices to prove that, if before visiting v the inequality claimed in the lemma is true, after finishing visiting v the inequality still holds.

We now prove that it suffices to prove the lemma for the visits to v that occur in the negative rounds before the last round. Since we are considering the times before the last round, by Lemma 6.35 the active step exists and is of the first type or of the second type. If the active step is of the second type, by Lemma 6.54 $\chi^-_{\text{vis}}(v) \leq 0$ and thus as by Lemma 6.57 $\mathsf{Credit}[v] \geq 0$, the lemma is true. So we suppose the active step is of the first type. Therefore, the round is either a null round or a negative round. If the current round is null, by definition it is a positive-visit. A positive-visit only changes credits of nodes in the "**case** *intersection node:* " block of Figure 5.4 and hence if v is a union node, a positive-visit does not change the credit of v. Also, by Definition 6.1 a visit modification may increase $\chi^-_{\text{vis}}(v)$ only in negative rounds and so in a null round $\chi^-_{\text{vis}}(v)$ is not increased. Consequently, if the current round is null, the inequality claimed in the lemma remains satisfied. Therefore, we will suppose the round is a negative round.

Before proving the lemma for times after negative-visits to v in negative rounds, we prove that we can suppose v is active before and after visiting v. If before visiting v, v is not active then right after starting the visit to v the visit modification increasing χ_{vis}-variables of v is not applied and so, due to Observation 6.52, χ_{vis}-variables of v are not increased. Also, in that situation by Lemma 6.54 just before visiting v χ_{vis}-variables of v are non-positive. Thus, after visiting v χ_{vis}-variables of v are non-positive and so the lemma is true. So we suppose before visiting v, v is active. Then, if after visiting v, v is not active, just before leaving $\text{Visit}^-(v)$, v is discharged and so $\chi^-_{\text{vis}}(v)$ will be at most 0. Thus in these situations also the lemma is true. Hence we suppose before and after visiting v, v is active.

Now we consider a negative-visiting of v in a negative round and we suppose just before visiting v the inequality claimed in the lemma holds for v and before and after visiting v, v is active. In such situations since before visiting v, by Lemma 6.13, the preconditions of $\text{Visit}^-(v)$ hold, the call is legal and so as all nodes are well-behaved, after visiting v the postconditions of the visit procedures hold. Thus, due to the preconditions (postconditions) of $\text{Visit}^-(v)$, before (after) visiting v all feature variables of all nodes in $Q[v]$ are well-valued. Also, as by assumption before and after visiting v, v is active, at both times $\mathsf{M}[v] < \mathsf{e}_{\text{goal}}$.

We now consider two cases based on v being an intersection node or a union node.

Case 1: v is an intersection node.
We first prove that there is a child w of v visited during the execution of Visit$^-(v)$ such that when visiting w, w is active and then we investigate the changes to $\chi^-_{\text{vis}}(v)$ when starting the visit to v and when starting the visit to w. Since T by definition is a sub-union tree and it contains the internal node v, T contains a child w of T. Also, as we proved when starting the visit to v all feature variables of nodes in $Q[v]$, including $\mathsf{M}[v]$, are well-valued, by Definition 5.6 $\mathsf{M}[v]$ is the maximum of $\mathsf{M}[v_0]$ over all children v_0 of v. Thus, as w is a child of v, when starting the visit to v, $\mathsf{M}[w] \leq \mathsf{M}[v]$ and hence as we proved when starting the visit to v, $\mathsf{M}[v] < \mathsf{e}_{\text{goal}}$, at that time $\mathsf{M}[w] < \mathsf{e}_{\text{goal}}$. Now the algorithm starts visiting children of v one by one (line 28). Since w is not a descendant of any child of v other than w, by visiting other children of v $\mathsf{M}[w]$ is not modified and so when visiting w still $\mathsf{M}[w] < \mathsf{e}_{\text{goal}}$. So, as w is selected as a node of the active tree, by definition just before visiting w, w is active.

Now we discuss the changes to $\chi^-_{\text{vis}}(v)$. Since by assumption when starting the visit to v, v is active, at the beginning of Visit$^-(v)$ the variable $\chi^-_{\text{vis}}(v)$ is increased by $L^-(v)W^-_L(v)$ and after that, during the visit to v, by Observation 6.52 no modification increases $\chi^-_{\text{vis}}(v)$. Also, when starting the visit to w (during the visit to v), as we proved w is active and so $\chi^-_{\text{vis}}(v)$ is decreased by $L^-(w)W^-_L(w)$. By Definition 2.28, $W^-_L(w) = W^-_L(v)\frac{L^-(v)}{L^-(w)}$. Hence, right after starting the visit to w, $\chi^-_{\text{vis}}(v)$ is decreased by $L^-(w)\left(W^-_L(v)\frac{L^-(v)}{L^-(w)}\right) = L^-(v)W^-_L(v)$. So, overall, $\chi^-_{\text{vis}}(v)$ is increased by at most $L^-(v)W^-_L(v) - L^-(w)W^-_L(w) = L^-(v)W^-_L(v) - L^-(v)W^-_L(v) = 0$. Thus, as by assumption before visiting v the inequality $\chi^-_{\text{vis}}(v) \leq 0$ held, after visiting v still this inequality holds.

Case 2: v is a union node.
First, for each direct execution of Block 11 by Visit$^-(v)$, we first prove that the child u of v selected before that execution of Block 11 is active. Since we proved before visiting v the preconditions of the procedure hold, before each execution of Block 11 by Lemmas 5.45 and 5.49 the three invariants hold and feature variables all nodes in $Q[v]$ are well-valued. Now consider a time *bef* right before starting a direct execution of Block 11 by Visit$^-(v)$.

Since we proved at both time *bef* and right after visiting v feature variables of nodes of $Q[v]$ are well-valued, $\mathsf{M}^{(bef)}[v]$ is at most the value of $\mathsf{M}[v]$ at the end of visiting v (Lemma 6.55) and so since at the end of visiting v, as argued, $\mathsf{M}[v] < \mathsf{e}_{\text{goal}}$, $\mathsf{M}^{(bef)}[v] < \mathsf{e}_{\text{goal}}$. Therefore, as at time *bef* $\mathsf{M}[v]$ is well-valued, by Definition 5.6 $\mathsf{M}^{(bef)}[v] = \mathsf{M}^{(bef)}[w]$ where w is a child of v in $N(Q)$ minimizing $\mathsf{M}^{(bef)}[w]$. Also before entering Block 11 at time *bef*, u is a selected by looking at the top of $\mathsf{H}[v]$ and since at time *bef* $\mathsf{H}[v]$ is well-valued, it follows from Definition 5.10 that u is a child of v in $N(Q)$ minimizing $\mathsf{M}^{(bef)}[u]$ over all children of v in $N(Q)$. Thus, $\mathsf{M}^{(bef)}[u] = \mathsf{M}^{(bef)}[w] = \mathsf{M}^{(bef)}[v] < \mathsf{e}_{\text{goal}}$. Also, since by definition the active tree is a sub-union tree of $N(Q)$ and v is in the active tree (because v is active), by definition every child of v in $N(Q)$ is in the active tree. So, u is in the active tree. Consequently, u satisfies both conditions of the definition of a node being active and hence u is active. Therefore, every time that Block 11 is executed, u is an active child.

Now we investigate the changes to $\mathsf{Credit}[v]$ and $\chi^-_{\text{vis}}(v)$. At the beginning of Visit$^-(v)$ the variable $\chi^-_{\text{vis}}(v)$ is increased by $L^-(v)W^-_L(v)$ and Credit$[v]$ is increased by $L^-(v)$ and hence $\chi^-_{\text{vis}}(v)$ and Credit$[v]W^-_L(v)$ are increased by the same amount. Now each time a child u of v is selected and then Block 11 is executed. We now prove that if before an execution of Block 11 the inequality $\chi^-_{\text{vis}}(v) \leq \text{Credit}[v]W^-_L(v)$ holds, after this execution still this inequality holds and in this way the lemma is proved. Visit$^-(u)$ does not change $\mathsf{Credit}[v]$ and after executing Visit$^-(u)$ in Block 11 $\mathsf{Credit}[v]$ is decreased by $L^-(u)$ (line 40). Thus, by an execution of Block 11 Credit$[v]W^-_L(v)$ is decreased by $L^-(u)W^-_L(v)$. We now prove that during Visit$^-(u)$ $\chi^-_{\text{vis}}(v)$ is decreased by at least $L^-(u)W^-(v)$ and so after these changes the inequality still holds. As we proved above, the child u selected by the algorithm is active. Therefore just after starting the visit to u, $\chi^-_{\text{vis}}(v)$ is decreased by $L^-(u)W^-_L(u)$. Hence, as by Observation 6.52 while visiting v no modification increasing $\chi^-_{\text{vis}}(v)$ is applied, by executing line 39 $\chi^-_{\text{vis}}(v)$ is decreased by at least $L^-(u)W^-_L(u)$. Moreover, since v is a union node, by Definition 2.28 $W^-_L(u) = W^-_L(v)$. So, $\chi^-_{\text{vis}}(v)$ is decreased by at least $L^-(u)W^-_L(v)$. So, after the execution of Block 11 the inequality $\chi^-_{\text{vis}}(v) \leq \text{Credit}[v]W^-_L(v)$ holds. □

Before proving a result similar to Lemma 6.58 for the variable $\chi^+_{\text{vis}}(v)$, for nodes v, we prove an upper bound on $\chi^+_{\text{vis}}(v)$ for the situation in which $\mathsf{E} \neq \mathsf{e}$.

Lemma 6.59 *Consider a time t before the last round such that at time t line 78 is not being executed, $cur^{(t)} \neq n+1$, $\pi_{cur^{(t)}}$ is of the second type, and $\mathsf{e} \neq E$. Then, at time t, $\chi^+_{\text{vis}}(v) \leq 0$.*

Proof. We first prove that every round started before t with the active step $\pi_{cur^{(t)}}$ is a null or a wasted round and then we conclude the correctness of the lemma.

We now assume there is round r, starting at a time *bef* before t, with the active step $\pi_{cur^{(t)}}$ that is not wasted nor null and we obtain a contradiction. Since the active step of the round r is of the second type, r is not a negative round. Hence r is a positive round and so at time *bef*, $\mathsf{E} = \mathsf{e}$. Also, as the active step of r is $\pi_{cur^{(t)}}$ and r starts at time *bef*, $cur^{(bef)} = cur^{(t)}$. So, applying Lemma 6.41 for $t_1 =$ *bef* and $t_2 = t$, we can conclude that $\mathsf{e}^{(t)} = \mathsf{E}^{(t)}$. This contradicts the assumption of the lemma. Hence, every round started before t with the active step $\pi_{cur^{(t)}}$ is a null or a wasted round.

We now prove that at time t, $\chi^+_{\text{vis}}(v) \leq 0$. Defining t_0 as the time at which *cur* has been set equal to $cur^{(t)}$, just before time t_0 all internal nodes are discharged and so at time t_0 $\chi^+_{\text{vis}}(v) \leq 0$. So, it suffices to prove that between the times t_0 and t, inclusive, $\chi^+_{\text{vis}}(v)$ is not increased. The time t_0 is not during a round as during a round *cur* is not modified. Also, we proved that every round started after t_0 and before t is wasted or null. Thus by definition no visit modification is applied between the times t_0 and t, inclusive. Hence, by Observation 6.52 between t_0 and t, inclusive, $\chi^+_{\text{vis}}(v)$ is not increased. Consequently at time t, $\chi^+_{\text{vis}}(v) \leq 0$. □

Lemma 6.60 *At any time before the last round, given an internal node v that is not being visited, $\chi^+_{\text{vis}}(v) \leq 0$ if v is a union node and $\chi^+_{\text{vis}}(v) \leq \text{Credit}[v]W^+_L(v)$, otherwise.*

Proof. As in the proof of the Lemma 6.58, we first show that it suffices to discuss the situations just after visiting v. Then we prove the claim in the cases in which the visit is not in a positive round and finally we consider the times just after visits to v in positive rounds. Since both $\chi^+_{\text{vis}}(v)$ and the credit of v are initialized to zero, before starting the first round the claim is true. Hence, as the credit of v only may change when visiting v and by Observation 6.52 $\chi^+_{\text{vis}}(v)$ is increased only during visits to v, it suffices to prove that, if

before visiting v the inequality claimed in the lemma is true, after finishing visiting v the inequality still holds.

We now consider a visit to v and the round r during which that visiting is occurred, we suppose before the visit to v the inequality of the lemma holds, and we discuss the cases in which r is negative, null, wasted, and positive, one by one. If the active step is of the first type, by Lemma 6.54 $\chi^{+}_{\text{vis}}(v) \leq 0$ and hence as by Lemma 6.57 $\mathsf{Credit}[v] \geq 0$ the lemma is true. So we suppose the active step is of the second type and hence r is not a negative round. If r is null, it is a negative-visiting and so if v is an intersection node, the credit of v does not change. Also, visit modifications are not applied in null rounds and thus by Observation 6.52 $\chi^{+}_{\text{vis}}(v)$ is not increased. Hence, if r is null the inequality claimed in the lemma remains satisfied. Also, if r is wasted, $\mathsf{E} \neq \mathsf{e}$ and so due to Lemma 6.59 when exiting the visit procedure $\chi^{+}_{\text{vis}}(v) \leq 0$. Hence, the lemma is correct for wasted rounds also.

Now we consider a visiting of v in a positive round. Then, a similar argument same as that used in the proof of Lemma 6.58 (the fourth paragraph of the proof) shows that if before or after visiting v, v is not active then the correctness of the lemma is trivial: it suffices to replace "χ^{-}_{vis}" and "Visit^{-}" with "χ^{+}_{vis}" and "Visit^{+}", respectively. Hence, we suppose that before and after visiting v, v is active. So, defining *aft* as the time just after visiting v, when starting the visit to v and at the times *aft*, by the definition of v being active, $\mu(\mathsf{W}[v]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$. Therefore, as by Lemma 6.40 $\mathsf{e} = \langle \mathsf{e}_{\text{goal}} \rangle$, when starting the visit to v and at time *aft*, $\mu(\mathsf{W}[v]) \neq \mathsf{e}$. Also, since before visiting v, by Lemma 6.13, the preconditions of Visit$^{+}(v)$ hold, the call is legal and so as all nodes are well-behaved after visiting v its postconditions hold. Thus, by the precondition (the postcondition) of Visit$^{+}(v)$ before (after) visiting v all feature variables of all nodes in $Q[v]$ are well-valued. We now consider two cases based on v being an intersection node or a union node.

Case 1: v is a union node.

We first prove that there is a child w of v visited by Visit$^{+}(v)$ such that when visiting w, w is active and then we investigate the changes to $\chi^{+}_{\text{vis}}(v)$. Since T by definition is a sub-intersection tree and it contains the internal node v, T contains a child w of T. Also, as we proved, when starting the visit to v all feature variables of nodes in $Q[v]$, including $\mathsf{W}[v]$, are well-valued. Hence, considering the facts that by the definition of

positive rounds $\mathsf{e} = \mathsf{E}$, by Lemma 6.40 $\mathsf{e} \notin \{-\infty, \infty\}$, and we proved when starting the visit to v $\mu(\mathsf{W}[v]) \neq \mathsf{e}$, by Definition 5.8 we can conclude that $\mu(\mathsf{W}[w]) \neq \mathsf{e} = \langle \mathsf{e}_{\text{goal}} \rangle$. Now the algorithm starts visiting children of v one by one (line 47). Since w is not a descendant of any child of v other than w, by visiting other children of v $\mathsf{W}[w]$ is not modified and hence when starting the visit to w still $\mu(\mathsf{W}[w]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$. So, as w is selected as a node of the active tree, by definition just before visiting w, w is active. Now the rest of the proof is the same as the second paragraph of Case 1 in the proof of Lemma 6.58: it suffices to replace occurrences of "Visit$^-$", "L^-", "W_L^-", "χ_{vis}^-", and "union" with "Visit$^+$", "L^+", "W_L^+", "χ_{vis}^+", and "intersection", respectively.

Case 2: v is an intersection node.
First, for each direct execution of Block 15 by Visit$^+(v)$, we prove that the child u of v selected before that execution of Block 15 is active and then we prove that before and after the execution of the block the inequality of the lemma holds.

We now first prove that before a direct execution of Block 15 by Visit$^+(v)$ there is a child w of v such that $\mu(\mathsf{W}[w]) \neq \mathsf{e}$ and then we conclude that just before the child u visited in the block is visited, u is active. Since we proved before visiting v the preconditions of the procedure hold, before each execution of Block 15 by Lemmas 5.45 and 5.52 the three invariants hold and feature variables all nodes in $Q[v]$ are well-valued. Now consider a time *bef* when starting a direct execution of Block 11 by Visit$^+(v)$. We have proved feature variables of nodes in $Q[v]$ at both times *bef* and *aft* are well-valued where *aft* was defined as the time just after visiting v. Hence, considering the facts that $\mu(\mathsf{W}^{(aft)}[v]) \neq \mathsf{e}$, by the definition of positive rounds $\mathsf{e} = \mathsf{E}$, and by Lemma 6.40 $\mathsf{e} \notin \{-\infty, \infty\}$, by Lemma 6.56 we can conclude that $\mu(\mathsf{W}^{(bef)}[v]) \neq \mathsf{e}$. So, as at time *bef* $\mathsf{W}[v]$ is well-valued, by Definition 5.6 there is a child w of v such that $\mu(\mathsf{W}^{(bef)}[w]) \neq \mathsf{e}$.

Now we prove that at the time *bef* the child u of v is active where u is the local variable of Visit$^+(v)$. Before entering Block 15 at time *bef* u is a selected by looking at the top of $\mathsf{H}[v]$. We first prove that $\mu(\mathsf{W}^{(bef)}[u]) \neq \mathsf{e}_{\text{goal}}$ and then we show that w is in the active tree. Since at time *bef* $\mathsf{H}[v]$ is well-valued, by Definition 5.11 the key of every child of v in $\mathsf{H}[v]$ is up-to-date and so since, as we proved, $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$, the key of every child v in w is at most e. So, if the key of u in $\mathsf{H}[v]$ is e, as u is the top element of the heap and so has

the minimum key, the key of every child of v, including w, in $\mathsf{H}[v]$ is also e. Therefore, as we proved the key of w in $\mathsf{H}[v]$ is up-to-date and the statements $\mathsf{e} = \mathsf{E}$ and $\mathsf{e} \notin \{-\infty, \infty\}$ are true, by Definition 5.11 $\mu(\mathsf{W}^{(bef)}[w]) = \mathsf{e}$ while we proved $\mu(\mathsf{W}^{(bef)}[w]) \neq \mathsf{e}$. Hence, the key of u in $\mathsf{H}[v]$ is less than e and thus as this key is up-to-date by Definition 5.11 $\mu(\mathsf{W}^{(bef)}[u]) \neq \mathsf{e} = \langle \mathsf{e}_{\text{goal}} \rangle$. Also, since by definition the active tree is a sub-intersection tree of Q and v is in the active tree (because v is active), by definition every child of v in Q is in the active tree. So, u is in the active tree. Hence, u satisfies both conditions of the definition of a node being active and so u is active. Therefore, every time that Block 15 is executed, u is an active child.

Now the rest of the proof is the same as the second paragraph of Case 2 in the proof of Lemma 6.58: it suffices to exchange occurrences of "Visit$^-$", "L^-", "W_L^-", "χ_{vis}^-", "Block 11", "lines 39 and 40", and "union" with "Visit$^+$", "L^+", "W_L^+", "χ_{vis}^+", "Block 15", "lines 56 and 57", and "intersection", respectively. □

We now use Lemmas 6.58 and 6.60 to give new upper bounds on χ_{vis}-variables. Given an internal node v, at any time before the last round at which v is not being visited, by Lemma 6.58, depending on v being a union node or an intersection node, $\chi_{\text{vis}}^-(v) \leq 0$ or $\chi_{\text{vis}}^-(v) \leq \mathsf{Credit}[v] W_L^-(v)$. Also, whenever a node starts being visited, possibly $\chi_{\text{vis}}^-(v)$ is increased by $L^-(v) W_L^-(v)$ and after that by Observation 6.52 $\chi_{\text{vis}}^-(v)$ is not increased. Thus, at any time before the last round $\chi_{\text{vis}}^-(v) \leq \mathsf{Credit}[v] W_L^-(v) + L^-(v) W_L^-(v)$. Hence, since $\mathsf{Credit}[v] \leq L_{\max}^-(v)$ (Lemma 6.1), at any time before the last round $\chi_{\text{vis}}^-(v) \leq W_L^-(v)(L^-(v) + L_{\max}^-(v))$. Also, as during the last round no visit modification applies, in the last round χ_{vis}-variables are not modified and so in the last round also $\chi_{\text{vis}}^-(v) \leq W_L^-(v)(L^-(v) + L_{\max}^-(v))$. So, at any time after initialization the inequality $\chi_{\text{vis}}^-(v) \leq W_L^-(v)(L^-(v) + L_{\max}^-(v))$ holds. Similarly, by replacing all occurrences of "χ_{vis}^-", "W_L^-", "L^-", "$L_{\max}^-$", and "Lemma 6.58" with "χ_{vis}^+", "W_L^+", "L^+", "$L_{\max}^+$", and "Lemma 6.60", it is proved that $\chi_{\text{vis}}^+(v) \leq 2L^+(v) W_L^+(v)(L^+(v) + L_{\max}^+(v))$. So, the next lemma is correct.

Lemma 6.61 *At any time after initialization,* $\chi_{\text{vis}}^-(v) \leq W_L^-(v)(L^-(v) + L_{\max}^-(v))$ *and* $\chi_{\text{vis}}^+(v) \leq W_L^+(v)(L^+(v) + L_{\max}^+(v))$*, for every internal node* v*.* □

Now we evaluate the sum of all charging variables at the end of the execution of the algorithm. We evaluate the sum of χ_{obj}-variables, χ_{vis}-variables, and χ_e-variables, in order.

Lemma 6.62 *Given a $\mathcal{P}$-visited or $\mathcal{P}$-touched object o of a leaf l, at the end of the execution of the algorithm $\chi_{\text{obj}}(o) \leq cost(o)$ where $cost(o) = W_L(l)(1 + \log len(o))$ and $len(o)$ is the length of the $\mathcal{P}$-gap containing o.*

Proof. We given an upper bound on the number of times that $\chi_{\text{obj}}(o)$ is increased and the amount by which $\chi_{\text{obj}}(o)$ is increased each time. Every time that the value of $\chi_{\text{obj}}(o)$ is increased, l is being visited and o is the first chargeable object of l. Also, each time $\chi_{\text{obj}}(o)$ is increased by one of values $W_L^-(l)$ or $W_L^+(l)$. Furthermore, every time that a leaf l is being visited, an l-gallop is executed. Moreover, by Lemma 6.46 the number of times that an l-gallop is executed and o is the smallest chargeable object of l is at most $1 + \log len(o)$. Hence, $\chi_{\text{obj}}(o)$ is increased at most $1 + \log len(o)$ times and each time it is increased by at most $\max(W_L^-(l), W_L^+(l)) \leq W_L(l)$. Consequently, at the end of the execution of the algorithm, $\chi_{\text{obj}}(o) \leq W_L(l)(1 + \log len(o)) = cost(o)$. □

Lemma 6.63 *The sum of $cost(o)$ over all $\mathcal{P}$-visited and $\mathcal{P}$-touched objects o is at most $2D_L(\mathcal{P})$.*

Proof. We first prove that this sum is at most twice the sum of $cost(o)$ over all $\mathcal{P}$-visited objects o and then we evaluate $\sum_{o \in Visited(\mathcal{P})} cost(o)$. By definition, a $\mathcal{P}$-touched object is an object o of a leaf l such that the object p after o is $\mathcal{P}$-visited. So, if o is not $\mathcal{P}$-visited, due to Definition 2.26, o belongs to the same gap that contains p. Hence, given a $\mathcal{P}$-touched object o of a leaf l that is not $\mathcal{P}$-visited, $len(o) = len(p)$ and hence $cost(o) = W_L(l)(1 + \log len(o)) = W_L(l)(1 + \log len(p)) = cost(p)$ where p is a $\mathcal{P}$-visited object and is just after o in $\omega(l)$. Consequently, the sum of $cost(o)$ over all objects o that are $\mathcal{P}$-touched but are not $\mathcal{P}$-visited is at most the sum of $cost(p)$ over all $\mathcal{P}$-visited objects p. Hence, the sum of $cost(o)$ over all $\mathcal{P}$-visited and $\mathcal{P}$-touched objects is at most twice the sum of $cost(p)$ over all $\mathcal{P}$-visited objects p.

Now, we prove that the sum of $cost(o)$ over all $\mathcal{P}$-visited objects o is at most $D_L(\mathcal{P})$, thereby proving the correctness of the lemma. Given a leaf l, defining $o_{l,1}, \ldots, o_{l,m_l}$ as the sequence of $\mathcal{P}$-visited objects of l, by Definition 2.26 l has m_l $\mathcal{P}$-gaps and the ith $\mathcal{P}$-gap of l is the $\mathcal{P}$-gap containing $o_{l,i}$, for every i, $1 \leq i \leq m_l$. Hence, the length of ith $\mathcal{P}$-gap

of l is $len(o_{l,i})$, for every leaf l and integer i, $1 \leq i \leq m_l$. Therefore, $D_L(\mathcal{P})$, as defined in Equation 2.3, equals

$$\begin{aligned} D_L(\mathcal{P}) = & \sum_{l \in leaves(Q)} W_L(l) \left(\sum_{1 \leq i \leq m_l} 1 + \log len(o_{l,i}) \right) \\ = & \sum_{l \in leaves(Q)} \sum_{1 \leq i \leq m_l} W_L(l)(1 + \log len(o_{l,i})) \\ = & \sum_{l \in leaves(Q)} \sum_{1 \leq i \leq m_l} cost(o_{l,i}) \\ = & \sum_{p \in Visited(\mathcal{P})} cost(p). \end{aligned}$$

□

The next lemma follows immediately from Lemmas 6.62 and 6.63.

Lemma 6.64 *The sum of χ_{obj}-variables at the end of the execution of the algorithm is at most $2D_L(\mathcal{P})$.* □

We next evaluate the sum of χ_e-variables at the end of the execution of the algorithm. We first give an upper bound on the number of times that a node is usefully discharged.

Lemma 6.65 *Before starting the last round, during the time period in which $cur = i$, for an i, $1 \leq i \leq n+1$, each internal node is discharged usefully at most twice.*

Proof. We assume to the contrary that a node v is discharged usefully at least three times. We first prove that $i \neq n+1$ and π_i is of the first or of the second type. Then, we consider the two first times that v is discharged usefully and we discuss the properties of these two times. Then, we consider two cases based on π_i being of the first type or of the second type and in each case we show that a contradiction exists.

Let us first prove that $i \neq n+1$ and π_i is not of the third type. Since before each change to *cur* we discharge all nodes, at the beginning of the period in which $cur = i$ χ_{vis}-variables of all nodes are non-positive. Also, visit modifications are applied only during rounds before the last round and in such rounds by Lemma 6.35 $cur \neq n+1$ and π_{cur} is of the first or of the second type. Hence, if $i = n+1$ or π_i is of the third type, during the time period in which $cur = i$ no visit modification is applied and thus by Observation 6.52 χ_{vis}-variables are not increased are remain non-positive. So, $i \neq n+1$ and π_i is not of the third type as otherwise there is no useful discharging, contradicting the assumption that in this time period three discharges are applied.

Now let us discuss the first two times that v is discharged usefully. Defining $\mathcal{T}$ as the time period in which $cur = i$, if after setting *cur* equal to i and before the last round *cur* changes, the last time that v is discharged in $\mathcal{T}$ is right before *cur* is set equal to a new integer other than i. Thus, as we suppose v is discharged usefully at least three times in $\mathcal{T}$ before the last round, the first two useful times that v is discharged in $\mathcal{T}$ are before that last discharging and also are before the last round. Also, by the definition of internal modifications, except the discharging of v that happens right before changing *cur* or the discharging that happens right after starting the last round, every other discharging of v in $\mathcal{T}$ is right before finishing an execution $\mathcal{E}$ of Visit$^-(v)$, Visit$^+(v)$, or Block 20 such that just before $\mathcal{E}$ v is active and right after $\mathcal{E}$ v is not active. Hence, the first (the second) discharging of v in $\mathcal{T}$ is right before finishing an execution $\mathcal{E}_1$ (an execution $\mathcal{E}_2$) of Visit$^-(v)$, Visit$^+(v)$, or Block 20. Now, we define bef_1 and aft_1 (bef_2 and aft_2) as the times just before and right after that execution $\mathcal{E}_1$ (the execution $\mathcal{E}_2$) of Block 20, Visit$^-(v)$, or Visit$^+(v)$. Therefore, at time bef_1 (at time bef_2) v is active and at time aft_1 (at time aft_2) v is inactive. Also, since e is only modified by Update$\mathsf{C}$ and hence *cur* is only modified during an execution of this procedure, during executions $\mathcal{E}_1$ and $\mathcal{E}_2$ of Visit$^-(v)$, Visit$^+(v)$, or Block 20 *cur* is not modified. Hence, as we considered the first two times that v is discharged usefully in the time period in which $cur = i$ and we proved these two times are during $\mathcal{E}_1$ and $\mathcal{E}_2$, $cur^{(aft_1)} = cur^{(bef_1)} = i$ and $cur^{(aft_2)} = cur^{(bef_2)} = i$.

We now first prove that $aft_1 < bef_2$ and then we show that at both times aft_1 and bef_2 feature variables of all nodes in $Q[v]$ are well-valued. Finally we use Lemmas 6.55 and 6.56 to obtain a contradiction. During a visit to v only children of v are visited

and v does not visit its self or its parents. Also, during a visit to v Block 20 is not executed and during an execution of Block 20 only UpdateE is executed. Hence, each of the times interval (bef_1, aft_1) and (bef_2, aft_2) are disjoint as otherwise during an execution of Visit$^-(v)$, Visit$^+(v)$, or Block 20, another execution of Visit$^-(v)$, Visit$^+(v)$, or Block 20 happens. Furthermore, aft_1 and aft_2 are times just after the first useful discharging and the second useful discharging of v happen, respectively, and thus $aft_1 < aft_2$. So, $bef_1 < aft_1 < bef_2 < aft_2$. Consequently, $aft_1 < bef_2$.

Next we prove that at both times aft_1 and bef_2 feature variables of all nodes in $Q[v]$ are well-valued. Each of aft_1 and bef_2 are just before starting or right after finishing an execution of Block 20, Visit$^-(v)$, or Visit$^+(v)$. Since by Lemma 6.13 each visit to v is legal and v is well-behaved, before visiting v the preconditions of the visit procedures holds and thus when finishing visiting v the postconditions of the visit procedures holds. Hence before and after each visit to v all feature variables of all nodes in $Q[v]$ are well-valued. Also, by Lemma 5.59 before and after each execution of Block 20 all variables are well-valued. So, at both times aft_1 and bef_2 all feature variables of all nodes in $Q[v]$ are well-valued.

Now we consider two cases depending on π_{cur} at times aft_1 and bef_2 is of the first type or of the second type and in each case we obtain a contradiction. Since we proved *cur* has the same value i at the times aft_1 and bef_2, $\mathsf{e}_{\text{goal}}^{(aft_1)} = \mathsf{e}_{\text{goal}}^{(bef_2)}$. Now, first suppose π_{cur} is of the first type and so $\pi_{cur} = \pi_T^-$, for a sub-union tree T of $N(Q)$. Then, since at the times bef_1 and bef_2 v is active, v is in T and thus v is in $N(Q)$. Therefore, as we proved at the times aft_1 and bef_2 feature variables of nodes in $Q[v]$ are well-valued, it follows from Lemma 6.55 that $\mathsf{M}^{(aft_1)}[v] \leq \mathsf{M}^{(bef_2)}[v]$. On the other hand, since time aft_1 v is inactive and bef_2 v is active, by definition the inequalities $\mathsf{M}^{(aft_1)}[v] \geq \mathsf{e}_{\text{goal}}^{(aft_1)}$ and $\mathsf{M}^{(bef_2)}[v] < \mathsf{e}_{\text{goal}}^{(bef_2)} = \mathsf{e}_{\text{goal}}^{(aft_1)}$ hold and thus $\mathsf{M}^{(bef_2)}[v] < \mathsf{M}^{(aft_1)}[v]$. Hence, there is a contradiction in this case.

Next we consider the case in which π_{cur} is of the second type. We first prove that at both times aft_1 and bef_2 the equality $\mathsf{E} = \mathsf{e}$ holds. Since just before time aft_1 we had a useful discharging, before the discharging of this time one of $\chi_{\text{vis}}^-(v)$ or $\chi_{\text{vis}}^+(v)$ was positive. As by assumption the current step is of the second type, by Lemma 6.54 $\chi_{\text{vis}}^-(v) \leq 0$. Hence, before the discharging of time aft_1 $\chi_{\text{vis}}^+(v) > 0$. Therefore, as the discharging that happened just before time aft_1 is not during an execution of line 78 (it can be just before finishing an execution of Block 20), by Lemma 6.59 at time aft_1 $\mathsf{e} = \mathsf{E}$. So, as $aft_1 < bef_2$

and bef_2 is not during an execution of line 78 (it can be just before an execution of line 78), by Lemma 6.41 for $t_1 = aft_1$ and $t_2 = bef_2$ at time bef_2 also $\mathsf{e} = \mathsf{E}$. Moreover since π_{cur} at these two times is of the second type, by Lemma 6.40 at these two times $\mathsf{e} \notin \{-\infty, \infty\}$ and $\mathsf{e} = \langle \mathsf{e}_{\text{goal}} \rangle$. Furthermore, as we proved, at each of these two times all feature variables of nodes in $Q[v]$ are well-valued. Consequently, as $bef_2 > aft_1$, by Lemma 6.56 for $bef = aft_1$ and $aft = bef_2$, if $\mu(\mathsf{W}^{(bef_2)}[v]) \neq \mathsf{e}$ then $\mu(\mathsf{W}^{(aft_1)}[v]) \neq \mathsf{e}$. Hence, as $\mathsf{e} = \langle \mathsf{e}_{\text{goal}} \rangle$, if $\mu(\mathsf{W}^{(bef_2)}[v]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$ then $\mu(\mathsf{W}^{(aft_1)}[v]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$. Now we obtain a contradiction by showing that $\mu(\mathsf{W}^{(bef_2)}[v]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$ but $\mu(\mathsf{W}^{(aft_1)}[v]) = \langle \mathsf{e}_{\text{goal}} \rangle$. Since we proved at time bef_2 v is active, $\mu(\mathsf{W}^{(bef_2)}[v]) \neq \langle \mathsf{e}_{\text{goal}} \rangle$ and at time bef_2 v is in the active tree. Also, since *cur* has the same values at the times aft_1 and bef_2, the active trees of these two times are the same and so at time aft_1 v is in the active tree, as well. Therefore, since we proved at time aft_1 v is not active, $\mu(\mathsf{W}^{(aft_1)}[v]) = \langle \mathsf{e}_{\text{goal}} \rangle$. So, in both cases we obtained a contradiction. Hence, the assumption that v is discharged usefully at least three times during $\mathcal{T}$ before the last round is false. □

The upper bound that we prove for the sum of all χ_e-variables at the end of the execution of the algorithm is as follows. Given a subtree T of Q, we define $K^-(T)$ and $K^+(T)$ as follows.

$$K^-(T) = \sum_{v \in nodes(T)} W_L^-(v)(L^-(v) + L_{\max}^-(v))$$

$$K^+(T) = \sum_{v \in nodes(T)} W_L^+(v)(L^+(v) + L_{\max}^+(v))$$

Now the maximum of integers $K^-(T)$, for all sub-union trees T of $N(Q)$, and $K^+(T)$, for all sub-intersections trees T of Q, is defined as $K_{\max}$.

Lemma 6.66 *The sum of all χ_e-variables at the end of the execution of the algorithm is at most $6K_{\max}(|Visited(\mathcal{P})| + 1)$.*

Proof. We consider every value i assigned to *cur* and we evaluate the amount m_i by which the sum of χ_e-variables is increased in the period in which $cur = i$. Then we find the sum of this amount over all possibilities for i.

Given an integer i assigned to *cur*, we prove that $m_i \leq K_{\max}$. First we consider the case in which $i \neq n + 1$ and $\pi_i = \pi_T^-$ ($\pi_i = \pi_T^+$), for some T. Then, during the time that

$cur = i$, by Lemma 6.65 before the last round each node is discharged usefully at most twice. Also, in the last round, by the definition of internal modifications, every node is discharged exactly once. Hence, during the time that $cur = i$, every node is discharged usefully at most three times. We now prove that each time that a node v is discharged $\chi_e(v)$ is increased only if v is in T and it is increased by at most $W_L^-(v)(L^-(v) + L^-_{\max}(v))$ (at most $W_L^+(v)(L^+(v) + L^+_{\max}(v))$). Since the active step is of the first (second) type, by Lemma 6.54 $\chi^+_{\text{vis}}(v) \leq 0$ ($\chi^-_{\text{vis}}(v) \leq 0$) and so $\chi_e(v)$ is increased by at most $\chi^-_{\text{vis}}(v)$ (at most $\chi^+_{\text{vis}}(v)$). Also, before *cur* is set equal to i all internal nodes are discharged and after that a visit modification is applied to the χ_{vis}-variables of v only if v has been active. Moreover, if v is not in T, during the time period in which $cur = i$, by the definition of v being active, v has not become active. Therefore, if v is not in T $\chi^-_{\text{vis}}(v)$ ($\chi^+_{\text{vis}}(v)$) is not increased and so it remains non-positive. Furthermore, by Lemma 6.61 at any time after initialization $\chi^-_{\text{vis}}(v) \leq W_L^-(v)(L^-(v) + L^-_{\max}(v))$ ($\chi^+_{\text{vis}}(v) \leq W_L^+(v)(L^+(v) + L^+_{\max}(v))$). Hence, each time that v is discharged, $\chi_e(v)$ is not increased if v is not in T; otherwise $\chi_e(v)$ is increased by at most $W_L^-(v)(L^-(v)+L^-_{\max}(v))$ (at most $W_L^+(v)(L^+(v)+L^+_{\max}(v))$). So, in the whole time that $cur = i$, as each node is discharged usefully at most three times, the sum of χ_e-variables is increased by at most $3\sum_{u \in nodes(T)} W_L^-(u)(L^-(u) + L^-_{\max}(u)) = 3K^-(T) \leq 3K_{\max}$ (by at most $3\sum_{u \in nodes(T)} W_L^+(u)(L^+(u) + L^+_{\max}(u)) = 3K^+(T) \leq 3K_{\max}$, respectively). So, $m_i \leq 3K_{\max}$.

Next we consider the case in which $i = n + 1$ or $\pi_i = \pi^\infty$. Just before *cur* is set to i all internal nodes are discharged and hence all χ_{vis}-variables have become non-positive. Also, every round during which $cur = i$ by Lemma 6.35 is the last round and thus no visit modification is applied in a round in which $cur = i$. So, in the whole time that $cur = i$ all χ_{vis}-variables remain non-positive and thus internal modifications do not increase χ_e-variables. Therefore, $m_i = 0$.

Now we can evaluate the the sum of all χ_e-values. By Lemma 6.37 *cur* is assigned at most $2|Visited(\mathcal{P})| + 2$ different values and we proved for every i assigned to *cur*, in the time that $cur = i$, $\chi_e(v)$ is increased by at most $3K_{\max}$. Thus, at the end of the algorithm, the sum of χ_e-variables is at most $6K_{\max}(|Visited(\mathcal{P})| + 1)$. So, the lemma is true. □

Now we can evaluate the total leaf-cost of all rounds as follows.

Lemma 6.67 *The sum of leaf-costs of all rounds is*

$$O(D_L(\mathcal{P}) + (K_{\max} + k)(|\mathit{Visited}(\mathcal{P})| + 1)).$$

Proof. We first evaluate the sum of all charging variables at the end of the execution of the algorithm, then we give an upper bound on the total leaf-cost of all positive and negative rounds, and finally we evaluate the total leaf-cost of all rounds. By Lemmas 6.64 and 6.66, the total value of all χ_{obj}-variables and all χ_e-variables at the end of the execution of the algorithm are at most $2D_L(\mathcal{P})$ and $6K_{\max}(|\mathit{Visited}(\mathcal{P})| + 1)$, respectively. Also, since at the beginning of the last round all internal nodes are discharged and during the last round no visit modification is applied, after the last round all χ_{vis}-variables are at most zero. Hence, the sum of all charging variables at the end of the execution of the algorithm is at most $2D_L(\mathcal{P}) + 6K_{\max}(|\mathit{Visited}(\mathcal{P})| + 1) = O(D_L(\mathcal{P}) + K_{\max}(|\mathit{Visited}(\mathcal{P})| + 1))$. Thus, it follows from Lemma 6.45 that the total leaf-cost of all negative and positive nodes except the last round is $O(D_L(\mathcal{P}) + K_{\max}(|\mathit{Visited}(\mathcal{P})| + 1))$. Therefore, by Lemma 6.39 the total leaf-cost of all null rounds except the last round is at most $O(D_L(\mathcal{P}) + K_{\max}(|\mathit{Visited}(\mathcal{P})| + 1) + 4k(|\mathit{Visited}(\mathcal{P})| + 1)) = O(D_L(\mathcal{P}) + (K_{\max} + k)(|\mathit{Visited}(\mathcal{P})| + 1))$. Additionally by Lemma 6.42 the total leaf-cost of all wasted rounds except the last round is at most $4k(|\mathit{Visited}(\mathcal{P})|+1)$. Hence, the total leaf-cost of all rounds except the last round is $O(D_L(\mathcal{P}) + (K_{\max} + k)(|\mathit{Visited}(\mathcal{P})| + 1) + 4k(|\mathit{Visited}(\mathcal{P})| + 1)) = O(D_L(\mathcal{P}) + (K_{\max} + k)(|\mathit{Visited}(\mathcal{P})| + 1))$. Furthermore, by Lemmas 6.3 and 6.4 the leaf-cost the last round is at most $\max(L^-(\text{root}), L^+(\text{root}))$ which is by Lemmas 6.23 and 6.24 at most $\max(|\mathit{leaves}(Q[\text{root}])|, |\mathit{leaves}(Q[\text{root}])|) = |\mathit{leaves}(Q)| = k$. Hence, the total leaf cost of all rounds is $O(D_L(\mathcal{P}) + (K_{\max} + k)(|\mathit{Visited}(\mathcal{P})| + 1))$. □

Now we evaluate the number of times that the procedure Gallop is called by line 75 and then we evaluate the value of g.

Lemma 6.68 *Line 75 is executed at most $2D_L(\mathcal{P})$ times.*

Proof. We first prove that the number of times that line 75 is executed is at most $\frac{1}{W_L(\mathit{sl}(Q))} 2D_L(\mathcal{P})$ and then we show that $\frac{1}{W_L(\mathit{sl}(Q))} \leq 1$. By Lemma 6.50 every time that Gallop is called by line 75, the speedy leaf has a chargeable object. Also, by definition a chargeable object is a $\mathcal{P}$-visited or a $\mathcal{P}$-touched object. Furthermore, for every

$\mathcal{P}$-visited or $\mathcal{P}$-touched object o of the speedy leaf, by Lemma 6.46 the number of times that Gallop($sl(Q)$) is called and o is the first chargeable object of the speedy leaf is at most $1 + \log len(o) = \frac{cost(o)}{W_L(sl(Q))}$ (the function *cost* is defined in Lemma 6.62). Hence, the number of times that line 75 calls Gallop($sl(Q)$) is at most the sum of $\frac{cost(o)}{W_L(sl(Q))}$ over all $\mathcal{P}$-visited and $\mathcal{P}$-touched objects of the speedy leaf. This sum by Lemma 6.63 is at most $\frac{1}{W_L(sl(Q))} 2D_L(\mathcal{P})$.

Now we complete the proof by showing that $\frac{1}{W_L(sl(Q))} \leq 1$. Since the speedy leaf exists, the root is a union node. Hence, as by Definition 2.28 $W_L^-(\text{root}) = 1$ and the speedy leaf is a child of the root, by Definition 2.28 $W_L^-(sl(Q)) = W_L^-(\text{root}) = 1$. Therefore, as by definition $W_L(sl(Q)) = W_L^-(sl(Q)) + W_L^+(sl(Q))$, $W_L(sl(Q)) \geq 1$ and so $\frac{1}{W_L(sl(Q))} \leq 1$. Thus, the lemma is correct. □

Lemma 6.69 *The value of g is at most $O(D_L(\mathcal{P}) + (K_{\max} + k)(|Visited(\mathcal{P})| + 1))$.*

Proof. We first evaluate the number of times that the visit procedures have called Gallop, then we evaluate the number of times that line 83 has called Gallop, and finally we summarize all results of this section and obtain an upper bound on g.

Let us first evaluate the number of times that the procedure Gallop is called during executions of the rounds. It follows from the definition of the leaf-cost (on page 212) that the total leaf-cost of all rounds is the amount by which the sum of credits of all nodes is increased by rounds of the algorithm plus the number of times that the procedure Gallop is called by rounds. Now considering the facts that the procedure Initialize sets the credit of every node equal to zero, at the end of the execution of the algorithm by Lemma 6.57 the credit of every node is non-negative, and after initialization the credit of nodes is modified only by visit procedures, we can conclude that the amount by which the sum of credits of all nodes is increased by rounds of the algorithm is not negative. Hence, the total leaf-cost of all nodes is at least as large as the number of times of the procedure Gallop is called by rounds. Thus, it follows from Lemma 6.67 that the number of times that Gallop is called during executions of lines 80 and 81 is $O(D_L(\mathcal{P}) + (K_{\max} + k)(|Visited(\mathcal{P})| + 1))$.

Next we prove that line 83 is executed at most once. When reaching line 82 by the condition checked on line 72, $\mathsf{e} = \infty$. Consequently, if line 83 is executed, on its first execution $\mathsf{e} = \infty$ and also by the condition checked on line 82 $\mathrm{fi}[sl(Q)] \neq \text{END}$. As a result,

by Definition 5.1 fi[$sl(Q)$] is an object of the speedy leaf. So, the index of fi[$sl(Q)$] in $\omega(sl(Q))$ is at most the length of $\omega(sl(Q))$. Therefore, when the algorithm is executing the Gallop called by the first execution of line 83, as line 11 sets i equal to the index of fi[$sl(Q)$] in $\omega(sl(Q))$, when executing line 12 $i \leq \text{length}(\omega(sl(Q)))$ and $\mu(\omega(sl(Q))[\text{length}(\omega(sl(Q)))]) < \infty = \mathtt{e}$. Thus, the condition checked on line 12 is true and so by Lemma 6.19, the $sl(Q)$-gallop changes the value of fi[$sl(Q)$] and sets it equal to END. So, as there is only one line in Gallop changing fi[$sl(Q)$] (line 20), fi[$sl(Q)$] is not modified more than once during the $sl(Q)$-gallop and hence when exiting Gallop fi[$sl(Q)$] = END and so the condition checked on line 82 is true. So, line 83 is not executed anymore.

Now we can evaluate g. Every call to g is during an execution of one of lines 80, 81, 75, or 83. We proved above that during executions of lines 80 and 81, Gallop is called at most $O(D_L(\mathcal{P}) + (K_{\max} + k)(|Visited(\mathcal{P})| + 1))$ times and line 83 is executed at most once. Also, by Lemma 6.68 Line 75 is executed at most $2D_L(\mathcal{P})$ times. So, the number of times that Gallop is called is $O(D_L(\mathcal{P}) + (K_{\max} + k)(|Visited(\mathcal{P})| + 1|) + 2D_L(\mathcal{P}) + 1) = O(D_L(\mathcal{P}) + (K_{\max} + k)(|Visited(\mathcal{P})| + 1))$. □

Finally since by Lemma 6.69 $g = O(D_L(\mathcal{P})+(K_{\max}+k)(|Visited(\mathcal{P})|+1))$ and according to Lemma 6.29 the algorithm is executed takes $O(g\mathcal{H} + hk^2)$ time, the next theorem is obtained.

Theorem 6.70 *The algorithm takes*

$$O\left(D_L(\mathcal{P})\mathcal{H} + ((K_{\max} + k)(|Visited(\mathcal{P})| + 1))\mathcal{H} + hk^2\right)$$

time. □

In this chapter we compared the time taken by the algorithm with the L-gap cost of a given proof $\mathcal{P}$ of the instance.

Chapter 7

Conclusion and Future Work

In this thesis we considered the problem of evaluating a set expression in which operands are ordered sets and operators are intersection and union. We established our generic difficulty function based on sets of objects visited by proofs. Then we presented an algorithm that takes at most $O\left(D_L(\mathcal{P})\mathcal{H} + ((K_{\max}+k)(|\mathit{Visited}(\mathcal{P})|+1))\mathcal{H} + hk^2\right)$ time, for any proof $\mathcal{P}$ of the given instance, where $\mathcal{H}$ and $K_{\max}$ are defined on pages 228 and 279, respectively, and h and k are the height and the size of the query tree, respectively.

As can be seen, compared to the L-gap cost of a given proof $\mathcal{P}$, the running time has an additional factor $\mathcal{H}$ and an additional additive factor of $\mathcal{H}(K_{\max}+k)|\mathit{Visited}(\mathcal{P})|$. Since for every object o of $\mathit{Visited}(\mathcal{P})$ the term $\mathit{len}(o)W_L(l)$ appears in $D_L(P)$, where l and $\mathit{len}(o)$ are the leaf and the length of the $\mathcal{P}$-gap containing o, the additional additive term of $\mathcal{H}(K_{\max}+k)|\mathit{Visited}(\mathcal{P})|$ can be ignored if the $\mathcal{P}$-gaps have sufficiently big lengths. But for those instances whose proofs visit big subsets of objects the above property does not hold and so a new design of the algorithm is required to avoid this much overhead.

Now consider the factor $\mathcal{H}$ in the term $D_L(\mathcal{P})\mathcal{H}$. It seems that this factor is not due to an inefficient design of the algorithm. Consider the instance corresponding to the input expression $(\{a_1\}\cup\{a_2\}\cup\ldots\{a_n\})\cap(\{b_1\}\cup\{b_2\}\cup\ldots\{b_n\})$. If $a_i = b_i$, for every i, $1 \leq i \leq n$, then the value set of the root will be $\{a_1\ldots,a_n\}$ and so the problem of evaluating a solution to this instance will be equivalent to the problem of sorting the sequence $a_1\ldots,a_n$ and thus will require $\Omega(n\log n)$ time. Even if we limit ourselves to the instances with empty solutions, one can prove that when all a_i's are distinct even numbers and $b_i = a_i + 1$, for

every i, $1 \leq i \leq n$, the algorithm has to sort a_i's and b_j's to make sure that no a_i is equal to a b_j, for the i's and the j's between 1 and n. Now considering the workload function L in which $L^-(v)$ and $L^+(v)$ are the numbers of leaf descendants of v, for every node v, the L-gap cost of a proof will be $O(n)$ while the best algorithm needs $\Omega(n \log n)$ to solve the problem for the instance. Hence, the definition of the cost of a proof should be changed so that it includes this additional logarithmic factor.

There are also a few unanswered questions regarding the results in the thesis. The first group of such questions includes problems regarding how to solve instances with more than one speedy leaf, which are discussed in Section 2.5. Another problem that should be considered is how to select an appropriate workload function for a given instance. One can analyze the effectiveness of the algorithm for the specific choices for the workload function, like the workload function L defined in the previous paragraph. The other approach is to determine the workload of every node in terms of the sizes of the given ordered sets so that the worst case running time is minimized. These solutions require further discussion. Proving lower bounds on running times of the best algorithms and considering the complement and difference set operators are also issues that can be discussed in future work.

Bibliography

[Bar03] J. Barbay. Optimality of randomized algorithms for the intersection problem. In Andreas Albrecht, editor, *Proceedings of the 2nd Symposium on Stochastic Algorithms, Foundations and applications*, pages 26–38. LNCS, Springer-Verlag, 2003.

[BK02] J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the 13th annual ACM-SIAM symposium on Discrete Algorithms*, pages 390–399. SIAM, 2002.

[BK03] J. Barbay and C. Kenyon. Deterministic algorithm for the t-threshold set problem. In *Proceedings of the 14th Annual International Symposium on Algorithms And Computation*, pages 575 – 584. Springer-Verlag, 2003.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[DLOM00] E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proc. of 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 743–752, 2000.

[DLOM01] E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Experiments on adaptive set intersections for text retrieval systems. In *Proceedings of the ALENEX: In-*

ternational Workshop on Algorithm Engineering and Experimentation, LNCS, pages 91 – 104, 2001.

[ECGS97] V. Estivill-Castro and M. De Luz Gasca-Soto. Adaptivity for two problems in networks. In *Proceedings of the 8th Australasian Workshop on Combinatorial Algorithms*, pages 37–49. Queensland University of Technology, 1997.

[ECW92] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24, 4:441–476, 1992.

[GMPR77] L. Guibas, E. McCreight, M. Plass, and J. Roberts. A new representation for linear lists. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 49–60. ACM Press, 1977.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms: 1. Searching and Sorting.* Springer, Berlin, 1984.