Deciding Second-order Logics using Database Evaluation Techniques

by

Gulay Unel

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

©Gulay Unel 2008

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

We outline a novel technique that maps the satisfiability problems of second-order logics, in particular WSnS (weak monadic second-order logic with n successors), S1S (monadic second-order logic with one successor), and of μ-calculus, to the problem of query evaluation of Complex-value Datalog queries. In this dissertation, we propose techniques that use database evaluation and optimization techniques for automata-based decision procedures for the above logics. We show how the use of advanced implementation techniques for Deductive databases and for Logic Programs, in particular the use of tabling, yields a considerable improvement in performance over more traditional approaches. We also explore various optimizations of the proposed technique, in particular we consider variants of tabling and goal reordering. We then show that the decision problem for S1S can be mapped to the problem of query evaluation of Complex-value Datalog queries. We explore optimizations that can be applied to various types of formulas. Last, we propose analogous techniques that allow us to approach μ-calculus satisfiability problem in an incremental fashion and without the need for re-computation. In addition, we outline a top-down evaluation technique to drive our incremental procedure and propose heuristics that guide the problem partitioning to reduce the size of the problems that need to be solved.

Acknowledgements

First of all I would like to thank to my supervisor David Toman for helping me to find a research topic which produced novel techniques and improved my research perspective considerably. I also thank him for his encouragement, patience and support during my PhD studies.

I would like thank to my thesis committee members Grant Weddell, Richard Trefler, John Thistle, and Stéphane Demri for their valuable comments on my final thesis document.

Last but not least, I would like to thank to my family, former teachers, and friends. I would not be in this stage, now writing my acknowledgements section as a last touch to my accepted PhD thesis, without them.

Contents

List of Figures

Chapter 1

Introduction

Logics provide means to specify regular properties of systems in a succinct way. In this dissertation we consider weak monadic second order logics with one and two successors (WS1S and WS2S), second order logics with one successor (S1S) and μ-calculus. These logics are decidable by the virtue of a connection to automata theory. Standard decision procedures for the satisfiability problem consist of translating a formula to an automaton accepting the models of the formula and checking whether the automaton is empty or not.

The automata-theoretic approach for monadic logics over finite words was developed by Büchi, Elgot, and Trakhtenbrot [9, 25, 89]. It was then extended to infinite words by Büchi [10], to finite trees by Thatcher and Wright [87], and generalized to infinite trees by Rabin [73]. Another automata theoretic construction was developed for μ-calculus [43, 98] and could be used, in turn, for reasoning in expressive description logics. The practical use of this connection was investigated for temporal logics and fixed-point logics which led to the theory of model checking [54, 19, 31, 100]. However, automata-based decision procedures do not enjoy the success predicted by the accompanying theory and are mostly used for showing decidability and complexity bounds rather than for implementation purposes. Indeed, in many cases, theoretically sub-optimal approaches, such as tableaux equipped with appropriate *blocking conditions* that prevent infinite expansions, are more successful [2, 41]. This rather surprising observation can be traced to severe difficulties in implementing automata-based decision procedures, in particular when inherently infinite models are considered. The main focus of this work is proposing implementation approaches for automata-based decision procedures for the above logics based on query evaluation and optimization techniques from database theory and logic programming.

First, we consider the logics WS1S and WS2S and propose an implementation of the decision procedure based on representing automata by logic programs. Given a WS1S/WS2S formula an automaton can be constructed inductively starting from the atomic subformulas and applying automata operations for the logical connectives and quantifiers. We represent this construction as a complex-value datalog (Datalog^{cv}) program consisting of views. The emptiness check on the automaton is then reduced to posing a query on these views. This representation combined with Datalog^{cv} program execution techniques, such as *Magic Set transformation* [3] and *SLG resolution*, a top-down resolution-based approach augmented with memoing [16, 17], guide the automaton construction such that intuitively only the states needed to show the emptiness are generated. We also conducted experiments that demonstrate the benefits of the proposed method over more standard approaches.

In our work, we classify formulas as conjunctions, negated and existential formulas and propose heuristics and optimizations depending on the type of the formulas. Our main focus is on conjunctive formulas where the standard automata-theoretic approach fails due to the state space explosion problem as the number of conjunctions increases. The difficulties are especially apparent when determining *logical consequences* of large *theories* of the form $\{\varphi_1, \ldots \varphi_n\} \models \varphi$, are considered. In this case, the automata-theoretic method constructs the automaton for the formula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi$ which can be quite large and then checks for its emptiness. However, our approach gives good results since we only construct the states required for answering the emptiness problem. In addition, these types of formulas are amenable to optimizations such as formula rewriting and goal reordering where different formula/view definition rewritings result in different query evaluation performances.

We show that a similar approach can be used for implementing S1S decision procedures: we provide a mapping for the complementation operation to complex-value datalog views which differs from the complementation operation in WSnS. Automata-based decision procedures are relatively simple in WSnS when compared to S1S. The complementation operation is complicated in the decision procedure for S1S. We propose a solution to this problem using the method we outlined for WSnS extended with the different complementation operation for S1S plus an optimization method for negated conjunctions which need to be adressed because of the exponential overhead of the complementation operation especially when we need to compute it on large conjunctions.

The automata-based approach for μ-calculus is usually based on translating a given formula to an *alternating parity automaton*. The emptiness test for alternating parity automaton, in particular when based on Safra's determinization approach [78, 79], is difficult to implement. This issue, for μ-calculus formulas, was addressed by using decision procedures that does not use Safra's construction based on transforming an alternating parity automaton to a non-deterministic Büchi automaton while preserving emptiness [55]. However, even this improvement does not yield a practical reasoning procedure.

Unlike the WSnS case, the automata-based decision procedure for μ-calculus does not have an inductive construction. This makes the problem even harder for our approach based on logic programming. In our work for μ-calculus, we explore techniques that attempt to remedy the mentioned difficulties by proposing an incremental and interleaved approach to constructing the automaton corresponding to the logical implication problem while simultaneously testing for satisfiability of the fragments constructed so far. In this work, we show how the decision problem can be split into a sequence of simpler problems, we show how the larger problems can be constructed from the simpler ones, and show how the top-down query evaluation techniques enhanced with memoing can be used to drive such an incremental computation.

The main contributions of this thesis are as follows:

- We show a connection between the automata based decision procedures and query evaluation in complex value datalog by proposing decision procedures for WSnS, S1S, and μ-calculus based on mapping the satisfiability problem to a query evaluation problem on a set of views.
- We present experimental results for WS1S and WS2S that show the benefits of our approach over an other implementation based on the standard automata based approach.
- We outline an incremental technique for the automata based decision procedure for conjunctive μ-calculus formulas.
- We propose heuristics and optimizations for different types of formulas for each outlined decision procedure.

1.1 Organization of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2 we formally introduce logics WS1S/WS2S, S1S, μ-calculus, their connection to finite automata, and we define Datalog^{cv} queries, state their computational properties, and discuss techniques for query evaluation. Chapter 3 presents our work on how Datalog^{cv} queries and views can be used to represent finite automata for deciding WS1S and WS2S, how to implement automata-theoretic operations on the representation, and experimental results for the proposed methods. In Chapter 4 we show that a similar approach to the one for WSnS can be used for S1S decision procedures providing a mapping for the negation operation to complex-value datalog views which differs from the negation operation in WSnS. We present an incremental technique for μ-Calculus decision procedures in Chapter 5. Finally, conclusions and future research directions are given in Chapter 6.

Chapter 2

Background and Definitions

In this chapter, we introduce the necessary theoretical foundations of logics, automata and database evaluation techniques. We review definitions for logics and automata in section 2.1. We define Datalog for Complex Values and present related query evaluation techniques in section 2.2.

2.1 Logics and Automata

In this section, we give an overview of several variants of monadic second order logics, automata on finite and infinite objects, and their connection which in turn provides decision procedures for the logics.

2.1.1 Logics

We define Monadic Second Order Logics and μ-Calculus as follows.

Monadic Second Order Logics

First, we define the syntax and semantics of the second-order logic of one and two successors.

Definition 2.1 Let $\mathsf{Var} = \{x, y, z, \ldots\}$ be an (infinite) set of variable names, the formulas of second-order logics are defined as follows.

- the expressions $s(x, y)$, $x \subseteq y$ for x, y second-order variables are atomic formulas, and

- given formulas φ and ϕ and a variable x, the expressions $\varphi \wedge \phi$, $\neg\varphi$, and $\exists x : \varphi$ are also formulas.

Additional common syntactic features can be defined as follows. Variables for individuals (first-order variables) can be simulated using second-order variables bound to singleton sets; a property expressible in WS1S. Thus we allow $x \in y$ for $x \subseteq y$ whenever we know that x is a singleton. We also use the standard abbreviations $\varphi \vee \psi$ for $\neg(\neg\varphi \wedge \neg\psi)$, $\varphi \rightarrow \psi$ for $\neg\varphi \vee \psi$, and $\forall x : \varphi$ for $\neg\exists x : \neg\varphi$.

The semantics of WS1S and S1S is defined w.r.t. the set of natural numbers (successors of 0); second-order variables are interpreted as finite sets of natural numbers in WS1S and as (possibly) infinite sets of natural numbers in S1S. The interpretation of the atomic formula $s(x, y)$ is fixed to relating singleton sets $\{n\}$ and $\{n + 1\}$, $n \in \mathbf{N}$.[1] Similarly, the semantics of WS2S and S2S are defined over an infinite binary tree $\mathbf{T} = (0 + 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$; first-order variables are interpreted as nodes of the binary tree, and second order variables are interpreted as finite subsets of the nodes in WS2S and as (possibly) infinite subsets of the nodes in S2S.

Definition 2.2 The definition of truth of a formula is defined over a transition system $\mathcal{T}$ (over $\mathbf{N}$ in WS1S/S1S and over $\mathbf{T}$ in WS2S/S2S). A valuation is defined as $\mathcal{D} : \mathsf{Var} \rightarrow 2^{\mathcal{Q}}$ where $\mathcal{Q} = \mathbf{N}$ in WS1S and S1S and $\mathcal{Q} = \mathbf{T}$ in WS2S and S2S. Given a valuation $\mathcal{D}$ and a transition system $\mathcal{T}$ we have:

- $\mathcal{T}, \mathcal{D} \models x \subseteq y$ if $\mathcal{D}(x) \subseteq \mathcal{D}(y)$
- $\mathcal{T}, \mathcal{D} \models s(x, y)$ if $\mathcal{D}(x)$ and $\mathcal{D}(y)$ are singletons $\{s_x\}$, $\{s_y\}$ and s_y is a successor of s_x
- $\mathcal{T}, \mathcal{D} \models \varphi \wedge \phi$ if $\mathcal{T}, \mathcal{D} \models \varphi$ and $\mathcal{T}, \mathcal{D} \models \phi$
- $\mathcal{T}, \mathcal{D} \models \neg\varphi$ if $\mathcal{T}, \mathcal{D} \not\models \varphi$
- $\mathcal{T}, \mathcal{D} \models \exists x : \varphi$ if there exists $M \subseteq \mathcal{Q}$ such that $\mathcal{D}(x) = M$ and $\mathcal{T}, \mathcal{D}[x \leftarrow M] \models \varphi$.

Note that for $\mathbf{N}$ there is one successor relation and for $\mathbf{T}$ there are two successor relations.

Example 2.3 The formula $\varphi = \exists x, y, z : x \subset y \wedge y \subset z$ is interpreted as "there exists sets x, y, z such that x is a subset of y and y is a subset of z".

[1]The atomic formula $s(x, y)$ is often written as $y = s(x)$ in literature, emphasizing its nature as a *successor* function.

Example 2.4 We can write a formula defining the property: "x is a singleton" as follows:

- we define: "the set x is equal to set y ($x = y$)" by:
  $\forall z : z \subseteq x \rightarrow z \subseteq y \wedge \forall t : t \subseteq y \rightarrow t \subseteq x$
- we define: "x is an empty set ($x = \emptyset$)" by:
  $\neg \exists y : y \neq x \wedge y \subseteq x$
- then "x is a singleton" can be defined by:
  $e = \emptyset \wedge e \subseteq x \wedge \exists z : ((z \neq e \wedge z \subseteq x) \wedge \neg \exists y : (y \neq e \wedge y \neq z \wedge y \subseteq x))$

μ-Calculus

The propositional μ-calculus was introduced by Kozen [49] for specifying properties of concurrent programs following earlier studies of fixpoint calculi [28, 67, 69]. The propositional μ-calculus is augmented with least and greatest fixpoint operators and expressively subsumes most propositional program logics, including dynamic logics [33, 50, 68, 38, 39] and temporal logics [72, 71, 70, 27, 42, 102]. The syntax and semantics of μ-calculus [7] is given below:

Definition 2.5 Let $\mathsf{Var} = \{x, y, z, \ldots\}$ be an (infinite) set of variable names, Prop a set of *atomic propositions*, and $\mathcal{L} = \{a, b, \ldots\}$ a finite set of *labels*. The set of L_μ formulas (with respect to Var, Prop, $\mathcal{L}$) is defined as follows:

- $p \in \mathsf{Prop}$ and $z \in \mathsf{Var}$ are formulas.
- If ϕ_1 and ϕ_2 are formulas, so is $\phi_1 \wedge \phi_2$.
- If ϕ is a formula, so are $[a]\phi$, $\neg\phi$, and $\mu z.\phi$ provided that every free occurrence of z in ϕ occurs positively (within the scope of an even number of negations).

A sentence is a formula that does not contain free variables.

If a formula is written as $\phi(z)$, it means that the subsequent writing of $\phi(\psi)$ is used for ϕ with ψ substituted for all free occurences of z.

We use derived operators such as $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\langle a \rangle \phi$ for $\neg[a]\neg\phi$, $\nu z.\phi(z)$ for $\neg\mu z.\neg\phi(\neg z)$, $[K]\phi$ for $\bigwedge_{a \in K}[a]\phi$, $[-]\phi$ for $[\mathcal{L}]\phi$, respectively.

Formulas of L_μ are interpreted with respect to labeled transition systems over Prop in which nodes are labeled by propositional assignments and edges by elements of $\mathcal{L}$.

Definition 2.6 An L_μ structure $\mathcal{T}$ (over $\mathsf{Prop}, \mathcal{L}$) is a labeled transition system, namely a set $\mathcal{Q}$ of states and a transition relation $\rightarrow \subseteq \mathcal{Q} \times \mathcal{L} \times \mathcal{Q}$ (written as $s \xrightarrow{a} t$), together with an interpretation $\mathcal{D}_{\mathsf{Prop}} : \mathsf{Prop} \rightarrow 2^{\mathcal{Q}}$ for the atomic propositions.

Given a structure $\mathcal{T}$ and an interpretation $\mathcal{D} : \mathsf{Var} \rightarrow 2^{\mathcal{Q}}$ of the variables, the set $\|\phi\|^{\mathcal{T}}_{\mathcal{D}}$ of states satisfying a formula ϕ is defined as follows:

- $\|p\|^{\mathcal{T}}_{\mathcal{D}} = \mathcal{D}_{\mathsf{Prop}}(p)$
- $\|z\|^{\mathcal{T}}_{\mathcal{D}} = \mathcal{D}(z)$
- $\|\neg\phi\|^{\mathcal{T}}_{\mathcal{D}} = \mathcal{Q} \setminus \|\phi\|^{\mathcal{T}}_{\mathcal{D}}$
- $\|\phi_1 \wedge \phi_2\|^{\mathcal{T}}_{\mathcal{D}} = \|\phi_1\|^{\mathcal{T}}_{\mathcal{D}} \cap \|\phi_2\|^{\mathcal{T}}_{\mathcal{D}}$
- $\|[a]\phi\|^{\mathcal{T}}_{\mathcal{D}} = \{s | \forall t : s \xrightarrow{a} t \Rightarrow t \in \|\phi\|^{\mathcal{T}}_{\mathcal{D}}\}$
- $\|\mu z.\phi\|^{\mathcal{T}}_{\mathcal{D}} = \bigcap\{S \subseteq \mathcal{Q} | S \supseteq \|\phi\|^{\mathcal{T}}_{\mathcal{D}[z:=S]}\}$

where $\mathcal{D}[z := S]$ is the valuation which maps z to S and otherwise agrees with $\mathcal{D}$. By duality, the definitions for the derived operators are as follows:

- $\|\phi_1 \vee \phi_2\|^{\mathcal{T}}_{\mathcal{D}} = \|\phi_1\|^{\mathcal{T}}_{\mathcal{D}} \cup \|\phi_2\|^{\mathcal{T}}_{\mathcal{D}}$
- $\|\langle a\rangle\phi\|^{\mathcal{T}}_{\mathcal{D}} = \{s | \exists t : s \xrightarrow{a} t \wedge t \in \|\phi\|^{\mathcal{T}}_{\mathcal{D}}\}$
- $\|\nu z.\phi\|^{\mathcal{T}}_{\mathcal{D}} = \bigcup\{S \subseteq \mathcal{Q} | S \subseteq \|\phi\|^{\mathcal{T}}_{\mathcal{D}[z:=S]}\}$

The semantics of μ-calculus is defined with respect to a *Kripke structure* $K = \langle W, R, L\rangle$ over (Var, Prop, $\mathcal{L}$), where W is a set of points, $R : \mathcal{L} \rightarrow 2^{W\times W}$ is a labeled transition relation over W, and $L : \mathsf{Prop} \rightarrow 2^W$ assigns each atomic proposition a set of points.

Satisfiable L_μ formulas enjoy the *tree model* property which means that if a sentence is satisfiable then it is satisfiable by a bounded-degree infinite tree structure. A tree structure is a Kripke structure $\langle W, R, L\rangle$ where W is a tree and for each label l if $(u, v) \in R(l)$, then v is a successor of u. Hence, $\mathbf{N}$ and $\mathbf{T}$ (with a valuation $\mathcal{D}$) are also Kripke structures. The standard way of proving the tree-model property is to take a model and straightforwardly *unravel* it [97]. Furthermore, every formula ϕ in L_μ is equivalent to a formula φ in S2S which means the interpretation of each free proposition $p_x \in \mathsf{Prop}$ in ϕ maps to the valuation for a free variable x in φ. Note that $\mathcal{D}_{\mathsf{Prop}}(p)$ is the same as $\mathcal{D}(x)$ in S2S.

Example 2.7 The formula $\psi = \mu x.(p \vee [a]x)$ is interpreted as "p eventually holds for all a-paths". We can also write a second order logic formula $\phi = \forall x : ((\forall z : z \in p \vee (\forall z' : s_a(z, z') \rightarrow z' \in x) \rightarrow z \in x) \rightarrow y \in x)$ to express this property where s_a is a successor relation defined on a-paths.

2.1.2 Automata

In this section, we introduce automata on finite and infinite strings and trees.

Definition 2.8 Given a (finite) set D of directions, a D-tree is a set $T \subseteq D^*$ such that if $x \cdot c \in T$ (an extension of x with c), where $x \in D^*$ and $c \in D$, then also $x \in T$. If $T = D^*$, we say that T is a full D-tree. The empty word ϵ is the root of T and the elements of T are called *nodes*. A path π of a tree T is a set $\pi \subseteq T$ such that $\epsilon \in \pi$ and for every $x \in \pi$ either x is a leaf or there exists a unique c such that $x \cdot c \in \pi$. If $|D| = 1$ then $W \subseteq D^*$ is a word. The infinite binary tree $\mathbf{T} = (0+1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$ is a full D-tree where $D = \{0, 1\}$. Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau \rangle$ where T is a tree and $\tau : T \rightarrow \Sigma$ maps each node of T to a letter in Σ.

For a set X, $\mathcal{B}^+(X)$ is the set of positive Boolean formulas over X; for a set $Y \subseteq X$ and a formula $\phi \in \mathcal{B}^+(X)$, we say that Y satisfies ϕ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes ϕ true. An alternating tree automaton is $A = \langle \Sigma, D, Q, S, \delta, F \rangle$, where Σ is the input alphabet, D is a set of directions, Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(D \times Q)$ is a transition function, $S \subseteq Q$ is a set of initial states, and F specifies the acceptance condition.

An alternating automaton A runs on Σ-labeled full D-trees. A run of A over a Σ-labeled D-tree $\langle T, \tau \rangle$ is a $(T \times Q)$-labeled tree $\langle T_r, r \rangle$ such that:

1. $\epsilon \in T_r$ and $r(\epsilon) = \langle \epsilon, q_i \rangle$ where $q_i \in S$.

2. For every $y \in T_r$ such that $r(y) = \langle x, q \rangle$ there is a set
$$\{(c_0, q_0), (c_1, q_1), \ldots (c_{n-1}, q_{n-1})\} \subseteq D \times Q$$
that satisfies $\delta(q, \tau(x))$, and for all $0 \leq j < n$, $y \cdot j \in T_r$, $r(y \cdot j) = \langle x \cdot c_j, q_j \rangle$.

For automata on finite words and trees a run $\langle T_r, r \rangle$ is accepting if all its paths end in a state $f \in F$.

For automata on infinite words and trees, a run $\langle T_r, r \rangle$ is accepting if all its infinite paths satisfy an acceptance condition. The set of states on a path $\pi \subseteq T_r$ that appear infinitely often is denoted with $\mathsf{inf}(\pi)$ where $\mathsf{inf}(\pi) \subseteq Q$ and $q \in \mathsf{inf}(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. The types of acceptance conditions are defined as follows:

- A path π satisfies Büchi acceptance condition $F \subseteq Q$ if $\mathsf{inf}(\pi) \cap F \neq \emptyset$.
- A path π satisfies co-Büchi acceptance condition $F \subseteq Q$ if $\mathsf{inf}(\pi) \cap F = \emptyset$.

- A path π satisfies parity acceptance condition $F = \{F_1, F_2, \ldots, F_h\}$ with $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_h = Q$ if the minimal index i for which $\mathsf{inf}(\pi) \cap F_i \neq \emptyset$ is even. The number h of sets in F is called the *index* of the automaton.

An automaton accepts a word/tree if there exists a run that accepts it. The set of all Σ-trees that are accepted by A is denoted by $\mathcal{L}(A)$.

An alternating automaton is:

- nondeterministic: if the formulas (c_1, q_1) and (c_2, q_2) appear in δ and are conjunctively related, then $c_1 \neq c_2$,
- universal: if all the formulas that appear in δ are conjunctions of atoms in $D \times Q$,
- deterministic: if it satisfies the conditions for being nondeterministic and universal at the same time.

Following shorthand notation is used to describe types of automata on infinite words and trees. The first letter describes the transition structure and is one of "D" (deterministic), "N" (nondeterministic),"U" (universal), and "A" (alternating). The second letter describes the acceptance condition: "P" (parity), "B" (Büchi), and "C" (co-Büchi) are used. The third letter describes the objects on which the automata run: "W" (infinite words), and "T" (infinite trees) are used.

Example 2.9 The automaton $A = \{\Sigma, D, Q, \{q_0\}, \delta, F\}$ where:

$$\begin{aligned}
&Q = \{q_0, q_1\}\\
&\Sigma = \{a, b\}\\
&\delta_1(q_0, a) = ((1, q_0) \wedge (2, q_1)) \vee (2, q_0)\\
&\delta_1(q_0, b) = (1, q_1) \wedge (2, q_1)\\
&\delta_1(q_1, a) = (1, q_0) \vee ((1, q_1) \wedge (2, q_0))\\
&\delta_1(q_1, b) = (1, q_1) \vee (2, q_1)
\end{aligned}$$

is nondeterministic.

Example 2.10 Let the universal Büchi automaton $A = \{\Sigma, D, Q, \{q_0\}, \delta, F\}$ where:

$$\begin{aligned}
&Q = \{q_0, q_1\},\\
&\Sigma = \{a, b\},\\
&\delta_1(q_0, a) = (1, q_0) \wedge (2, q_0),\\
&\delta_1(q_0, b) = (1, q_1) \wedge (2, q_1),\\
&\delta_1(q_1, a) = (1, q_0) \wedge (2, q_0),\\
&\delta_1(q_1, b) = (1, q_1) \wedge (2, q_1), \text{ and}\\
&F = \{q_1\}.
\end{aligned}$$

This automaton accepts trees such that each path contains infinitely many bs.

Bottom-up Tree Automata on Finite Trees

Bottom-up tree automata on finite trees is equivalent to nondeterministic (top-down) tree automata on finite trees given in Definition 2.8. There are two differences with bottom-up tree automata: first, $Q_i \subseteq Q$, the set of its initial states replaces F; second, its transition rules are the converse, that is, $\delta : \mathcal{B}^+(D \times Q) \times \Sigma \rightarrow Q$. In bottom-up tree automata, a transition $((c_0, q_0), (c_1, q_1), \ldots, (c_k, q_k), a, q)$ allows to proceed from states q_0, q_1, …, q_k at the successor nodes $u.c_0, u.c_1, \ldots, u.c_k$ of a node u to q at u while reading letter a as label of u. A run of A on an input tree T is a mapping ρ from the nodes of T to the states of A. A run ρ is called successful if $\rho(u_0) \in F$ where u_0 is the root of the tree.

2.1.3 Logic-Automata Connection

The connection between logic and automata was first considered by Büchi [9] and Elgot [25]. They have shown that monadic second-order logic over finite words and finite automata have the same expressive power, and we can transform formulas of this logic to finite automata which means that for every formula ϕ we can construct an automaton accepting the models of ϕ and vice versa. Later, this connection was extended to monadic second order logics over finite trees and finite automata [87]. Büchi [10], McNaughton [59], and Rabin [73] proved that monadic second-order logic over infinite words (and trees) and automata on infinite words (and trees) also have the same expressive power. The practical use of this connection was investigated for temporal logics and fixed-point logics [26, 29, 32, 45, 61, 62, 95, 96, 99, 101] which led to the theory of model checking [54, 19, 31, 100]. Efficient algorithms for temporal logics based on logic-automata connection are proposed [20, 21, 34, 35, 44, 82]. A tutorial and brief survey on constructing automata from temporal logic formulas can be found in [103]. Automata theoretic construction for μ-calculus [29, 30, 31, 43, 66, 84, 98] could also be used, in turn, for reasoning in expressive description logics. An extensive survey on automata and logics can be found in [36, 88].

Monadic Second Order Logics and Automata

Computational properties of automata are a basis for solutions to many problems. One of these problems is building decision procedures for various logics. In this section we outline the connection between automata and monadic second order logics WS1S/WS2S and focus on constructing automata from WS1S formulas. The logic-automaton connection can be generalized to build decision procedures for different logics such as second order logics with

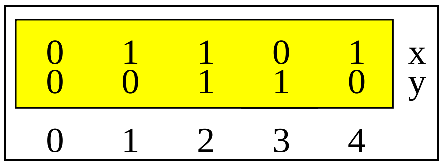


Figure 2.1: String representing the interpretation $\mathcal{D}(x) = \{1, 2, 4\}, \mathcal{D}(y) = \{2, 3\}$.

one or two successors (S1S or S2S). Automata that accept infinite regular languages can be used for this purpose.

The crux of the connection for monadic second order logics lies in an observation that, for every formula, there is an automaton that accepts exactly the models of a given formula [88]. Models can be represented by finite strings in WS1S. Since each variable of WS1S is interpreted by a finite set of natural numbers, such an interpretation can be captured by a finite string. Satisfying interpretations of formulas (with k free variables) can be represented as sets of strings over $\{0, 1\}^k$. The i-th component corresponds to the interpretation of the i-th variable and is called a *track*. It turns out that sets of the above strings form regular languages and thus can be recognized using an automaton. Satisfiability then reduces to checking for non-emptiness of the language accepted by such an automaton. Similarly, there is an automaton that accepts the (tree representations of) models of a given formula for WS2S.

Example 2.11 Suppose we have a formula $\phi = x \subseteq y$ and a valuation $\mathcal{D} : \{x, y\} \rightarrow 2^{\mathbf{N}}$ such that $\mathcal{D}(x) = \{1, 2, 4\}$, and $\mathcal{D}(y) = \{2, 3\}$ then we can represent this valuation (interpretation) by the string given in Figure 2.1, the first track of the string is for x and the second track is for y.

Given a WS1S/WS2S formula φ, the automaton A_φ can be effectively constructed starting from automata for atomic formulas using automata-theoretic operations.

Proposition 2.12 *Let φ be a WS1S/WS2S formula. Then there is an automaton A_φ that accepts exactly the (string representations of) models of φ and φ is satisfiable if and only if $L(A_\varphi) \neq \emptyset$, where $L(A)$ is the language accepted by A.*

Example 2.13 The automaton A_φ for the formula $\varphi = x \subseteq y$ is shown in the left part of Figure 2.2, the complement automaton $A_{\neg\varphi}$ that represents $\neg\varphi = \neg(x \subseteq y)$ is shown in the right part of Figure 2.2. The labels on the edges are elements of the alphabet of the automaton that capture the valuations of variables allowed for a particular transition. The tracks in the strings accepted by the automata represents the valuation for the variables x (first track), and y (second track).

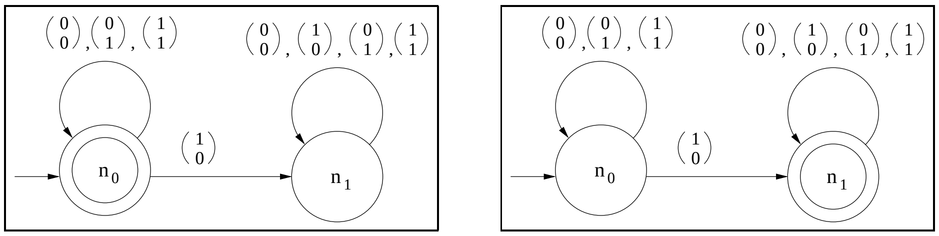

$\binom{0}{0},\binom{0}{1},\binom{1}{1}$
$\binom{0}{0},\binom{1}{0},\binom{0}{1},\binom{1}{1}$
$\binom{1}{0}$
n$_0$
n$_1$
$\binom{0}{0},\binom{0}{1},\binom{1}{1}$
$\binom{0}{0},\binom{1}{0},\binom{0}{1},\binom{1}{1}$
$\binom{1}{0}$
n$_0$
n$_1$

Figure 2.2: Automata representing the formulas $x \subseteq y$ and $\neg(x \subseteq y)$.

Similarly, the automaton $A_{\varphi\wedge\phi}$ is the product automaton of A_φ and A_ϕ and accepts $L(A_\varphi) \cap L(A_\phi)$, the satisfying interpretations of $\varphi \wedge \phi$. The automaton $A_{\exists x:\varphi}$, the projection automaton of A_φ, accepts satisfying interpretations of $\exists x : \varphi$. Intuitively, the automaton $A_{\exists x:\varphi}$ acts as the automaton A_φ for φ except that it is allowed to guess the bits on the track of the variable x. While checking for emptiness can be done in time polynomial in the size of an automaton, the size of A_φ is non-elementary in the size of φ (more precisely, in the depth of quantifier alternation of φ). This bound is tight for WSnS decision problem yielding an overall non-elementary decision procedure.

Similarly, for every S1S formula, there is an automaton on infinite words that accepts exactly the (string representations of) models of a given formula. Note that the automaton has an acceptance condition defined for infinite words in this case.

Given a S1S formula φ, the automaton A_φ can be effectively constructed starting from automata for atomic formulas using automata-theoretic operations. As in the case of WS1S an automaton can be constructed for each atomic formula. The automaton $A_{\varphi\wedge\phi}$ is the product automaton of A_φ and A_ϕ and accepts $L(A_\varphi) \cap L(A_\phi)$, the satisfying interpretations of $\varphi \wedge \phi$. The automaton $A_{\exists x:\varphi}$, the projection automaton of A_φ, accepts satisfying interpretations of $\exists x : \varphi$. The complementation operation is not as trivial as the one given for WS1S.

Proposition 2.14 *Let φ be a S1S formula. Then there is a Büchi automaton A_φ that accepts exactly the (string representations of) models of φ and φ is satisfiable if and only if $L(A_\varphi) \neq \emptyset$, where $L(A)$ is the language accepted by A.*

Theorem 2.15 *The complexity of the automata-based decision procedure for monadic second order logics is non-elementary [60, 83].*

μ-Calculus and Automata

Automata theoretic decision procedures for the μ-calculus and its fragments are given in [31, 43, 84, 99, 98]. Most of these techniques rely on the translation of μ-calculus formulas to alternating automata.

The tree model property of μ-calculus formulas provides a link to automata theory. If a formula ϕ is satisfiable then it is satisfiable at the root of a tree whose branching degrees are bounded by the length of ϕ ($\|\phi\|$) [98]. Satisfiability of a L_μ formula is equivalent to checking whether a corresponding alternating parity automaton that accepts tree models of the formula is non-empty whose number of states is $O(\|\phi\|)$.

Example 2.16 Consider a formula $\varphi = \nu x.(\psi \wedge \langle - \rangle x)$ where $\psi = \mu y.(b \vee \langle - \rangle y)$ and $\mathsf{Prop} = \{a, b\}$ models of φ are tree models that have at least one path with infinitely many b's. An APT (on binary trees) accepting models of φ is $A = \{\Sigma, D, Q, \{q_1\}, \delta, F\}$ where:

$$
\begin{aligned}
&Q = \{q_0, q_1\}\\
&\delta(q_0, a) = (1, q_0) \vee (2, q_0)\\
&\delta(q_0, b) = (1, q_1) \vee (2, q_1)\\
&\delta(q_1, a) = (1, q_0) \vee (2, q_0)\\
&\delta(q_1, b) = (1, q_1) \vee (2, q_1)\\
&F = \{\{q_0\}, \{q_0, q_1\}\}
\end{aligned}
$$

The connection between L_μ formulas and alternating automata is captured by the following theorem [24, 43, 98].

Theorem 2.17 *Let $\varphi \in L_\mu$. Then there is an alternating parity tree automaton A_φ that can be constructed effectively from φ, such that the language of trees accepted by A_φ is the set of tree models of φ.*

Theorem 2.18 *The complexity of testing emptiness of alternating automata is ExpTime-complete.*

Hence, it remains to solve the emptiness problem for alternating automata to decide the satisfiability of μ-calculus formulas.

2.2 Datalog for Complex Values

In this section we define a query language that serves as the target of our approach to WS1S/WS2S decision procedure.

Complex Data Model.

The complex-value data model is an extension of the standard relational model that allows tuples and finite sets to serve as values in the place of atomic values [1]. Each value is assigned a finite *type* generated by the type grammar "$\tau := \iota \mid [\tau_1, \ldots, \tau_k] \mid \{\tau\}$", where ι stands for the type of uninterpreted atomic constants, $[\tau_1, \ldots, \tau_k]$ for a k-tuple consisting of values belonging to the types $\tau_1, \ldots, \tau_k$, respectively, and $\{\tau\}$ for a finite set of values of type τ. Relations are interpreted as sets of values of a given type[2]. The model is equipped with several *built-in* relations, e.g., the equality $=$ (extended to all types), the subset relation $\subseteq$ (defined for set types), the tuple constructor (that relates tuples of values to the individual values), the singleton set constructor (relating values of a type to singleton sets of the appropriate set type), etc.

Complex-value Queries.

The extended data model induces extensions to relational query languages and leads to the definition of *complex-value relational calculus* (calc$^{\text{cv}}$) and a deductive language for complex values, Datalog$^{\text{cv}}$—the language of Horn clauses built from literals whose arguments range over complex-valued variables and constants [3, 77]. Datalog$^{\text{cv}}$ programs and queries are defined as follows:

Definition 2.19 A Datalog$^{\text{cv}}$ *atom* is a predicate symbol with variables or complex-value constants as arguments.

A Datalog$^{\text{cv}}$ *database (program)* is a finite collection of Horn clauses of the form $h \leftarrow g_1, \ldots, g_k$, where h (called head) is an atom with an optional *grouping specification* and $g_1, \ldots, g_k$ (called goals) are literals (atoms or their negations).

The grouping is syntactically indicated by enclosing the *grouped* argument in the $\langle\cdot\rangle$ constructor; the values then range over the set type of the original argument.

We require that in every occurrence of an atom the corresponding arguments have the same finite type and that the clauses are stratified with respect to negation.

A Datalog$^{\text{cv}}$ query is a clause of the form $\leftarrow g_1, \ldots, g_k$.

Evaluation of a Datalog$^{\text{cv}}$ query (with respect to a Datalog$^{\text{cv}}$ database P) determines whether $P \models g_1, \ldots, g_k$.

Datalog$^{\text{cv}}$ is equivalent to the complex-value calculus in expressive power [1]. However, the ability to express transitive closure without resorting to the powerset construction aids our

[2]We use relations of arity higher than one as a shorthand for sets—unary relations—of tuples of the same arity.

goal of using Datalog$^{\text{cv}}$ to represent finite automata and to test for emptiness.

Proposition 2.20 *The complexity of* Datalog$^{\text{cv}}$ *query evaluation is non-elementary [51].*

Note that the complexity matches that of decision procedures for WS1S/WS2S and thus mapping of WS1S/WS2S formulas to Datalog$^{\text{cv}}$ queries can be done efficiently.

To simplify the notation in the following we allow terms constructed of constants, variables, and finite number of applications of tuple and set constructors to appear as arguments of atoms. For example $p(\{x\}, y) \leftarrow q([x, y])$ is a shorthand for $p(z, y) \leftarrow q(w), w = [x, y], z = \{x\}$, where $w = [x, y]$ is an instance of a tuple constructor and $z = \{x\}$ of a set constructor built-in relations as discussed in our overview of the complex-value data model.

Example 2.21 A set can be constructed by listing all of its elements using set enumeration. Consider the following relation which represents the starting node for an automaton:
$\mathsf{Start}(0)$
The following rule generates a singleton set containing the starting node and stores it in a new relation:
$\mathsf{Start}^d(\{n\}) \leftarrow \mathsf{Start}(n)$
As a result the rule generates:
$\mathsf{Start}^d(\{0\})$

Example 2.22 A set is constructed by defining a property to be satisfied by its elements with set grouping. Consider the following facts which represent transitions for an automaton:
$\mathsf{Transition}(1, 2)$
$\mathsf{Transition}(1, 3)$
$\mathsf{Transition}(2, 3)$
The following rule groups all target nodes of a particular node and stores it in a set:
$\mathsf{Transition}^d(n_1, \langle n_2 \rangle) \leftarrow \mathsf{Transition}(n_1, n_2)$
As a result the rule generates:
$\mathsf{Transition}^d(1, \{2, 3\})$
$\mathsf{Transition}^d(2, \{3\})$

2.2.1 Query Evaluation in Datalog$^{\text{cv}}$

The basic technique for evaluation of Datalog$^{\text{cv}}$ programs is commonly based on a fixed-point construction of the minimal Herbrand model (for Datalog$^{\text{cv}}$ programs with *stratified negation* the model is constructed w.r.t. the stratification) and then testing whether a ground (instance of the) query is contained in the model. The type restrictions guarantee that the fixpoint iteration terminates after finitely many steps. While the naive fixed-point computation can be implemented directly, efficient query evaluation engines use more involved techniques such as the semi-naive evaluation, goal/join ordering, etc. In addition, whenever the query is known as part of the input, techniques that allow constructing only the relevant parts of the minimal Herbrand model have been developed. Among these the most prominent are the magic set rewriting (followed by subsequent fixed-point evaluation) [4, 64] and the top-down resolution with memoing—the SLG resolution [16, 17].

Magic-set Evaluation

The main idea behind this approach is to restrict the values derived by a fixpoint computation to those that can potentially aid answering a given query. This is achieved by program transformation based on adding *magic predicates* to clauses that limit the breadth of the fixpoint computation at each step. These predicates are *seeded* by the values in the query (as those are the only ones the user desires to derive); more values are added to the interpretations of the magic predicates by means of additional clauses that *relax* the limit depending on what additional subqueries for a particular predicate need to be asked to answer the original query. This process then becomes a part of the fixpoint evaluation itself.

Definition 2.23 Let x be a variable and P a Datalog$^{\text{cv}}$ program. We say that x is free with respect to P if for a valuation θ (an assignment of values to variable names) such that $\theta \models P$ the valuation $\theta[x/a] \models P$ for all a. Otherwise we say that x is bound in P.

Definition 2.24 (Adornment) Let h be an atom of arity k. An adornment A for this atom is a string over $\{b, f\}$ of length k. We say that the ith argument of h is adorned by $b(f)$ if the ith position in A is $b(f)$, respectively. We say that A is an adornment of h with respect to P, where P is a Datalog$^{\text{cv}}$ program, if the ith position of A is b if the ith argument of h is bound in P or the ith position of A is f if the ith argument of h is free in P.

Let $h \leftarrow d, g_1, \ldots, g_k$, be a clause and A an adornment of h. Then an adornment of the clause with respect to h and A is the set of adornments $\{A_{g_1}, \ldots, A_{g_k}\}$ such that A_{g_i} is an adornment for g_i . The adornment for the atoms in the body of the clause is constructed as follows: the jth argument of g_i is adorned by b in A_{g_i} if:

1. the jth argument of g_i occurs in the head of the clause as an argument adorned by b in A, or

2. jth argument of g_i is a variable bound in d, or

3. jth argument of g_i occurs as a variable in a subgoal preceding g_i.

Otherwise the jth argument of g_i is adorned by f. A magic atom for an atom h and an adornment A is the atom m_h that has only those arguments of h adorned by b in A, i.e., the arity of m_h is less or equal to the arity of h depending on how many arguments of h are adorned by b. Arguments adorned by f are removed.

The magic-set transformation is defined on adorned programs, and is guided by a sideways information strategy. A sideways information strategy (sips) is a decision on how to pass information sideways in the body of the rule. A sips determines how bindings in the head will be used, the order of evaluation for the subgoals in the body (join order), and how bindings will be passed between predicates in the body. A formal definition of sips was given by Beeri and Ramakrishnan [4].

The idea behind the magic-set evaluation is to compute an auxiliary predicate for each predicate in the original program called magic predicate. The magic predicate stores all the bindings for the associated predicate that would be generated by the top-down evaluation of the program. The program is rewritten using the magic predicates so that irrelevant tuples are not generated during the bottom-up evaluation of the program.

Definition 2.25 Given an adorned program AP, an adorned query goal q^{α}, and a full sips for each rule for AP, the magic sets transformation of AP, called MP is derived as follows:

1. Create a magic predicate m_p for each derived predicate p in AP. The arity of m_p is the number of bound arguments of p.

2. For each rule r in AP, add a modified version of r to MP. If rule r has head $p(\overline{t})$, where $\overline{t}$ is a shorthand for all arguments of p, the modified version of r is obtained by adding $m\_p(\overline{t^b})$ to the body of r, where $\overline{t^b}$ represents all the bound arguments of $p(\overline{t})$.

3. For each rule r in AP with head $p(\overline{t})$, and for each subgoal $q_i(\overline{t_i})$ where q_i is a derived predicate, add a magic rule to MP, where the head is $m\_q_i(\overline{t_i^b})$ and the body contains all the subgoals that precede q_i in the sips order associated with r, as well as the literal $m\_p(\overline{t^b})$.

4. Create a seed fact $m_q(\overline{c})$, where $\overline{c}$ is the set of constants in the bound arguments of the query goal.

Example 2.26 Consider the transitive closure program TC:
$P1$: $\mathsf{TranClos}(nf, nt) \leftarrow \mathsf{Transition}(nf, nt)$
$P2$: $\mathsf{TranClos}(nf, nt) \leftarrow \mathsf{Transition}(nf, nk),\ \mathsf{TranClos}(nk, nt)$
and the query:
$Q : \leftarrow \mathsf{TranClos}(1, nk)$
Program TC and the query Q are adorned as:
$AP1$: $\mathsf{TranClos}^{bf}(nf, nt) \leftarrow \mathsf{Transition}_1(nf, nt)$
$AP2$: $\mathsf{TranClos}^{bf}(nf, nt) \leftarrow \mathsf{Transition}_2(nf, nk),\ \mathsf{TranClos}^{bf}(nk, nt)$
$AQ : \leftarrow \mathsf{TranClos}^{bf}(1, nk)$
Two occurrences of Transition in rules $AP1$ and $AP2$ are marked by subscripts. The magic transformation of TC is derived as follows:

1. Create the magic predicate $\mathsf{m\_TranClos}^{bf}$ for the derived predicate $\mathsf{TranClos}^{bf}$.

2. Modify rules $AP1$ and $AP2$:
   $MP1$: $\mathsf{TranClos}^{bf}(nf, nt) \leftarrow \mathsf{m\_TranClos}^{bf}(nf),\ \mathsf{Transition}_1(nf, nt)$
   $MP2$: $\mathsf{TranClos}^{bf}(nf, nt) \leftarrow \mathsf{m\_TranClos}^{bf}(nf),\ \mathsf{Transition}_2(nf, nk),$
   $\mathsf{TranClos}^{bf}(nk, nt)$

3. From rule $AP2$ and the subgoal $\mathsf{TranClos}^{bf}(nk, nt)$, derive the magic rule:
   $MP3$: $\mathsf{m\_TranClos}^{bf}(nk) \leftarrow \mathsf{m\_TranClos}^{bf}(nf),\ \mathsf{Transition}_2(nf, nk)$

4. add the seed fact:
   $MP4$: $\mathsf{m\_TranClos}^{bf}(1)$

Extended Magic-set Evaluation

Some implementations of Datalog$^{\text{cv}}$ such as Relationlog [56] use magic set strategies extended to be used with nested levels. An extended adornment for an atom h with arity k is a string of length k on the alphabet $\{b, f, C, P, T\}$, where b is for atomic argument and stands for bound, f is for atomic argument and stands for free, C is for complete set term (terms with the set constructor $\{\}$), P is for partial set term (terms with the grouping constructor $\langle\rangle$), T is for tuple term. Furthermore, for a complete set term, partial set term or a tuple term, another string on the alphabet $\{b, f, C, P, T\}$ is used as a superscript to represent the status of arguments in it.

Example 2.27 For an argument $(a, \{x\})$ where a is a constant and x is an atomic variable the adornment is bC^f.

Top-down (SLD) Resolution with Memoing: SLG

Top-down approaches naturally focus attention on relevant facts. Hence, they avoid, to the extent possible, the production of states that are not needed to be searched. The basic top-down evaluation procedure is SLD resolution (Linear resolution with Selection function for Definite programs) [93] which views a program clause as a procedure declaration, and each literal in the body of the clause as a procedural call. The most serious drawback of this computational mechanism is that, it is not guaranteed to terminate for logic-programming oriented recursive languages. In addition to this, SLD has a tendency to rederive the same fact. An alternative way is a top-down evaluation with memoing strategy called SLG resolution (Linear resolution with Selection function for General logic programs) [80, 17, 16] which extends SLD resolution by adding tabling to make evaluations finite and non-redundant, and by adding a scheduling strategy to treat negation efficiently. The idea behind memoing is to maintain a table of procedure calls and the values to return during execution. If the same call is made later in the execution, use the saved answer to update the current state. There are efficient scheduling strategies implemented for tabled logic programs:

- Batched Scheduling: provides space and time reduction over the naive strategy which is called single stack scheduling.

- Local Scheduling: provides speedups for programs that require answer subsumption.

Example 2.28 Consider the fact base:
$\mathsf{Transition}(1,2)$
$\mathsf{Transition}(2,1)$
$\mathsf{Transition}(2,3)$
and the transitive closure program TC:
$P1 : \mathsf{TranClos}(nf, nt) \leftarrow \mathsf{Transition}(nf, nt)$
$P2 : \mathsf{TranClos}(nf, nt) \leftarrow \mathsf{Transition}(nf, nk),\ \mathsf{TranClos}(nk, nt)$
for query:
$Q : \leftarrow \mathsf{TranClos}(1,3)$
Figure 2.3 shows an infinite sequence of SLD resolution steps for this program and the given query. Each row in Figure 2.3 is a list of atoms, the first is the query. A pseudoatom $ans[]$ is added to the end of the query to collect the answer. Each row follows from the previous row by matching the first atom with the head of a rule (or a fact), replacing the atom by the body of the matching rule, and applying the match to all the atoms. The rows are divided into columns to emphasize the procedure-calling nature of the computation. Each row represents a state of the run time stack growing to the left and each column

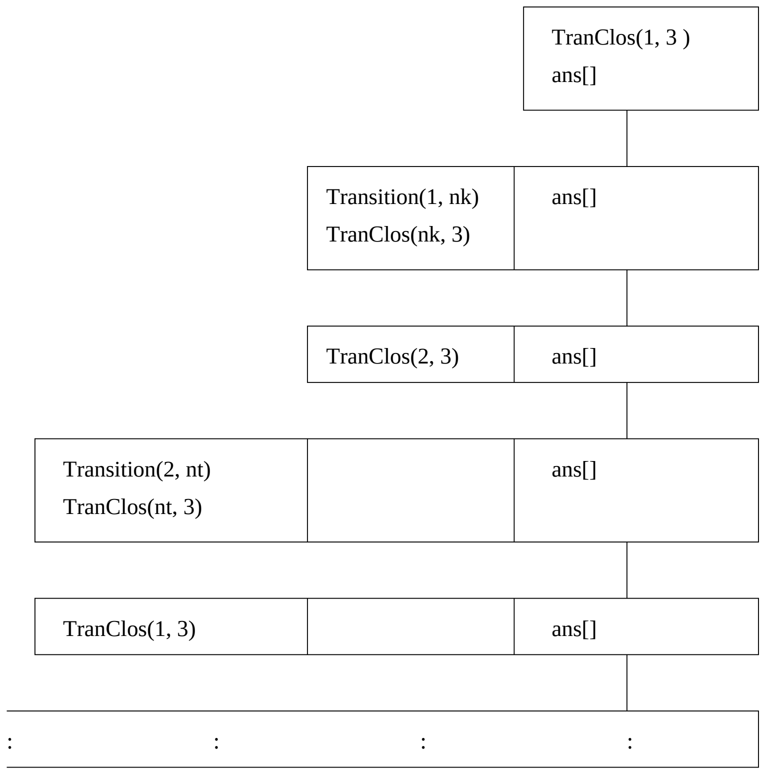


Figure 2.3: Infinite SLD resolution path for Q

represents a stack frame when viewing the computation as as execution of a procedural program. The contents of a frame are the calls of the subprocedures that remain to be made in that level. Consider the resolution steps in Figure 2.3, the last row shown has the identical list of atoms as the first, so this cycle can be repeated forever.

The procedural control in SLD resolution handles parameter passing by matching an atom with the head of a rule and applying the match to every atom in the entire stack. There are several other extensions of the SLD resolution. Example 2.29 shows an incremental version which handles passing of parameters into and out of procedures explicitly called OLD[3] refutation. The reason for this representation is to make the procedure calls and returns explicit so that they can be used for memoing. Example 2.30 shows OLDT strategy which adds memoing to OLD refutation.

Example 2.29 Consider the fact base:

[3]The letters of the name come from *Ordered selection strategy with Linear resolution for Definite clauses*

$\mathsf{Transition}(1,2)$
$\mathsf{Transition}(3,2)$
$\mathsf{Transition}(2,4)$
and the transitive closure program TC:
$P1 : \mathsf{TranClos}(nf,nt) \leftarrow \mathsf{Transition}(nf,nt)$
$P2 : \mathsf{TranClos}(nf,nt) \leftarrow \mathsf{Transition}(nf,nk)$, $\mathsf{TranClos}(nk,nt)$
for query:
$Q : \leftarrow \mathsf{TranClos}(1,A)$
OLD refutation of TC on Q which handles passing of parameters into and out explicitly to be used in memoing is shown in Figure 2.4.

Example 2.30 Consider the fact base:
$\mathsf{Transition}(1,2)$
$\mathsf{Transition}(2,3)$
$\mathsf{Transition}(2,1)$
and the transitive closure program TC:
$P1 : \mathsf{TranClos}(nf,nt) \leftarrow \mathsf{Transition}(nf,nt)$
$P2 : \mathsf{TranClos}(nf,nt) \leftarrow \mathsf{Transition}(nf,nk)$, $\mathsf{TranClos}(nk,nt)$
for query:
$Q : \leftarrow \mathsf{TranClos}(1,A)$
SLD tree for Q is infinite. OLDT forest for the same query is shown in Figure 2.5 which is finite.

Each subgoal has a corresponding OLDT tree. A node in an OLDT tree is labeled by a definite clause. Head of the clause shows relevant variable bindings and body contains the subgoals to be solved. If the same subgoal occurs later, it is resolved using only the answers that have been computed or will be computed. The memoing technique both detects positive loops and avoids redundant computation of identical subgoals.

It can be shown that both of the techniques: Magic-set Evaluation and Top-down (SLD) Resolution with Memoing: SLG simulate each other and that the magic predicates match the memoing data structures (modulo open terms). For detailed description of the above techniques see [4, 64] and [16, 17, 80], respectively.

2.2.2 Deductive Database Systems supporting $\mathrm{Datalog}^{\mathrm{cv}}$

Deductive database systems are database management systems that use a logical model of data as a query language and storage structure. There are various deductive database implementations that support Datalog^{cv}. In this section we examine four different deductive database systems: LDL, Relationlog, CORAL, and XSB.

LDL (Logical Data Structure) [3, 18, 65, 90] is the first language that supports Datalog^{cv} with well defined semantics. LDL supports tuples indirectly by using functors and it supports sets directly. It allows the use of member predicate to access the elements in a set, and provides set enumeration and set grouping mechanisms for the construction of sets. LDL system uses magic sets technique in query evaluation. LDL has a language with a first-order syntax and higher-order semantics. There are also logics that have a higher-order syntax but a first-order semantics such as F-logic (Frame Logic) [46] and HiLog [15] which support sets.

Relationlog (Relation LOGic) [56] is another Datalog^{cv} language with powerful tuple and set constructors. The main novelty of the language is the use of partial and complete set terms for representing and manipulating both partial and complete information on nested sets, tuples and relations. They generalize the set enumeration and set grouping mechanisms of LDL and allow direct encoding of open and closed world assumptions on nested sets, tuples and relations. It is argued in [57] that the traditional semi-naive and magic set rule rewriting techniques cannot be used directly in processing. Hence, extended semi-naive and magic sets techniques are used for evaluating Relationlog programs. Extended semi-naive technique uses grouping and difference operators to carry out the evaluation on sets, while the magic set technique extends the use of adornments on predicates for set and tuple terms in order to represent the bound and free information in a nested level.

CORAL (COmbining Relations and Logic) [75, 76, 77] is a declarative language that supports definite clauses with negation, multiset generation and set grouping. General matching and unification of sets is not supported [76]. CORAL deductive database system uses bottom-up evaluation with magic rewriting.

The XSB system [74, 80] uses a language that is very similar to Prolog. XSB supports tuple constructors, however it does not support sets. Set enumeration and grouping operations can be implemented on XSB using list constructors. The query evaluation on XSB is based on SLG resolution [85] that combines SLD resolution with memoing. XSB applies SLD resolution for non-tabled predicates, and uses memoization for tabled predicates.

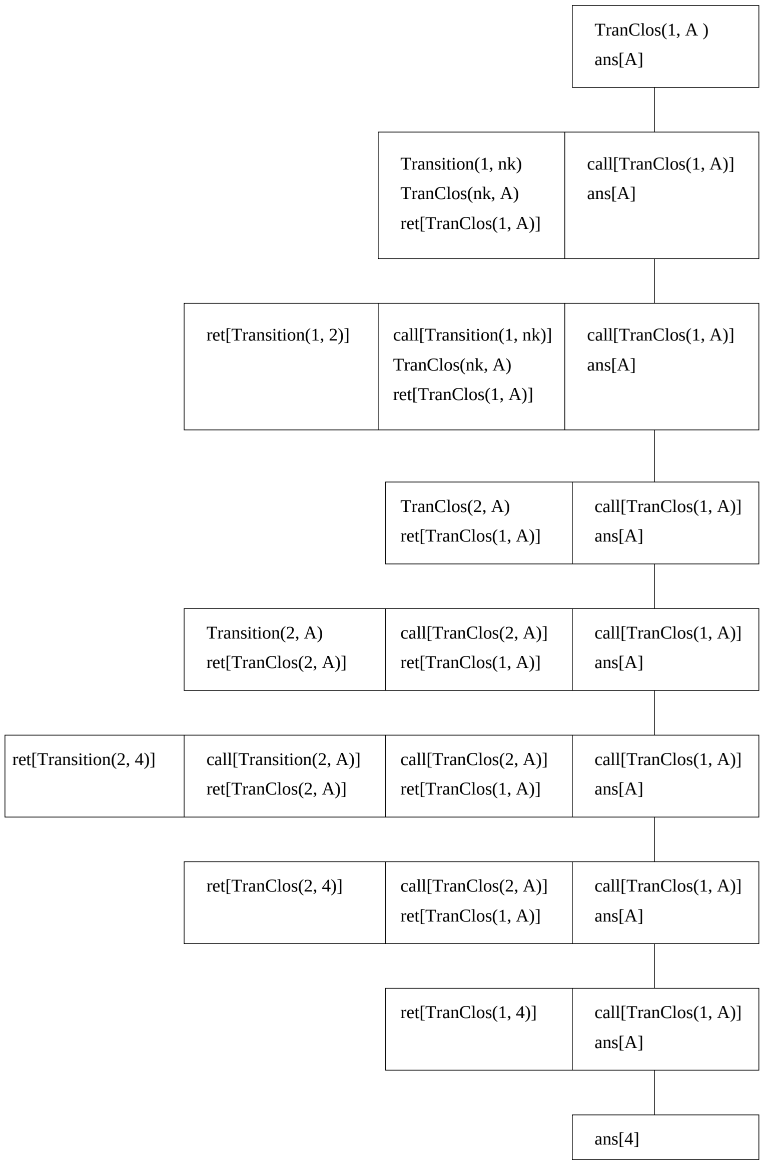


Figure 2.4: OLD refutation of TC on Q

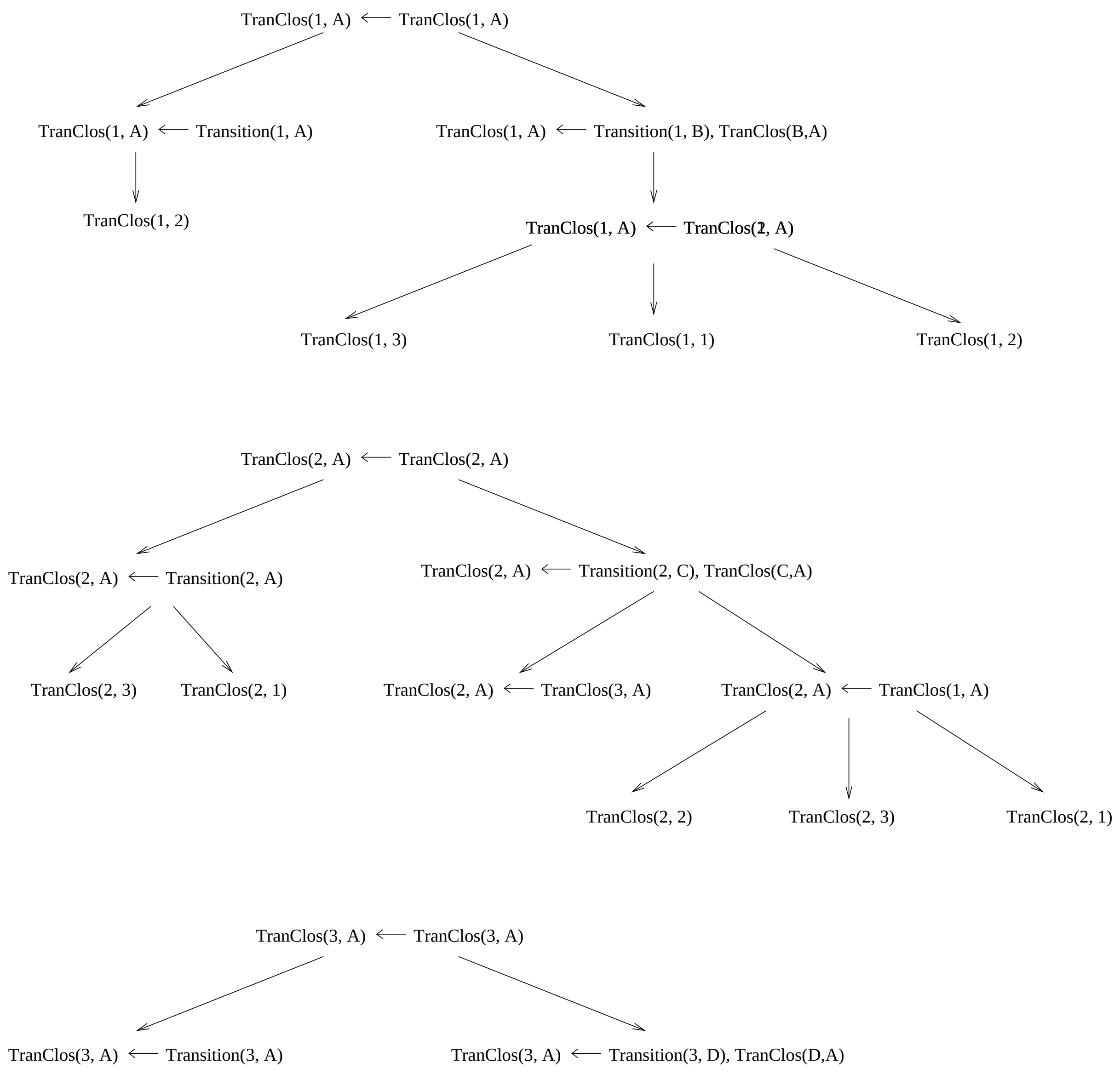

TranClos(1, A) ⟵ TranClos(1, A)
TranClos(1, A) ⟵ Transition(1, A)
TranClos(1, A) ⟵ Transition(1, B), TranClos(B,A)
TranClos(1, 2)
TranClos(1, A) ⟵ TranClos(2, A)
TranClos(1, 3)
TranClos(1, 1)
TranClos(1, 2)
TranClos(2, A) ⟵ TranClos(2, A)
TranClos(2, A) ⟵ Transition(2, A)
TranClos(2, A) ⟵ Transition(2, C), TranClos(C,A)
TranClos(2, 3)
TranClos(2, 1)
TranClos(2, A) ⟵ TranClos(3, A)
TranClos(2, A) ⟵ TranClos(1, A)
TranClos(2, 2)
TranClos(2, 3)
TranClos(2, 1)
TranClos(3, A) ⟵ TranClos(3, A)
TranClos(3, A) ⟵ Transition(3, A)
TranClos(3, A) ⟵ Transition(3, D), TranClos(D,A)

Figure 2.5: OLDT forest for Q

Chapter 3

Logic Programming Approach to Decision Procedures for Weak Second-order Logics

Given a WS1S/WS2S formula φ we create a Datalog$^{\text{cv}}$ program P_φ such that an answer to a reachability/transitive closure goal w.r.t. this program proves satisfiability of φ.

However, we do not attempt to map the formula φ itself to Datalog$^{\text{cv}}$. Rather, we represent the construction of A_φ—the finite automaton that captures models of φ—as a Datalog$^{\text{cv}}$ program P_φ. This enables the use of the efficient evaluation techniques for Datalog$^{\text{cv}}$ discussed in Section 2.2.1.

3.1 Introduction

Tools based on the connection between logic and automata—in particular the MONA system [47]—have been developed and shown to be efficient enough for practical applications [40]. However, for reasoning in large theories consisting of relatively simple constraints, such as theories capturing UML class diagrams or database schemata, the MONA system runs into a serious state-space explosion problem—the size of the automaton capturing the (language of) models for a given formula quickly exceeded the space available in most computers. The problem can be traced to the *automata product* operation that is used to translate conjunction in the original formulas rather than to the projection/determinization operations needed to handle quantifier alternations.

This work introduces a technique that combats the problem. However, unlike most other approaches that usually attempt to use various compact representation techniques for automata, e.g., based on BDDs [8, 40, 47, 48] or on state space factoring using a *guided* automaton [6], our approach is based on techniques developed for query evaluation in deductive databases, in particular on the *Magic Set transformation* [3] and the *SLG resolution* [16, 17]. We also study the impact of using other optimization techniques developed for Logic Programs, such as goal reordering.

The main contribution of the work we present in this chapter is establishing the connection between the automata-based decision procedures for WS1S (and, analogously, for WS2S) and query evaluation in Complex-value Datalog (Datalog$^{\text{cv}}$). Indeed, the complexity of query evaluation in Datalog$^{\text{cv}}$ matches the complexity of the WS1S decision procedure and thus it seems to be an appropriate tool for this task. Our approach is based on representing automata using nested relations and on defining the necessary automata-theoretic operations using Datalog$^{\text{cv}}$ programs. This reduces to posing a closed Datalog$^{\text{cv}}$ query over a Datalog$^{\text{cv}}$ program representing implicitly the final automaton. This observation combined with the powerful query evaluation techniques developed for deductive databases, limit the explored state space to elements needed to show non-emptiness of the automaton and, in turn, satisfiability of the corresponding formula.

In addition to showing the connection between the automata-based decision procedures and query evaluation in Datalog$^{\text{cv}}$, we have also conducted experiments with the XSB [80] system that demonstrate the benefits of the proposed method over more standard approaches.

3.2 A Decision Procedure for WS1S

In this section we outline the decision procedure for WS1S. We first define a representation and give automata operations as Datalog$^{\text{cv}}$ views and then show our experimental results.

3.2.1 Representation of Automata

First, we fix the representation for automata $A_\varphi = (\Sigma_\varphi, Q_\varphi, S_\varphi, \delta_\varphi, F_\varphi)$ that capture models of a WS1S formula φ. Note that we omit the set of directions D_φ since it is fixed. Given a WS1S formula φ with free variables $x_1, \ldots, x_k$ we define a Datalog$^{\text{cv}}$ program P_φ that defines the following predicates:

1. $\mathsf{Node}_\varphi(n)$ representing the nodes of A_φ,
2. $\mathsf{Start}_\varphi(n)$ representing the set of starting states,
3. $\mathsf{Final}_\varphi(n)$ representing the set of final states, and

4. $\mathsf{Trans}_\varphi(nf_1, nt_1, \overline{x})$ representing the transition function δ_φ as a relation such that $(q, t, \sigma) \in \delta_\varphi$ if there is a transition in A_φ from node q to node t with letter σ.

where $\overline{x} = \{x_1, x_2, \ldots, x_k\}$ is the set of free variables of φ; concatenation of their binary valuations represents a letter of A_φ's alphabet.

Definition 3.1 P_φ represents A_φ iff the interpretation of (Node_φ, Start_φ, Trans_φ, Final_φ) in the minimal model of P_φ is isomorphic to $A_\varphi = (Q_\varphi,\ S_\varphi,\ \delta_\varphi,\ F_\varphi)$.

First, we define the automata for the atomic formulas.

Definition 3.2 The following program P_φ represents the automaton A_φ for $\varphi = x \subseteq y$ (shown in the left part of Figure 2.2):

$$\begin{array}{lll}
\mathsf{Node}_\varphi(n_0) \leftarrow & \mathsf{Trans}_\varphi(n_0, n_0, 0, 0) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_1, 0, 0) \leftarrow \\
\mathsf{Node}_\varphi(n_1) \leftarrow & \mathsf{Trans}_\varphi(n_0, n_0, 0, 1) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_1, 1, 0) \leftarrow \\
\mathsf{Start}_\varphi(n_0) \leftarrow & \mathsf{Trans}_\varphi(n_0, n_0, 1, 1) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_1, 0, 1) \leftarrow \\
\mathsf{Final}_\varphi(n_0) \leftarrow & \mathsf{Trans}_\varphi(n_0, n_1, 1, 0) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_1, 1, 1) \leftarrow
\end{array}$$

Definition 3.3 The following program P_φ represents the automaton A_φ for $\varphi = s(x, y)$:

$$\begin{array}{lll}
 & \mathsf{Trans}_\varphi(n_0, n_0, 0, 0) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_0, 1, 0) \leftarrow \\
\mathsf{Node}_\varphi(n_0) \leftarrow & \mathsf{Trans}_\varphi(n_0, n_1, 0, 1) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_1, 1, 1) \leftarrow \\
\mathsf{Node}_\varphi(n_1) \leftarrow & \mathsf{Trans}_\varphi(n_0, n_2, 1, 1) \leftarrow & \mathsf{Trans}_\varphi(n_2, n_2, 0, 0) \leftarrow \\
\mathsf{Node}_\varphi(n_2) \leftarrow & \mathsf{Trans}_\varphi(n_0, n_2, 1, 0) \leftarrow & \mathsf{Trans}_\varphi(n_2, n_2, 0, 1) \leftarrow \\
\mathsf{Start}_\varphi(n_0) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_2, 0, 0) \leftarrow & \mathsf{Trans}_\varphi(n_2, n_2, 1, 0) \leftarrow \\
\mathsf{Final}_\varphi(n_0) \leftarrow & \mathsf{Trans}_\varphi(n_1, n_2, 0, 1) \leftarrow & \mathsf{Trans}_\varphi(n_2, n_2, 1, 1) \leftarrow
\end{array}$$

Note that while for atomic formulas, the values representing nodes are atomic, for automata corresponding to complex formulas these values become complex.

3.2.2 Automata-theoretic Operations

We define the appropriate automata-theoretic operations: negation, conjunction, projection, and determinization used in decision procedures for the logics under consideration as programs in Datalog$^{\text{cv}}$ as follows.

Definition 3.4 The program $P_{\neg\alpha}$ consists of the following clauses added to the program P_α:

1. $\mathsf{Node}_{\neg\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n)$

2. $\mathsf{Start}_{\neg\alpha}(n) \leftarrow \mathsf{Start}_{\alpha}(n)$
3. $\mathsf{Final}_{\neg\alpha}(n) \leftarrow \mathsf{Node}_{\alpha}(n), \neg\mathsf{Final}_{\alpha}(n)$
4. $\mathsf{Trans}_{\neg\alpha}(nf_1, nt_1, \overline{x}) \leftarrow \mathsf{Trans}_{\alpha}(nf_1, nt_1, \overline{x})$

Definition 3.5 Let $A_\alpha = (\Sigma_\alpha, Q_\alpha, S_\alpha, \delta_\alpha, F_\alpha)$ be a deterministic finite automaton capturing the models of α. Then $A_{\neg\alpha} = (Q_{\neg\alpha}, \Sigma_{\neg\alpha}, S_{\neg\alpha}, \delta_{\neg\alpha}, F_{\neg\alpha})$ where $Q_{\neg\alpha} = Q_\alpha$, $\Sigma_{\neg\alpha} = \Sigma_\alpha$, $S_{\neg\alpha} = S_\alpha$, $\delta_{\neg\alpha} = \delta_\alpha$, and $F_{\neg\alpha} = Q_\alpha \backslash F_\alpha$.

The following lemma is immediate:

Lemma 3.6 *If P_α represents A_α then $P_{\neg\alpha}$ represents $A_{\neg\alpha}$.*

Proof: Using rules 1, 2, and 4, we can conclude that $Q_{\neg\alpha} = Q_\alpha$, $\Sigma_{\neg\alpha} = \Sigma_\alpha$, $S_{\neg\alpha} = S_\alpha$, $\delta_{\neg\alpha} = \delta_\alpha$, and from rule 3, $F_{\neg\alpha} = Q_\alpha \backslash F_\alpha$ since $\mathsf{Node}_{\neg\alpha}$ relation represents $Q_{\neg\alpha}$, $\mathsf{Start}_{\neg\alpha}$ represents $S_{\neg\alpha}$, $\mathsf{Final}_{\neg\alpha}$ represents $F_{\neg\alpha}$, and $\mathsf{Trans}_{\neg\alpha}$ represents $\delta_{\neg\alpha}$. Hence, $P_{\neg\alpha}$ represents $A_{\neg\alpha}$, which is the automaton representing $\neg\alpha$. □

The proof is straightforward since the complementation operation on a deterministic finite word automaton is achieved by assigning the nodes that are not final states in the original automaton as final states in the complementation automaton. The conjunction automaton which represents the conjunction of the two formulas that original automata represent is defined as follows.

Definition 3.7 The program $P_{\alpha_1\wedge\alpha_2}$ consists of the union of programs P_{α_1} and P_{α_2} and the following clauses

1. $\mathsf{Node}_{\alpha_1\wedge\alpha_2}([n_1, n_2]) \leftarrow \mathsf{Node}_{\alpha_1}(n_1), \mathsf{Node}_{\alpha_2}(n_2)$
2. $\mathsf{Start}_{\alpha_1\wedge\alpha_2}([n_1, n_2]) \leftarrow \mathsf{Start}_{\alpha_1}(n_1), \mathsf{Start}_{\alpha_2}(n_2)$
3. $\mathsf{Final}_{\alpha_1\wedge\alpha_2}([n_1, n_2]) \leftarrow \mathsf{Final}_{\alpha_1}(n_1), \mathsf{Final}_{\alpha_2}(n_2)$
4. $\mathsf{Trans}_{\alpha_1\wedge\alpha_2}([nf_1, nf_2], [nt_1, nt_2], \overline{x}, \overline{y}, \overline{z}) \leftarrow$
   $\mathsf{Trans}_{\alpha_1}(nf_1, nt_1, \overline{x}, \overline{y}),\ \mathsf{Trans}_{\alpha_2}(nf_2, nt_2, \overline{y}, \overline{z})$

The sets of variables $\overline{x}, \overline{y}$ represent the free variables of the formula A_{α_1} and $\overline{y}, \overline{z}$ of the formula A_{α_2}.

Definition 3.8 It can be shown that if $A_{\alpha_1} = (\Sigma_{\alpha_1}, Q_{\alpha_1}, S_{\alpha_1}, \delta_{\alpha_1}, F_{\alpha_1})$ and $A_{\alpha_2} = (\Sigma_{\alpha_2}, Q_{\alpha_2}, S_{\alpha_2}, \delta_{\alpha_2}, F_{\alpha_2})$ then $A_{\alpha_1\wedge\alpha_2} = (Q_{\alpha_1\wedge\alpha_2}, \Sigma_{\alpha_1\wedge\alpha_2}, S_{\alpha_1\wedge\alpha_2}, \delta_{\alpha_1\wedge\alpha_2}, F_{\alpha_1\wedge\alpha_2})$, where $Q_{\alpha_1\wedge\alpha_2} = Q_{\alpha_1} \times Q_{\alpha_2}$, $\Sigma_{\alpha_1\wedge\alpha_2} = \{\overline{xyz} \mid \overline{xy} \in \Sigma_{\alpha_1} \textit{ and } \overline{yz} \in \Sigma_{\alpha_2}\}$, $S_{\alpha_1\wedge\alpha_2} = S_{\alpha_1} \times S_{\alpha_2}$, $\delta_{\alpha_1\wedge\alpha_2} = \delta_{\alpha_1} \bowtie \delta_{\alpha_2}$ (natural join on δ_{α_1} and δ_{α_2}), and $F_{\alpha_1\wedge\alpha_2} = F_{\alpha_1} \times F_{\alpha_2}$.

Again, immediately from the definition we have:

Lemma 3.9 *Let P_{α_1} represent A_{α_1} and P_{α_2} represent A_{α_2}. Then $P_{\alpha_1 \wedge \alpha_2}$ represents $A_{\alpha_1 \wedge \alpha_2}$.*

Proof: Using rules 1, 2, 3, and 4 we can conclude that $Q_{\alpha_1 \wedge \alpha_2} = Q_{\alpha_1} \times Q_{\alpha_2}$, $S_{\alpha_1 \wedge \alpha_2} = S_{\alpha_1} \times S_{\alpha_2}$, $\delta_{\alpha_1 \wedge \alpha_2} = \delta_{\alpha_1} \bowtie \delta_{\alpha_2}$, and $F_{\alpha_1 \wedge \alpha_2} = F_{\alpha_1} \times F_{\alpha_2}$ since $\mathsf{Node}_{\alpha_1 \wedge \alpha_2}$ relation represents $Q_{\alpha_1 \wedge \alpha_2}$, $\mathsf{Start}_{\alpha_1 \wedge \alpha_2}$ represents $S_{\alpha_1 \wedge \alpha_2}$, $\mathsf{Final}_{\alpha_1 \wedge \alpha_2}$ represents $F_{\alpha_1 \wedge \alpha_2}$, and $\mathsf{Trans}_{\alpha_1 \wedge \alpha_2}$ represents $\delta_{\alpha_1 \wedge \alpha_2}$. □

The proof above is based on the intersection operation on finite automata, where the starting state, final state and nodes of a conjunction automaton $A_{\alpha_1 \wedge \alpha_2}$ are represented as pairs of the starting states, final states and nodes of the automata A_{α_1} and A_{α_2}, and the transitions among nodes are computed based on the transition relations and the intersection of the alphabets of of A_{α_1} and A_{α_2}. The projection automaton which represents the existential quantification of a given formula is defined as follows.

Definition 3.10 The program $P^u_{\exists \overline{x}:\alpha}$ is defined as the union of P_α with the clauses

1. $\mathsf{Node}^u_{\exists \overline{x}:\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n)$
2. $\mathsf{Start}^u_{\exists \overline{x}:\alpha}(n) \leftarrow \mathsf{Start}_\alpha(n)$
3. $\mathsf{Final}^u_{\exists \overline{x}:\alpha}(n) \leftarrow \mathsf{Final}_\alpha(n)$
   $\mathsf{Final}^u_{\exists \overline{x}:\alpha}(n_0) \leftarrow \mathsf{Trans}_\alpha(n_0, n_1, \overline{x}, \overline{o}), \mathsf{Final}^u_{\exists \overline{x}:\alpha}(n_1)$
4. $\mathsf{Trans}^u_{\exists \overline{x}:\alpha}(nf_1, nt_1, \overline{y}) \leftarrow \mathsf{Trans}_\alpha(nf_1, nt_1, \overline{x}, \overline{y})$

The sets of variables $\overline{y}$ and $\overline{x}$ represent the free variables of the formula α, and $\overline{o} = \{0, 0, \ldots, 0\}$ where $|\overline{o}| = |\overline{y}|$.

Definition 3.11 For an automaton $A_\alpha = (\Sigma_\alpha,\ Q_\alpha,\ S_\alpha,\ \delta_\alpha, F_\alpha)$ the automaton $A^u_{\exists \overline{x}:\alpha} = (Q^u_{\exists \overline{x}:\alpha},\ \Sigma^u_{\exists \overline{x}:\alpha},\ S^u_{\exists \overline{x}:\alpha},\ \delta^u_{\exists \overline{x}:\alpha},\ F^u_{\exists \overline{x}:\alpha})$, where $Q^u_{\exists \overline{x}:\alpha} = Q_\alpha$, $\Sigma^u_{\exists \overline{x}:\alpha} = \{\overline{y} \mid \overline{xy} \in \Sigma_\alpha\}$, $S^u_{\exists \overline{x}:\alpha} = S_\alpha$, $F^u_{\exists \overline{x}:\alpha} = F_\alpha \cup F^i$, where $F^i = \{n \in Q_\alpha \mid \exists \mu \in L^i : n \textit{ has a path to an } f \in F_\alpha \textit{ with } \mu\}$, and $L^i = \{\omega \in (\{0,1\}^k)^* \mid \textit{the } j-\textit{th track of } \omega \textit{ is of the form } 0^* \textit{ for } j \neq i\}$, where k is the number of free variables in α, and i is the track which corresponds to the interpretation of the quantified variable $\overline{x}$, and $(n_f, n_t, \overline{y}) \in \delta^u_{\exists \overline{x}:\alpha}$ iff $(n_f, n_t, \overline{x}, \overline{y}) \in \delta_\alpha$.

Lemma 3.12 *If P_α represents A_α then $P^u_{\exists \overline{x}:\alpha}$ represents $A^u_{\exists \overline{x}:\alpha}$ which is a nondeterministic automaton accepting the models of the formula $\exists \overline{x} : \alpha$.*

Proof:

Using rules 1, 2, and 3, we can conclude that $Q^u_{\exists \overline{x}:\alpha} = Q_\alpha$, $S^u_{\exists \overline{x}:\alpha} = S_\alpha$, $F^u_{\exists \overline{x}:\alpha} = F_\alpha \cup F^i$, from rule 4, $\Sigma^u_{\exists \overline{x}:\alpha} = \{\overline{y} \mid \overline{xy} \in \Sigma_\alpha\}$, and $(n_f, n_t, \overline{y}) \in \delta^u_{\exists \overline{x}:\alpha}$ iff $(n_f, n_t, \overline{x}, \overline{y}) \in \delta_\alpha$. □

The above proof is straightforward except the part about the final states. We need to update final states because there may be leading zeros at the end of a string representing a model that should still be accepted. If this is the case then there is a state s in the original automaton such that a final state can be reached from s on a string of the form given by the definition of L^i. Thus, we need to characterize such states s as final states.

The automaton obtained by the projection operation is nondeterministic. The following Datalog$^{\text{cv}}$ program produces the representation of a deterministic automaton which accepts the same language as the nondeterministic one.

Definition 3.13 The program $P_{\exists\overline{x}:\alpha}$ consists of the program $P^u_{\exists\overline{x}:\alpha}$ and the following clauses

1. $\mathsf{Node}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Start}_{\exists\overline{x}:\alpha}(N)$
   $\mathsf{Node}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Trans}_{\exists\overline{x}:\alpha}(N_1, N, \overline{x})$
2. $\mathsf{Start}_{\exists\overline{x}:\alpha}(\{n\}) \leftarrow \mathsf{Start}^u_{\exists\overline{x}:\alpha}(n)$
3. $\mathsf{Final}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Node}_{\exists\overline{x}:\alpha}(N), \mathsf{Final}^u_{\exists\overline{x}:\alpha}(n), n \in N$
4. $\mathsf{Trans}_{\exists\overline{x}:\alpha}(N_1, \langle n\rangle, \overline{x}) \leftarrow \mathsf{Node}_{\exists\overline{x}:\alpha}(N_1),\ \mathsf{Next}_{\exists\overline{x}:\alpha}(N_1, n, \overline{x})$
   $\mathsf{Next}_{\exists\overline{x}:\alpha}(N_1, n_2, \overline{x}) \leftarrow n_1 \in N_1,\ \mathsf{Trans}^u_{\exists\overline{x}:\alpha}(n_1, n_2, \overline{x})$

Definition 3.14 Let $A^u_{\exists\overline{x}:\alpha} = (\Sigma^u_{\exists\overline{x}:\alpha}, Q^u_{\exists\overline{x}:\alpha}, S^u_{\exists\overline{x}:\alpha}, \delta^u_{\exists\overline{x}:\alpha}, F^u_{\exists\overline{x}:\alpha})$ be a nondeterministic automaton. Then $A_{\exists\overline{x}:\alpha} = (Q_{\exists\overline{x}:\alpha}, \Sigma_{\exists\overline{x}:\alpha}, S_{\exists\overline{x}:\alpha}, \delta_{\exists\overline{x}:\alpha}, F_{\exists\overline{x}:\alpha})$, where $Q_{\exists\overline{x}:\alpha}$ is a subset of the power set of $Q^u_{\exists\overline{x}:\alpha}$ that contains only the nodes reachable from $S_{\exists\overline{x}:\alpha}$ by $\delta_{\exists\overline{x}:\alpha}$, $\Sigma_{\exists\overline{x}:\alpha}$ is equal to $\Sigma^u_{\exists\overline{x}:\alpha}$, $S_{\exists\overline{x}:\alpha}$ is a singleton set such that $S^u_{\exists\overline{x}:\alpha} \in S_{\exists\overline{x}:\alpha}$, $F_{\exists\overline{x}:\alpha}$ is the set of all states in $Q_{\exists\overline{x}:\alpha}$ containing at least one final state of $A^u_{\exists\overline{x}:\alpha}$, $\delta_{\exists\overline{x}:\alpha}$ is the transition function of $A_{\exists\overline{x}:\alpha}$ where $(Q_F, Q_T, \overline{x}) \in \delta_{\exists\overline{x}:\alpha}$ iff $(n_f, n_t, \overline{x}) \in \delta^u_{\exists\overline{x}:\alpha}$ for all $n_f \in Q_F$ and $n_t \in Q_T$.

Lemma 3.15 *If $P^u_{\exists\overline{x}:\alpha}$ represents $A^u_{\exists\overline{x}:\alpha}$ then $P_{\exists\overline{x}:\alpha}$ represents the deterministic automaton $A_{\exists\overline{x}:\alpha}$.*

Proof: Using rules 1, 2, and 3, we can conclude that $Q_{\exists\overline{x}:\alpha}$ is a subset of the power set of $Q^u_{\exists\overline{x}:\alpha}$ which contains only the nodes reachable from $S_{\exists\overline{x}:\alpha}$ by $\delta_{\exists\overline{x}:\alpha}$, $S_{\exists\overline{x}:\alpha}$ is a singleton set such that $S^u_{\exists\overline{x}:\alpha} \in S_{\exists\overline{x}:\alpha}$, $F_{\exists\overline{x}:\alpha}$ is the set of all states in $Q_{\exists\overline{x}:\alpha}$ containing at least one final state of $A^u_{\exists\overline{x}:\alpha}$, since relation $\mathsf{Node}_{\exists\overline{x}:\alpha}$ represents $Q_{\exists\overline{x}:\alpha}$, $\mathsf{Start}_{\exists\overline{x}:\alpha}$ represents $S_{\exists\overline{x}:\alpha}$, and $\mathsf{Final}_{\exists\overline{x}:\alpha}$ represents $F_{\exists\overline{x}:\alpha}$. Using rule 4, $\Sigma_{\exists\overline{x}:\alpha}$ is equal to $\Sigma^u_{\exists\overline{x}:\alpha}$, and $(Q_F, Q_T, \overline{x}) \in \delta_{\exists\overline{x}:\alpha}$ iff $(n_f, n_t, \overline{x}) \in \delta^u_{\exists\overline{x}:\alpha}$ for all $n_f \in Q_F$ and $n_t \in Q_T$, since $\mathsf{Trans}_{\exists\overline{x}:\alpha}$ represents $\delta_{\exists\overline{x}:\alpha}$. $\Box$

The determinization representation presented here is the Datalog$^{\text{cv}}$ version of the subset construction algorithm described below:

- create the starting node of the deterministic finite automaton (DFA) by constructing the set containing starting node of the nondeterministic finite automaton (NFA),
- for the new DFA node and for each possible input symbol, find the set of nodes reachable by one transition on the input and add those set of nodes as a single node to the DFA,
- each time we generate a new DFA node we apply the above step to it, and
- the final nodes of the DFA are those which contain any of the final nodes of the NFA.

Last, the test for emptiness of an automaton has to be defined: To find out whether the language accepted by A_α is non-empty and thus whether α is satisfiable, a *reachability (transitive closure)* query is used.

Definition 3.16 The following program TC_α computes the transitive closure of the transition function of A_α.

1. $\mathsf{TransClos}_\alpha(n, n) \leftarrow$
2. $\mathsf{TransClos}_\alpha(nf_1, nt_1) \leftarrow \mathsf{Trans}_\alpha(nf_1, nt_2, \overline{x}),\ \mathsf{TransClos}_\alpha(nt_2, nt_1)$

Note that the use of magic sets and/or SLG resolution automatically transforms the transitive closure query into a reachability query.

Theorem 3.17 *Let φ be a WS1S formula. Then φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x), \mathsf{Final}_\varphi(y), \mathsf{TransClos}_\varphi(x, y)$.*

<u>Proof:</u> We know that φ is satisfiable iff A_φ has a path from $s_\varphi \in S_\varphi$ to $f_\varphi \in F_\varphi$. Start_φ represents S_φ, and Final_φ represents F_φ. There is a path from $s_\varphi \in S_\varphi$ to $f_\varphi \in F_\varphi$ iff $x \in \mathsf{Start}_\varphi$, $y \in \mathsf{Final}_\varphi$, and $(x, y) \in \mathsf{TransClos}_\varphi$. Hence, φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x),\ \mathsf{Final}_\varphi(y),\ \mathsf{TransClos}_\varphi(x, y)$. □

Example 3.18 Suppose that we have a formula $\exists y : y \subseteq x$, let A_ϕ be the automaton for the subformula $\phi = y \subseteq x$, we can use the following logic program to construct the automaton $A_{\exists y:\phi}$:

$$\begin{array}{l}
\mathsf{Node}^u_{\exists y:\phi}(n) \leftarrow \mathsf{Node}_\phi(n) \\
\mathsf{Start}^u_{\exists y:\phi}(n) \leftarrow \mathsf{Start}_\phi(n) \\
\mathsf{Final}^u_{\exists y:\phi}(n) \leftarrow \mathsf{Final}_\phi(n) \\
\mathsf{Final}^u_{\exists y:\phi}(n_0) \leftarrow \mathsf{Trans}_\phi(n_0, n_1, 0, y), \mathsf{Final}^u_{\exists y:\phi}(n_1) \\
\mathsf{Trans}^u_{\exists y:\phi}(n_1, n_2, x) \leftarrow \mathsf{Trans}_\phi(n_1, n_2, x, y)
\end{array}$$

This part computes the nondeterministic automaton ($A^u_{\exists y:\phi}$) representing the formula (see Definition 3.10).

$$
\begin{array}{l}
\mathsf{Node}_{\exists y:\phi}(N) \leftarrow \mathsf{Start}_{\exists y:\phi}(N) \\
\mathsf{Node}_{\exists y:\phi}(N) \leftarrow \mathsf{Node}_{\exists y:\phi}(N_1),\ \mathsf{Trans}_{\exists y:\phi}(N_1, N, x) \\
\mathsf{Start}_{\exists y:\phi}(\{n\}) \leftarrow \mathsf{Start}^u_{\exists y:\phi}(n) \\
\mathsf{Final}_{\exists y:\phi}(N) \leftarrow \mathsf{Node}_{\exists y:\phi}(N), \mathsf{Final}^u_{\exists y:\phi}(n), n \in N \\
\mathsf{Trans}_{\exists y:\phi}(N_1, \langle n \rangle, x) \leftarrow \mathsf{Node}_{\exists y:\phi}(N_1), \mathsf{Next}_{\exists y:\phi}(N_1, n, x) \\
\mathsf{Next}_{\exists y:\phi}(N_1, n_2, x) \leftarrow n_1 \in N_1, \mathsf{Trans}^u_{\exists y:\phi}(n_1, n_2, x) \\
\mathsf{TransClos}_{\exists y:\phi}(n, n) \leftarrow \\
\mathsf{TransClos}_{\exists y:\phi}(n_1, n_2) \leftarrow \mathsf{Trans}_{\exists y:\phi}(n_1, n_3, x), \mathsf{TransClos}_{\exists y:\phi}(n_3, n_2)
\end{array}
$$

This part computes the deterministic automaton ($A_{\exists y:\phi}$) representing the formula (see Definition 3.13), and the transitive closure of its transition relation (see Definition 3.16). Note that determinization is not needed unless there is a negation operation after this step. The satisfiability query is:

$$\leftarrow \mathsf{Start}_{\exists y:\phi}(n), \mathsf{Final}_{\exists y:\phi}(m), \mathsf{TransClos}_{\exists y:\phi}(n, m).$$

Example 3.19 Consider the formula $\phi = \neg(x \in V) \wedge (\exists W : (y \in W))$, the automaton accepting the models of this formula is shown in Figure 3.1 and its transition relation in clausal form is given below. In both representations the order of the free variables represented by the alphabet is x, y, V.

$$
\begin{array}{ll}
\mathsf{Trans}_\phi(1,1,0,0,X) \leftarrow & \\
\mathsf{Trans}_\phi(1,2,0,1,X) \leftarrow & \mathsf{Trans}_\phi(3,3,X,0,X) \leftarrow \\
\mathsf{Trans}_\phi(1,3,1,0,0) \leftarrow & \mathsf{Trans}_\phi(3,5,X,1,X) \leftarrow \\
\mathsf{Trans}_\phi(1,4,1,0,1) \leftarrow & \mathsf{Trans}_\phi(4,4,X,0,X) \leftarrow \\
\mathsf{Trans}_\phi(1,5,1,1,0) \leftarrow & \mathsf{Trans}_\phi(4,6,X,1,X) \leftarrow \\
\mathsf{Trans}_\phi(1,6,1,1,1) \leftarrow & \mathsf{Trans}_\phi(5,5,X,X,X) \leftarrow \\
\mathsf{Trans}_\phi(2,2,0,X,X) \leftarrow & \mathsf{Trans}_\phi(6,6,X,X,X) \leftarrow \\
\mathsf{Trans}_\phi(2,5,1,X,0) \leftarrow & \\
\mathsf{Trans}_\phi(2,6,1,X,1) \leftarrow &
\end{array}
$$

This is a compact representation of the transition relation where X stands for 0 or 1.

The use of SLG resolution to evaluate the transitive closure goal allows us to construct only the relevant parts of the automaton in a goal-driven way:

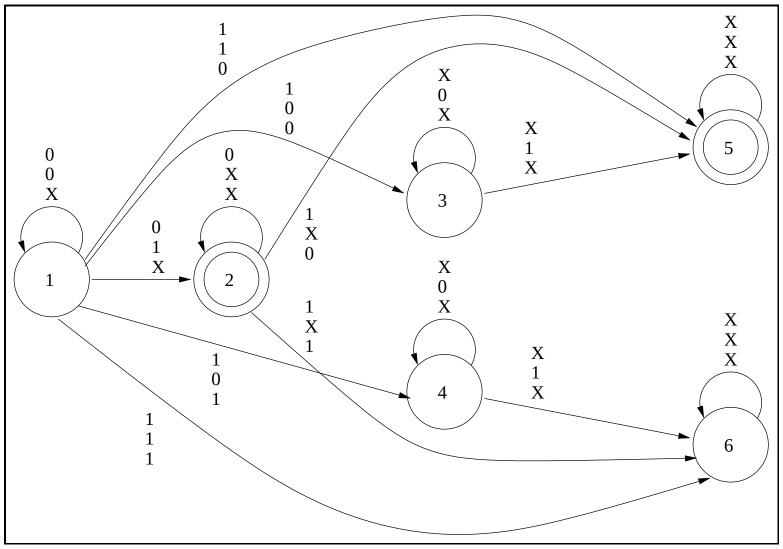


Figure 3.1: Automata representing the formula ϕ.

Example 3.20 For the formula $\phi = \neg(x \in v) \vee \neg(\exists w : (y \in w) \wedge (z \in w))$ the bottom-up evaluation creates 240 transitions, and 16 transitive closure tuples for the starting node while the top-down evaluation with memoing technique creates only 1 transition, and 1 tuple in the transitive closure for the starting node as shown in Figure 3.2. Let $A_{x \in v}$ be the automaton for the subformula $x \in v$, $A_{y \in w}$ be the automaton for the subformula $y \in w$, and $A_{z \in w}$ be the automaton for the subformula $z \in w$, we can use the following logic program to construct the automaton A_ϕ:

$$\begin{aligned}
&\mathsf{Node}_{(y\in w)\wedge(z\in w)}([n_1,n_2]) \leftarrow \mathsf{Node}_{y\in w}(n_1), \mathsf{Node}_{z\in w}(n_2)\\
&\mathsf{Start}_{(y\in w)\wedge(z\in w)}([n_1,n_2]) \leftarrow \mathsf{Start}_{y\in w}(n_1), \mathsf{Start}_{z\in w}(n_2)\\
&\mathsf{Final}_{(y\in w)\wedge(z\in w)}([n_1,n_2]) \leftarrow \mathsf{Final}_{y\in w}(n_1), \mathsf{Final}_{z\in w}(n_2)\\
&\mathsf{Trans}_{(y\in w)\wedge(z\in w)}([nf_1,nf_2],[nt_1,nt_2],y,z,w) \leftarrow \mathsf{Trans}_{y\in w}(nf_1,nt_1,y,w),\\
&\qquad\qquad \mathsf{Trans}_{z\in w}(nf_2,nt_2,z,w)
\end{aligned}$$

This part computes the conjunction automaton ($A_{(y\in w)\wedge(z\in w)}$) representing the subformula $(y \in w) \wedge (z \in w)$.

$$\begin{aligned}
&\mathsf{Node}^u_{\exists w:(y\in w)\wedge(z\in w)}(n) \leftarrow \mathsf{Node}_{(y\in w)\wedge(z\in w)}(n)\\
&\mathsf{Start}^u_{\exists w:(y\in w)\wedge(z\in w)}(n) \leftarrow \mathsf{Start}_{(y\in w)\wedge(z\in w)}(n)\\
&\mathsf{Final}^u_{\exists w:(y\in w)\wedge(z\in w)}(n) \leftarrow \mathsf{Final}_{(y\in w)\wedge(z\in w)}(n)\\
&\mathsf{Final}^u_{\exists w:(y\in w)\wedge(z\in w)}(n_0) \leftarrow \mathsf{Trans}_{(y\in w)\wedge(z\in w)}(n_0,n_1,0,y), \mathsf{Final}^u_{\exists w:(y\in w)\wedge(z\in w)}(n_1)\\
&\mathsf{Trans}^u_{\exists w:(y\in w)\wedge(z\in w)}(n_1,n_2,y,z) \leftarrow \mathsf{Trans}_{(y\in w)\wedge(z\in w)}(n_1,n_2,y,z,w)
\end{aligned}$$

This part computes the projection automaton ($A^u_{\exists w:(y\in w)\wedge(z\in w)}$) representing the subformula $\exists w : (y \in w) \wedge (z \in w)$.

$\mathsf{Node}_{\exists w:(y\in w)\wedge(z\in w)}(N) \leftarrow \mathsf{Start}_{\exists w:(y\in w)\wedge(z\in w)}(N)$
$\mathsf{Node}_{\exists w:(y\in w)\wedge(z\in w)}(N) \leftarrow \mathsf{Node}_{\exists w:(y\in w)\wedge(z\in w)}(N_1),\ \mathsf{Trans}_{\exists w:(y\in w)\wedge(z\in w)}(N_1, N, x)$
$\mathsf{Start}_{\exists w:(y\in w)\wedge(z\in w)}(\{n\}) \leftarrow \mathsf{Start}^u_{\exists w:(y\in w)\wedge(z\in w)}(n)$
$\mathsf{Final}_{\exists w:(y\in w)\wedge(z\in w)}(N) \leftarrow \mathsf{Node}_{\exists w:(y\in w)\wedge(z\in w)}(N), \mathsf{Final}^u_{\exists w:(y\in w)\wedge(z\in w)}(n), n \in N$
$\mathsf{Trans}_{\exists w:(y\in w)\wedge(z\in w)}(N_1, \langle n\rangle, y, z) \leftarrow \mathsf{Node}_{\exists w:(y\in w)\wedge(z\in w)}(N_1),$
$\quad\quad \mathsf{Next}_{\exists w:(y\in w)\wedge(z\in w)}(N_1, n, y, z)$
$\mathsf{Next}_{\exists w:(y\in w)\wedge(z\in w)}(N_1, n_2, y, z) \leftarrow n_1 \in N_1, \mathsf{Trans}^u_{\exists w:(y\in w)\wedge(z\in w)}(n_1, n_2, y, z)$

This part computes the deterministic automaton ($A_{\exists w:(y\in w)\wedge(z\in w)}$) representing the subformula $\exists w : (y \in w) \wedge (z \in w)$.

$\mathsf{Node}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}([n_1, n_2]) \leftarrow \mathsf{Node}_{x\in v}(n_1), \mathsf{Node}_{\exists w:(y\in w)\wedge(z\in w)}(n_2)$
$\mathsf{Start}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}([n_1, n_2]) \leftarrow \mathsf{Start}_{x\in v}(n_1), \mathsf{Start}_{\exists w:(y\in w)\wedge(z\in w)}(n_2)$
$\mathsf{Final}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}([n_1, n_2]) \leftarrow \mathsf{Final}_{x\in v}(n_1), \mathsf{Final}_{\exists w:(y\in w)\wedge(z\in w)}(n_2)$
$\mathsf{Trans}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}([nf_1, nf_2], [nt_1, nt_2], x, v, y, z) \leftarrow \mathsf{Trans}_{x\in v}(nf_1, nt_1, x, v),$
$\quad\quad \mathsf{Trans}_{\exists w:(y\in w)\wedge(z\in w)}(nf_2, nt_2, y, z)$

This part computes the conjunction automaton ($A_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}$) representing the subformula $(x \in v) \wedge (\exists w : (y \in w) \wedge (z \in w))$.

$\mathsf{Node}_\phi(n) \leftarrow \mathsf{Node}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}(n)$
$\mathsf{Start}_\phi(n) \leftarrow \mathsf{Start}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}(n)$
$\mathsf{Final}_\phi(n) \leftarrow \mathsf{Node}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}(n), \neg\mathsf{Final}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}(n)$
$\mathsf{Trans}_\phi(nf_1, nt_1, x, v, y, z) \leftarrow \mathsf{Trans}_{(x\in v)\wedge(\exists w:(y\in w)\wedge(z\in w))}(nf_1, nt_1, x, v, y, z)$

This part computes the complementation automaton (A_ϕ) representing the formula ϕ.

$\mathsf{TransClos}_\phi(n, n) \leftarrow$
$\mathsf{TransClos}_\phi(n_1, n_2) \leftarrow \mathsf{Trans}_\phi(n_1, n_3, x, v, y, z), \mathsf{TransClos}_\phi(n_3, n_2)$

This part computes the transitive closure of the transition relation of the automaton A_ϕ. The satisfiability query is:

$\leftarrow \mathsf{Start}_\phi(n), \mathsf{Final}_\phi(m), \mathsf{TransClos}_\phi(n, m).$

$? - \mathsf{Start}_\phi(n), \mathsf{Final}_\phi(m), \mathsf{TransClos}_\phi(n, m).$

(0) $\mathit{Call} : \mathsf{Start}_\phi(n)?$

$\ldots$

$\ldots$

(0) $\mathit{Exit} : \mathsf{Start}_\phi([1, \{[1, 1]\}])?$

(8) $\mathit{Call} : \mathsf{Final}_\phi(m)?$

$\ldots$

$\ldots$

(8) $\mathit{Exit} : \mathsf{Final}_\phi([1, \{[1, 1]\}])?$

(18) $\mathit{Call} : \mathsf{TransClos}_\phi([1, \{[1, 1]\}], [1, \{[1, 1]\}])?$

(19) $\mathit{Call} : \mathsf{Trans}_\phi([1, \{[1, 1]\}], [1, \{[1, 1]\}], x, y, z, v)?$

$\ldots$

$\ldots$

(19) $\mathit{Exit} : \mathsf{Trans}_\phi([1, \{[1, 1]\}], [1, \{[1, 1]\}], 0, 0, 0, 0)?$

(18) $\mathit{Exit} : \mathsf{TransClos}_\phi([1, \{[1, 1]\}], [1, \{[1, 1]\}])?$

$n = [1, \{[1, 1]\}]$

$m = [1, \{[1, 1]\}]$

Figure 3.2: Top-down evaluation of the program in Example 3.20.

The top-down evaluation of this query shown in Figure 3.2 first calls the rule(s) for the conjunct $\mathsf{Start}_\phi(n)$, an answer tuple $[1, \{[1, 1]\}]$ is returned. Refering to the program P_ϕ constructing A_ϕ we see that 1 is an answer tuple for $\mathsf{Start}_{x \in v}(n_1)$, and $\{[1, 1]\}$ is an answer tuple for $\mathsf{Start}_{\exists w:(y \in w) \wedge (z \in w)}(n_2)$. Note that the set value is a result of the subset constrution representation. Further, $[1, 1]$ is an answer tuple for $\mathsf{Start}_{(y \in w) \wedge (z \in w)}([n_1, n_2])$. The rest of the evaluation is completed similarly producing the answer tuples $[1, \{[1, 1]\}]$ and $[1, \{[1, 1]\}]$ for n and m.

3.2.3 Experimental Evaluation

We compare the performance of the technique proposed in this chapter and implemented using the XSB system[1] with the MONA system [40, 47], one of the most advanced tools for reasoning in weak second-order logics (WS1S and WS2S).

[1]XSB supports set operations on lists, hence we simulate the set values in Datalog^{cv} by lists in XSB.

In the experiments we present thoughout this thesis, we used a machine with 1.80 GHz Intel(R) Pentium 4 processor and 512 RAM. The performance results for a set of formulas are given in Figures 3.3, 3.4. We present a sample set of size 10 in each case from the set of formulas we used in the experiments (which is much larger than 10) where #i represents a particular formula. The response times are measured in seconds; N/A means "Not Answered" in 120 seconds which is the maximum waiting time we picked. The formulas are similar to the ones in Tableaux'98 (T98) satisfiability test suite except we varied their sizes, the number of existential quantifiers, and free variables.

The results show that XSB outperforms MONA for the formulas with many free variables since it performs large numbers of conjunction operations very efficiently with the use of top-down query evaluation and pruning techniques. This can be easily traced to the effects of goal-driven evaluation of P_φ which become more pronounced for large theories consisting of relatively simple formulas, such as those corresponding to constraints used in database schemata or UML diagrams. The experiments also compared different scheduling strategies of XSB namely the batched(XSB B) and the local(XSB L) ones. Batched scheduling performs better than local since our programs do not require answer subsumption. Experiments also show that tabling more predicates in addition to the auto-tabled ones (results in columns XSB B(T) and XSB L(T)) increases space requirements but enhances the performance substantially. The additional predicates we tabled are the Trans predicates in the programs that represent the determinization step. Since this step is critical in automaton construction, tabling the Trans predicate in addition to the Node predicate gives better results (see formulas 5, 9, 10 in Figures 3.3, 3.4).

On the other hand, MONA usually performs better on formulas that have less free variables and more quantifiers as it performs the projection operation faster than XSB. We believe that this is a practical problem caused by the implementation XSB uses for the evaluation of programs with nested relations and can be avoided using a more sophisticated implementation of Datalog^{cv}. In addition to this MONA uses a compact representation of automata based on BDDs [40, 47, 48] to enhance its performance, whereas XSB uses tries as the basis for tables combined with unification factoring [74, 22]. The size of the trie structures is, in general, larger than the size of a corresponding BDD. However it is easier to insert tuples to a trie than into a BDD.

In the preliminary experiments [92] we conducted we also used CORAL, a deductive system that supports Datalog^{cv} and Magic sets. Our results showed that CORAL also performs better than MONA for the same formulas as XSB, however XSB is faster than CORAL in all cases.

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
MONA	2.66	4.95	N/A	N/A	0.42	0.01	0.01	0.05	0.09	0.39
XSB B	0.01	0.01	0.11	0.01	35.72	1.74	N/A	0.01	6.02	94.64
XSB B(T)	0.01	0.01	0.01	0.01	15.88	0.18	N/A	0.01	0.29	10.96
XSB L	0.01	0.01	1.68	0.01	41.33	N/A	N/A	12.59	8.52	N/A
XSB L(T)	0.01	0.01	1.73	0.01	15.03	N/A	N/A	6.63	0.73	N/A

Figure 3.3: Performance (secs) w.r.t. increasing number of quantifiers

	#7	#6	#8	#10	#5	#9	#1	#2	#3	#4
MONA	0.01	0.01	0.05	0.39	0.42	0.09	2.66	4.95	N/A	N/A
XSB B	N/A	1.74	0.01	94.64	35.72	6.02	0.01	0.01	0.11	0.01
XSB B(T)	N/A	0.18	0.01	10.96	15.88	0.29	0.01	0.01	0.01	0.01
XSB L	N/A	N/A	12.59	N/A	41.33	8.52	0.01	0.01	1.68	0.01
XSB L(T)	N/A	N/A	6.63	N/A	15.03	0.73	0.01	0.01	1.73	0.01

Figure 3.4: Performance (secs) w.r.t. increasing number of variables

3.3 Decision Procedures for WS2S

In this section, we propose decision procedures for WS2S. We first outline a decision procedure based on bottom-up automata and present our experimental results where we compare our implementation with the MONA system [40] which also has a decision procedure for WS2S based on bottom-up automata, then we give a decision procedure based on top-down automata.

3.3.1 A Decision Procedure based on Bottom-up Automata

In this section, we outline the decision procedure for WS2S based on bottom-up automata. We first provide a representation which can be defined analogously to 3.1 and give automata operations as Datalog$^{\text{cv}}$ views and then show our experimental results.

Representation of Automata

Similarly to the WS1S case, we fix the representation for automata that capture models of WS2S formulas. Given a WS2S formula φ with free variables $x_1, \ldots, x_k$ we define a

Datalog$^{\text{cv}}$ program P_φ that defines the following predicates:

1. $\mathsf{Node}_\varphi(n)$ representing the nodes of A_φ,
2. $\mathsf{Start}_\varphi(n)$ representing the starting state,
3. $\mathsf{Final}_\varphi(n)$ representing the set of final states, and
4. $\mathsf{Trans}_\varphi(nf_1, nf_2, nt_1, \overline{x})$ representing the transition function δ_φ as a relation such that $(q_1, q_2, t, \sigma) \in \delta_\varphi$ if there is a transition in A_φ from nodes q_1 and q_2 to node t with letter σ.

where $\overline{x} = \{x_1, x_2, \ldots, x_k\}$ is the set of free variables of φ; concatenation of their binary valuations represents a letter of A_φ's alphabet.

The automata for the atomic formulas can be defined similarly to the automata for the atomic formulas in WS1S.

Automata-theoretic Operations

We define the appropriate automata-theoretic operations: negation, conjunction, projection, and determinization used in decision procedures based on bottom-up tree automata for WS2S as programs in Datalog$^{\text{cv}}$ as follows.

Definition 3.21 The program $P_{\neg\alpha}$ consists of the following clauses added to the program P_α:

1. $\mathsf{Node}_{\neg\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n)$
2. $\mathsf{Start}_{\neg\alpha}(n) \leftarrow \mathsf{Start}_\alpha(n)$
3. $\mathsf{Final}_{\neg\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n), \neg\mathsf{Final}_\alpha(n)$
4. $\mathsf{Trans}_{\neg\alpha}(nf_1, nf_2, nt_1, \overline{x}) \leftarrow \mathsf{Trans}_\alpha(nf_1, nf_2, nt_1, \overline{x})$

This definition is similar to definition 3.4 except for the last rule for the transition relation. The following lemma is immediate from the definition:

Lemma 3.22 *If P_α represents A_α then $P_{\neg\alpha}$ represents $A_{\neg\alpha}$.*

The proof is the same as the proof of lemma 3.6. The conjunction automaton which represents the conjunction of the two formulas that original automata represent is defined as follows.

Definition 3.23 The program $P_{\alpha_1\wedge\alpha_2}$ consists of the union of programs P_{α_1} and P_{α_2} and the following clauses

1. $\mathsf{Node}_{\alpha_1\wedge\alpha_2}([n_1,n_2]) \leftarrow \mathsf{Node}_{\alpha_1}(n_1), \mathsf{Node}_{\alpha_2}(n_2)$
2. $\mathsf{Start}_{\alpha_1\wedge\alpha_2}([n_1,n_2]) \leftarrow \mathsf{Start}_{\alpha_1}(n_1), \mathsf{Start}_{\alpha_2}(n_2)$
3. $\mathsf{Final}_{\alpha_1\wedge\alpha_2}([n_1,n_2]) \leftarrow \mathsf{Final}_{\alpha_1}(n_1), \mathsf{Final}_{\alpha_2}(n_2)$
4. $\mathsf{Trans}_{\alpha_1\wedge\alpha_2}([nf_{11},nf_{21}],[nf_{12},nf_{22}],[nt_{11},nt_{21}],\overline{x},\overline{y},\overline{z}) \leftarrow$
   $\mathsf{Trans}_{\alpha_1}(nf_{11},nf_{12},nt_{11},\overline{x},\overline{y}),\ \mathsf{Trans}_{\alpha_2}(nf_{21},nf_{22},nt_{21},\overline{y},\overline{z})$

The sets of variables $\overline{x},\overline{y}$ represent the free variables of the formula A_{α_1} and $\overline{y},\overline{z}$ of the formula A_{α_2}.

Again the definition is similar to definition 3.7 except for the last rule for the transition relation and from the definition we have:

Lemma 3.24 *Let P_{α_1} represent A_{α_1} and P_{α_2} represent A_{α_2}. Then $P_{\alpha_1\wedge\alpha_2}$ represents $A_{\alpha_1\wedge\alpha_2}$.*

The proof is the same as the proof of lemma 3.9. The projection automaton which represents the existential quantification of a given formula is defined as follows.

Definition 3.25 The program $P^u_{\exists\overline{x}:\alpha}$ is defined as the union of P_α with the clauses

1. $\mathsf{Node}^u_{\exists\overline{x}:\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n)$
2. $\mathsf{Start}^u_{\exists\overline{x}:\alpha}(n) \leftarrow \mathsf{Start}_\alpha(n)$
3. $\mathsf{Final}^u_{\exists\overline{x}:\alpha}(n) \leftarrow \mathsf{Final}_\alpha(n)$
   $\mathsf{Final}^u_{\exists\overline{x}:\alpha}(n_0) \leftarrow \mathsf{Trans}_\alpha(n_0,n_1,n_2,\overline{x},\overline{o}), \mathsf{Final}^u_{\exists\overline{x}:\alpha}(n_2)$
   $\mathsf{Final}^u_{\exists\overline{x}:\alpha}(n_1) \leftarrow \mathsf{Trans}_\alpha(n_0,n_1,n_2,\overline{x},\overline{o}), \mathsf{Final}^u_{\exists\overline{x}:\alpha}(n_2)$
4. $\mathsf{Trans}^u_{\exists\overline{x}:\alpha}(nf_1,nf_2,nt_1,\overline{y}) \leftarrow \mathsf{Trans}_\alpha(nf_1,nf_2,nt_1,\overline{x},\overline{y})$

The sets of variables $\overline{y}$ and $\overline{x}$ represent the free variables of the formula α, and $\overline{o} = \{0,0,\ldots,0\}$ where $|\overline{o}| = |\overline{y}|$.

Definition 3.26 For an automaton $A_\alpha = (\Sigma_\alpha,\ Q_\alpha,\ S_\alpha,\ \delta_\alpha, F_\alpha)$ the automaton $A^u_{\exists\overline{x}:\alpha} = (Q^u_{\exists\overline{x}:\alpha},\ \Sigma^u_{\exists\overline{x}:\alpha},\ S^u_{\exists\overline{x}:\alpha},\ \delta^u_{\exists\overline{x}:\alpha},\ F^u_{\exists\overline{x}:\alpha})$, where $Q^u_{\exists\overline{x}:\alpha} = Q_\alpha$, $\Sigma^u_{\exists\overline{x}:\alpha} = \{\overline{y} \mid \overline{xy} \in \Sigma_\alpha\}$, $S^u_{\exists\overline{x}:\alpha} = S_\alpha$, $F^u_{\exists\overline{x}:\alpha} = F_\alpha \cup F^i$, where $F^i = \{n_0 \in Q_\alpha \mid \exists\mu \in L^i$: there is a reduction from (n_0,n_1) where $n_1 \in Q_\alpha$, *to an* $f \in F_\alpha$ *with* $\mu\} \cup \{n_1 \in Q_\alpha \mid \exists\mu \in L^i$: there is a reduction from (n_0,n_1) where $n_0 \in Q_\alpha$, *to an* $f \in F_\alpha$ *with* $\mu\}$, and $L^i = \{\omega \in (\{0,1\}^k)^* \mid$ *the* $j-th$ *track of* ω *is of the form* 0^* *for* $j \neq i\}$, where k is the number of free variables in α, and i is the track which corresponds to the interpretation of the quantified variable $\overline{x}$, and $(n_{f_1},n_{f_2},n_{t_1},\overline{y}) \in \delta^u_{\exists\overline{x}:\alpha}$ iff $(n_{f_1},n_{f_2},n_{t_1},\overline{x},\overline{y}) \in \delta_\alpha$.

Lemma 3.27 *If P_α represents A_α then $P^u_{\exists\overline{x}:\alpha}$ represents $A^u_{\exists\overline{x}:\alpha}$ which is nondeterministic automaton for the formula $\exists\overline{x} : \alpha$.*

Proof: Using rules 1, 2, and 3, we can conclude that $Q^u_{\exists\overline{x}:\alpha} = Q_\alpha$, $S^u_{\exists\overline{x}:\alpha} = S_\alpha$, $F^u_{\exists\overline{x}:\alpha} = F_\alpha \cup F^i$, from rule 4, $\Sigma^u_{\exists\overline{x}:\alpha} = \{\overline{y} \mid \overline{xy} \in \Sigma_\alpha\}$, and $(n_{f_1}, n_{f_2}, n_{t_1}, \overline{y}) \in \delta^u_{\exists\overline{x}:\alpha}$ iff $(n_{f_1}, n_{f_2}, n_{t_1}, \overline{x}, \overline{y}) \in \delta_\alpha$. □

The following Datalog$^{\text{cv}}$ program produces the representation of a deterministic automaton which accepts the same language as the nondeterministic one obtained by the projection operation.

Definition 3.28 The program $P_{\exists\overline{x}:\alpha}$ consists of the program $P^u_{\exists\overline{x}:\alpha}$ and the following clauses

1. $\mathsf{Node}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Start}_{\exists\overline{x}:\alpha}(N)$
   $\mathsf{Node}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Trans}_{\exists\overline{x}:\alpha}(N_1, N_2, N, \overline{x})$
2. $\mathsf{Start}_{\exists\overline{x}:\alpha}(\{n\}) \leftarrow \mathsf{Start}^u_{\exists\overline{x}:\alpha}(n)$
3. $\mathsf{Final}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Node}_{\exists\overline{x}:\alpha}(N), \mathsf{Final}^u_{\exists\overline{x}:\alpha}(n), n \in N$
4. $\mathsf{Trans}_{\exists\overline{x}:\alpha}(N_1, N_2, \langle n\rangle, \overline{x}) \leftarrow \mathsf{Node}_{\exists\overline{x}:\alpha}(N_1),\ \mathsf{Node}_{\exists\overline{x}:\alpha}(N_2),$
   $\mathsf{Next}_{\exists\overline{x}:\alpha}(N_1, N_2, n, \overline{x})$
   $\mathsf{Next}_{\exists\overline{x}:\alpha}(N_1, N_2, n_3, \overline{x}) \leftarrow n_1 \in N_1,\ n_2 \in N_2,\ \mathsf{Trans}^u_{\exists\overline{x}:\alpha}(n_1, n_2, n_3, \overline{x})$

Definition 3.29 Let $A^u_{\exists\overline{x}:\alpha} = (\Sigma^u_{\exists\overline{x}:\alpha},\ Q^u_{\exists\overline{x}:\alpha},\ S^u_{\exists\overline{x}:\alpha},\ \delta^u_{\exists\overline{x}:\alpha},\ F^u_{\exists\overline{x}:\alpha})$ be a nondeterministic automaton. Then $A_{\exists\overline{x}:\alpha} = (Q_{\exists\overline{x}:\alpha},\ \Sigma_{\exists\overline{x}:\alpha},\ S_{\exists\overline{x}:\alpha}, \delta_{\exists\overline{x}:\alpha}, F_{\exists\overline{x}:\alpha})$, where $Q_{\exists\overline{x}:\alpha}$ is a subset of the power set of $Q^u_{\exists\overline{x}:\alpha}$ that contains only the nodes n_3 where there is a reduction from $(n_1 \in S_{\exists\overline{x}:\alpha}, n_2 \in S_{\exists\overline{x}:\alpha})$ to n_3 by $\delta_{\exists\overline{x}:\alpha}$, $\Sigma_{\exists\overline{x}:\alpha}$ is equal to $\Sigma^u_{\exists\overline{x}:\alpha}$, $S_{\exists\overline{x}:\alpha}$ is a singleton set such that $S^u_{\exists\overline{x}:\alpha} \in S_{\exists\overline{x}:\alpha}$, $F_{\exists\overline{x}:\alpha}$ is the set of all states in $Q_{\exists\overline{x}:\alpha}$ containing at least one final state of $A^u_{\exists\overline{x}:\alpha}$, $\delta_{\exists\overline{x}:\alpha}$ is the transition function of $A_{\exists\overline{x}:\alpha}$ where $(Q_{F_1}, Q_{F_2}, Q_T, \overline{x}) \in \delta_{\exists\overline{x}:\alpha}$ iff $(n_{f_1}, n_{f_2}, n_t, \overline{x}) \in \delta^u_{\exists\overline{x}:\alpha}$ for all $n_{f_1} \in Q_{F_1}$, $n_{f_2} \in Q_{F_2}$ and $n_t \in Q_T$.

Lemma 3.30 *If $P^u_{\exists\overline{x}:\alpha}$ represents $A^u_{\exists\overline{x}:\alpha}$ then $P_{\exists\overline{x}:\alpha}$ represents a deterministic automaton $A_{\exists\overline{x}:\alpha}$.*

Proof: Using rules 1, 2, and 3, we can conclude that $Q_{\exists\overline{x}:\alpha}$ is a subset of the power set of $Q^u_{\exists\overline{x}:\alpha}$ that contains only the nodes n_3 where there is a reduction from $(n_1 \in S_{\exists\overline{x}:\alpha}, n_2 \in S_{\exists\overline{x}:\alpha})$ to n_3 by $\delta_{\exists\overline{x}:\alpha}$, $S_{\exists\overline{x}:\alpha}$ is a singleton set such that $S^u_{\exists\overline{x}:\alpha} \in S_{\exists\overline{x}:\alpha}$, $F_{\exists\overline{x}:\alpha}$ is the set of all states in $Q_{\exists\overline{x}:\alpha}$ containing at least one final state of $A^u_{\exists\overline{x}:\alpha}$, since relation $\mathsf{Node}_{\exists\overline{x}:\alpha}$ represents $Q_{\exists\overline{x}:\alpha}$, $\mathsf{Start}_{\exists\overline{x}:\alpha}$ represents $S_{\exists\overline{x}:\alpha}$, and $\mathsf{Final}_{\exists\overline{x}:\alpha}$ represents $F_{\exists\overline{x}:\alpha}$. Using rule 4, $\Sigma_{\exists\overline{x}:\alpha}$ is equal to $\Sigma^u_{\exists\overline{x}:\alpha}$, and $(Q_{F_1}, Q_{F_2}, Q_T, \overline{x}) \in \delta_{\exists\overline{x}:\alpha}$ iff $(n_{f_1}, n_{f_2}, n_t, \overline{x}) \in \delta^u_{\exists\overline{x}:\alpha}$ for all $n_{f_1} \in Q_{F_1}$, $n_{f_2} \in Q_{F_2}$ and $n_t \in Q_T$, since $\mathsf{Trans}_{\exists\overline{x}:\alpha}$ represents $\delta_{\exists\overline{x}:\alpha}$. □

Last, we define the the test for emptiness of an automaton where we use a *reachability (transitive closure)* query.

Definition 3.31 The following program TC_α computes the transitive closure of the transition function of A_α.

1. $\mathsf{TransClos}_\alpha(n, n, n) \leftarrow$
2. $\mathsf{TransClos}_\alpha(nf_1, nf_2, nt_1) \leftarrow \mathsf{Trans}_\alpha(nf_1, nf_2, nt_2, \overline{x}),\ \mathsf{TransClos}_\alpha(nt_2, nf_3, nt_1)$
3. $\mathsf{TransClos}_\alpha(nf_1, nf_2, nt_1) \leftarrow \mathsf{Trans}_\alpha(nf_1, nf_2, nt_2, \overline{x}),\ \mathsf{TransClos}_\alpha(nf_3, nt_2, nt_1)$

The following theorem shows our correctness result for the decision procedure we propose for WS2S based on bottom-up tree automata.

Theorem 3.32 *Let φ be a WS2S formula such that the models of φ are computed by a bottom-up tree automaton. Then φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x)$, $\mathsf{Start}_\varphi(y)$, $\mathsf{Final}_\varphi(z)$, $\mathsf{TransClos}_\varphi(x, y, z)$.*

Proof: We know that φ is satisfiable iff there is a $f_\varphi \in F_\varphi$ that can be reached from $s_\varphi \in S_\varphi$, and $t_\varphi \in S_\varphi$. Start_φ represents S_φ, and Final_φ represents F_φ. There is a $f_\varphi \in F_\varphi$ that can be reached from $s_\varphi \in S_\varphi$, and $t_\varphi \in S_\varphi$ iff $x \in \mathsf{Start}_\varphi$, $y \in \mathsf{Start}_\varphi$, $z \in \mathsf{Final}_\varphi$, and $(x, y, z) \in \mathsf{TransClos}_\varphi$. Hence, φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x)$, $\mathsf{Start}_\varphi(y)$, $\mathsf{Final}_\varphi(z)$, $\mathsf{TransClos}_\varphi(x, y, z)$. □

Experimental Evaluation

We compare the performance of the technique proposed for WS2S in this chapter and implemented using the XSB system[2] with the MONA system [40, 47]. The performance results for a set of formulas are given in Figures 3.5, 3.6. We present a sample set of size 10 in each case from the set of formulas we used in the experiments where #i represents a particular formula. The response times are measured in seconds; N/A means "Not Answered" in 120 seconds.

The results show that XSB outperforms MONA for the formulas with many free variables, on the other hand, MONA usually performs better on formulas that have fewer free variables and more quantifiers and can be analyzed similarly to the results for WS1S.

[2]We simulate the set values in Datalog$^{\text{cv}}$ using lists in XSB.

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
MONA	N/A	0.05	9.22	0.68	0.01	1.23	0.01	0.01	0.01	2.11
XSB B	0.01	0.01	0.01	0.02	0.01	0.01	1.42	0.01	N/A	1.62

Figure 3.5: Performance (secs) w.r.t. increasing number of quantifiers

	#7	#8	#9	#10	#2	#4	#3	#5	#6	#1
MONA	0.01	0.01	0.01	2.11	0.05	0.68	9.22	0.01	1.23	N/A
XSB B	1.42	0.01	N/A	1.62	0.01	0.02	0.01	0.01	0.01	0.01

Figure 3.6: Performance (secs) w.r.t. increasing number of variables

3.3.2 A Decision Procedure based on Top-down Automata

In this section, we outline the decision procedure for WS2S based on top-down automata. Automata-based decision procedures for logics with semantics over infinite trees use top-down automata, hence this section can be thought of an introduction to these procedures. We provide a representation which can be defined analogously to 3.1 and give automata operations as Datalog$^{\text{cv}}$ views.

Representation of Automata

First, we fix the representation for top-down tree automata that capture models of WS2S formulas. Given a WS2S formula φ with free variables $x_1, \ldots, x_k$ we define a Datalog$^{\text{cv}}$ program P_φ with the following predicates:

1. $\mathsf{Node}_\varphi(n)$ representing the nodes of A_φ,
2. $\mathsf{Start}_\varphi(n)$ representing the starting state,
3. $\mathsf{Final}_\varphi(n)$ representing the set of final states, and
4. $\mathsf{Trans}_\varphi(nf_1, nt_1, nt_2, \overline{x})$ representing the transition function δ_φ as a relation such that $(q, t_1, t_2, \sigma) \in \delta_\varphi$ if there is a transition in A_φ from node q to nodes t_1 and t_2 with letter σ.

where $\overline{x} = \{x_1, x_2, \ldots, x_k\}$ is the set of free variables of φ; concatenation of their binary valuations represents a letter of A_φ's alphabet.

This representation is the same as that for finite and bottom-up automata except for the representation of the transition relation. Again, the automata for the atomic formulas can be defined similarly to the automata for the atomic formulas in WS1S.

Automata-theoretic Operations

As in the cases for finite and bottom-up tree automata we define the appropriate automata-theoretic operations: negation, conjunction, projection, and determinization used in decision procedures for WS2S as programs in Datalog$^{\mathrm{cv}}$ as follows.

Definition 3.33 The program $P_{\neg\alpha}$ consists of the following clauses added to the program P_α:

1. $\mathsf{Node}_{\neg\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n)$
2. $\mathsf{Start}_{\neg\alpha}(n) \leftarrow \mathsf{Start}_\alpha(n)$
3. $\mathsf{Final}_{\neg\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n), \neg\mathsf{Final}_\alpha(n)$
4. $\mathsf{Trans}_{\neg\alpha}(nf_1, nt_1, nt_2, \overline{x}) \leftarrow \mathsf{Trans}_\alpha(nf_1, nt_1, nt_2, \overline{x})$

This definition is the same as definitions 3.4 and 3.21 except for the last rule. The following lemma is immediate:

Lemma 3.34 *If P_α represents A_α then $P_{\neg\alpha}$ represents $A_{\neg\alpha}$.*

The proof of this lemma is the same as the proof of lemma 3.4. The conjunction automaton is defined as follows.

Definition 3.35 The program $P_{\alpha_1\wedge\alpha_2}$ consists of the union of programs P_{α_1} and P_{α_2} and the following clauses

1. $\mathsf{Node}_{\alpha_1\wedge\alpha_2}([n_1, n_2]) \leftarrow \mathsf{Node}_{\alpha_1}(n_1), \mathsf{Node}_{\alpha_2}(n_2)$
2. $\mathsf{Start}_{\alpha_1\wedge\alpha_2}([n_1, n_2]) \leftarrow \mathsf{Start}_{\alpha_1}(n_1), \mathsf{Start}_{\alpha_2}(n_2)$
3. $\mathsf{Final}_{\alpha_1\wedge\alpha_2}([n_1, n_2]) \leftarrow \mathsf{Final}_{\alpha_1}(n_1), \mathsf{Final}_{\alpha_2}(n_2)$
4. $\mathsf{Trans}_{\alpha_1\wedge\alpha_2}([nf_{11}, nf_{21}], [nt_{11}, nt_{21}], [nt_{12}, nt_{22}], \overline{x}, \overline{y}, \overline{z}) \leftarrow$
   $\mathsf{Trans}_{\alpha_1}(nf_{11}, nt_{11}, nt_{12}, \overline{x}, \overline{y}),\ \mathsf{Trans}_{\alpha_2}(nf_{21}, nt_{21}, nt_{22}, \overline{y}, \overline{z})$

The sets of variables $\overline{x}, \overline{y}$ represent the free variables of the formula A_{α_1} and $\overline{y}, \overline{z}$ of the formula A_{α_2}.

This definition is same as the definitions 3.7 and 3.23 except for the last rule defining the transition relation. Again, we have the following lemma immediate from the definition:

Lemma 3.36 *Let P_{α_1} represent A_{α_1} and P_{α_2} represent A_{α_2}. Then $P_{\alpha_1 \wedge \alpha_2}$ represents $A_{\alpha_1 \wedge \alpha_2}$.*

The proof of this lemma is the same as the proof of lemma 3.9. The projection automaton is defined as follows.

Definition 3.37 The program $P^u_{\exists \overline{x}:\alpha}$ is defined as the union of P_α with the clauses

1. $\mathsf{Node}^u_{\exists \overline{x}:\alpha}(n) \leftarrow \mathsf{Node}_\alpha(n)$
2. $\mathsf{Start}^u_{\exists \overline{x}:\alpha}(n) \leftarrow \mathsf{Start}_\alpha(n)$
3. $\mathsf{Final}^u_{\exists \overline{x}:\alpha}(n) \leftarrow \mathsf{Final}_\alpha(n)$
   $\mathsf{Final}^u_{\exists \overline{x}:\alpha}(n_0) \leftarrow \mathsf{Trans}_\alpha(n_0, n_1, n_2, \overline{x}, \overline{o}), \mathsf{Final}^u_{\exists \overline{x}:\alpha}(n_1),\ \mathsf{Final}^u_{\exists \overline{x}:\alpha}(n_2)$
4. $\mathsf{Trans}^u_{\exists \overline{x}:\alpha}(nf_1, nt_1, nt_2, \overline{y}) \leftarrow \mathsf{Trans}_\alpha(nf_1, nt_1, nt_2, \overline{x}, \overline{y})$

The sets of variables $\overline{y}$ and $\overline{x}$ represent the free variables of the formula α, and $\overline{o} = \{0, 0, \ldots, 0\}$ where $|\overline{o}| = |\overline{y}|$.

Definition 3.38 For an automaton $A_\alpha = (\Sigma_\alpha,\ Q_\alpha,\ S_\alpha,\ \delta_\alpha, F_\alpha)$ the automaton $A^u_{\exists \overline{x}:\alpha} = (Q^u_{\exists \overline{x}:\alpha},\ \Sigma^u_{\exists \overline{x}:\alpha},\ S^u_{\exists \overline{x}:\alpha},\ \delta^u_{\exists \overline{x}:\alpha},\ F^u_{\exists \overline{x}:\alpha})$, where $Q^u_{\exists \overline{x}:\alpha} = Q_\alpha$, $\Sigma^u_{\exists \overline{x}:\alpha} = \{\overline{y} \mid \overline{xy} \in \Sigma_\alpha\}$, $S^u_{\exists \overline{x}:\alpha} = S_\alpha$, $F^u_{\exists \overline{x}:\alpha} = F_\alpha \cup F^i$, where $F^i = \{n_0 \in Q_\alpha \mid \exists \mu \in L^i$: there is a reduction from n_0 *to an* (f_1, f_2) where $f_1 \in F_\alpha$ and $f_2 \in F_\alpha$ *with* $\mu\}$, and $L^i = \{\omega \in (\{0,1\}^k)^* \mid$ *the $j-th$ track of ω is of the form 0^* for $j \neq i$*$\}$, where k is the number of free variables in α, and i is the track which corresponds to the interpretation of the quantified variable $\overline{x}$, and $(n_{f_1}, n_{t_1}, n_{t_2}, \overline{y}) \in \delta^u_{\exists \overline{x}:\alpha}$ iff $(n_{f_1}, n_{t_1}, n_{t_2}, \overline{x}, \overline{y}) \in \delta_\alpha$.

Lemma 3.39 *If P_α represents A_α then $P^u_{\exists \overline{x}:\alpha}$ represents $A^u_{\exists \overline{x}:\alpha}$ which is nondeterministic automaton for the formula $\exists \overline{x} : \alpha$.*

<u>Proof:</u> Using rules 1, 2, and 3, we can conclude that $Q^u_{\exists \overline{x}:\alpha} = Q_\alpha$, $S^u_{\exists \overline{x}:\alpha} = S_\alpha$, $F^u_{\exists \overline{x}:\alpha} = F_\alpha \cup F^i$, from rule 4, $\Sigma^u_{\exists \overline{x}:\alpha} = \{\overline{y} \mid \overline{xy} \in \Sigma_\alpha\}$, and $(n_{f_1}, n_{t_1}, n_{t_2}, \overline{y}) \in \delta^u_{\exists \overline{x}:\alpha}$ iff $(n_{f_1}, n_{t_1}, n_{t_2}, \overline{x}, \overline{y}) \in \delta_\alpha$. □

Again, the automaton obtained by the projection operation is nondeterministic. The following Datalog$^{\text{cv}}$ program produces the representation of a deterministic automaton which accepts the same language as the nondeterministic one.

Definition 3.40 The program $P_{\exists \overline{x}:\alpha}$ consists of the program $P^u_{\exists \overline{x}:\alpha}$ and the following clauses

1. $\mathsf{Node}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Start}_{\exists\overline{x}:\alpha}(N)$
   $\mathsf{Node}_{\exists\overline{x}:\alpha}(N_1) \leftarrow \mathsf{Trans}_{\exists\overline{x}:\alpha}(N, N_1, N_2, \overline{x})$
   $\mathsf{Node}_{\exists\overline{x}:\alpha}(N_2) \leftarrow \mathsf{Trans}_{\exists\overline{x}:\alpha}(N, N_1, N_2, \overline{x})$
2. $\mathsf{Start}_{\exists\overline{x}:\alpha}(\{n\}) \leftarrow \mathsf{Start}^u_{\exists\overline{x}:\alpha}(n)$
3. $\mathsf{Final}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Node}_{\exists\overline{x}:\alpha}(N), \mathsf{Final}^u_{\exists\overline{x}:\alpha}(n), n \in N$
4. $\mathsf{Trans}_{\exists\overline{x}:\alpha}(N_1, \langle n_1\rangle, \langle n_2\rangle, \overline{x}) \leftarrow \mathsf{Node}_{\exists\overline{x}:\alpha}(N_1),\ \mathsf{Next}_{\exists\overline{x}:\alpha}(N_1, n_1, n_2, \overline{x})$
   $\mathsf{Next}_{\exists\overline{x}:\alpha}(N_1, n_2, n_3, \overline{x}) \leftarrow n_1 \in N_1,\ \mathsf{Trans}^u_{\exists\overline{x}:\alpha}(n_1, n_2, \overline{x}),\ \mathsf{Trans}^u_{\exists\overline{x}:\alpha}(n_1, n_3, \overline{x})$

Definition 3.41 Let $A^u_{\exists\overline{x}:\alpha} = (\Sigma^u_{\exists\overline{x}:\alpha}, Q^u_{\exists\overline{x}:\alpha}, S^u_{\exists\overline{x}:\alpha}, \delta^u_{\exists\overline{x}:\alpha}, F^u_{\exists\overline{x}:\alpha})$ be a nondeterministic automaton. Then $A_{\exists\overline{x}:\alpha} = (Q_{\exists\overline{x}:\alpha}, \Sigma_{\exists\overline{x}:\alpha}, S_{\exists\overline{x}:\alpha}, \delta_{\exists\overline{x}:\alpha}, F_{\exists\overline{x}:\alpha})$, where $Q_{\exists\overline{x}:\alpha}$ is a subset of the power set of $Q^u_{\exists\overline{x}:\alpha}$ that contains only the nodes n_2 where there is a reduction from $n_1 \in S_{\exists\overline{x}:\alpha}$ to (n_2, n_3) by $\delta_{\exists\overline{x}:\alpha}$ and n_3 where there is a reduction from $n_1 \in S_{\exists\overline{x}:\alpha}$ to (n_2, n_3) by $\delta_{\exists\overline{x}:\alpha}$, $\Sigma_{\exists\overline{x}:\alpha}$ is equal to $\Sigma^u_{\exists\overline{x}:\alpha}$, $S_{\exists\overline{x}:\alpha}$ is a singleton set such that $S^u_{\exists\overline{x}:\alpha} \in S_{\exists\overline{x}:\alpha}$, $F_{\exists\overline{x}:\alpha}$ is the set of all states in $Q_{\exists\overline{x}:\alpha}$ containing at least one final state of $A^u_{\exists\overline{x}:\alpha}$, $\delta_{\exists\overline{x}:\alpha}$ is the transition function of $A_{\exists\overline{x}:\alpha}$ where $(Q_F, Q_{T_1}, Q_{T_2}, \overline{x}) \in \delta_{\exists\overline{x}:\alpha}$ iff $(n_f, n_{t_1}, n_{t_2}, \overline{x}) \in \delta^u_{\exists\overline{x}:\alpha}$ for all $n_f \in Q_F$, $n_{t_1} \in Q_{T_1}$ and $n_{t_2} \in Q_{T_2}$.

Lemma 3.42 *If $P^u_{\exists\overline{x}:\alpha}$ represents $A^u_{\exists\overline{x}:\alpha}$ then $P_{\exists\overline{x}:\alpha}$ represents a deterministic automaton $A_{\exists\overline{x}:\alpha}$.*

Proof: Using rules 1, 2, and 3, we can conclude that $Q_{\exists\overline{x}:\alpha}$ is a subset of the power set of $Q^u_{\exists\overline{x}:\alpha}$ that contains only the nodes n_2 where there is a reduction from $(n_1 \in S_{\exists\overline{x}:\alpha}$ to (n_2, n_3) by $\delta_{\exists\overline{x}:\alpha}$ and n_3 where there is a reduction from $(n_1 \in S_{\exists\overline{x}:\alpha}$ to (n_2, n_3) by $\delta_{\exists\overline{x}:\alpha}$, $S_{\exists\overline{x}:\alpha}$ is a singleton set such that $S^u_{\exists\overline{x}:\alpha} \in S_{\exists\overline{x}:\alpha}$, $F_{\exists\overline{x}:\alpha}$ is the set of all states in $Q_{\exists\overline{x}:\alpha}$ containing at least one final state of $A^u_{\exists\overline{x}:\alpha}$, since relation $\mathsf{Node}_{\exists\overline{x}:\alpha}$ represents $Q_{\exists\overline{x}:\alpha}$, $\mathsf{Start}_{\exists\overline{x}:\alpha}$ represents $S_{\exists\overline{x}:\alpha}$, and $\mathsf{Final}_{\exists\overline{x}:\alpha}$ represents $F_{\exists\overline{x}:\alpha}$. Using rule 4, $\Sigma_{\exists\overline{x}:\alpha}$ is equal to $\Sigma^u_{\exists\overline{x}:\alpha}$, and $(Q_F, Q_{T_1}, Q_{T_2}, \overline{x}) \in \delta_{\exists\overline{x}:\alpha}$ iff $(n_f, n_{t_1}, n_{t_2}, \overline{x}) \in \delta^u_{\exists\overline{x}:\alpha}$ for all $n_f \in Q_F$, $n_{t_1} \in Q_{T_1}$ and $n_{t_2} \in Q_{T_2}$, since $\mathsf{Trans}_{\exists\overline{x}:\alpha}$ represents $\delta_{\exists\overline{x}:\alpha}$. □

Last, we define the test for emptiness of an automaton to find out whether the language accepted by A_α is non-empty and thus whether α is satisfiable using a *reachability (transitive closure)* query.

Definition 3.43 The following program TC_α computes the transitive closure of the transition function of A_α.

1. $\mathsf{TransClos}_\alpha(n_1, N) \leftarrow \mathsf{Trans}_\alpha(n_1, n_2, n_3, \overline{x}),\ \mathsf{Final}_\alpha(n_2),\ \mathsf{Final}_\alpha(n_3),\ N = \{n_2\} \cup \{n_3\}$
   $\mathsf{TransClos}_\alpha(n_1, N) \leftarrow \mathsf{Trans}_\alpha(n_1, n_2, n_3, \overline{x}),\ \mathsf{Final}_\alpha(n_2),\ \mathsf{TransClos}_\alpha(n_3, N_5),$

$$
\begin{aligned}
& \qquad N = \{n_2\} \cup N_5 \\
& \mathsf{TransClos}_\alpha(n_1, N) \leftarrow \mathsf{Trans}_\alpha(n_1, n_2, n_3, \overline{x}),\ \mathsf{Final}_\alpha(n_3),\ \mathsf{TransClos}_\alpha(n_2, N_5), \\
& \qquad N = \{n_3\} \cup N_5 \\
& \mathsf{TransClos}_\alpha(n_1, N) \leftarrow \mathsf{Trans}_\alpha(n_1, n_2, n_3, \overline{x}),\ \mathsf{TransClos}_\alpha(n_2, N_4),\ \mathsf{TransClos}_\alpha(n_3, N_5), \\
& \qquad N = N_4 \cup N_5
\end{aligned}
$$

We have the following theorem showing how our procedure decides for the satisfiability of a given WS2S formula.

Theorem 3.44 *Let φ be a WS2S formula such that the models of φ are computed by a top-down tree automaton. Then φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x)$, $\mathsf{TransClos}_\varphi(x, N)$.*

Proof: We know that φ is satisfiable iff the transition relation can generate a subtree from a starting state $s_\varphi \in S_\varphi$ such that all the leaves $f_\varphi \in \mathsf{Final}_\varphi$. Start_φ represents S_φ, and Final_φ represents F_φ. The transition relation generates a subtree from a starting state $s_\varphi \in S_\varphi$ such that all the leaves $f_\varphi \in \mathsf{Final}_\varphi$ iff $x \in \mathsf{Start}_\varphi$, and $(x, N) \in \mathsf{TransClos}_\varphi$. Hence, φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x), \mathsf{TransClos}_\varphi(x, N)$. □

3.4 Heuristics and Optimizations

In this section we give heuristics and optimizations for conjunctions of formulas, negated formulas, and existential formulas.

3.4.1 Large Conjunctions of Formulas

Representing theories that capture database schemata and/or UML diagrams often leads to large conjunctions of relatively simple formulas. Hence we develop heuristics that improve on the naive translation of a formula φ to a Datalog$^{\text{cv}}$ program P_φ presented in this chapter. Many of these heuristics are based on adapting existing optimization techniques for logic programs.

First, given a formula $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ we have to decide which way the conjunctions should be associated (parenthesized). Figure 3.7 shows how performance depends on parenthesizing of a 4-way conjunction. In the experiment we test all permutations of two sets of 4 formulas w.r.t. all possible parenthesizations. The table reports the best and average times over all permutations for a given parenthesization. The results show that left associative parenthesizing is generally preferable.

Formula	Parenthesizing	Best Time	Average Time	# Not Answered
1	$\varphi_i \wedge (\varphi_j \wedge (\varphi_k \wedge \varphi_l))$	0.56	23.23	5
	$((\varphi_i \wedge \varphi_j) \wedge (\varphi_k \wedge \varphi_l))$	0.77	7.00	4
	$((\varphi_i \wedge \varphi_j) \wedge \varphi_k) \wedge \varphi_l$	0.62	5.67	0
	$(\varphi_i \wedge (\varphi_j \wedge \varphi_k)) \wedge \varphi_l$	0.61	8.60	1
	$\varphi_i \wedge ((\varphi_j \wedge \varphi_k) \wedge \varphi_l)$	0.59	10.26	6
2	$\varphi_i \wedge (\varphi_j \wedge (\varphi_k \wedge \varphi_l))$	0.72	14.56	12
	$((\varphi_i \wedge \varphi_j) \wedge (\varphi_k \wedge \varphi_l))$	1.26	25.43	8
	$((\varphi_i \wedge \varphi_j) \wedge \varphi_k) \wedge \varphi_l$	0.94	24.18	2
	$(\varphi_i \wedge (\varphi_j \wedge \varphi_k)) \wedge \varphi_l$	1.65	27.00	5
	$\varphi_i \wedge ((\varphi_j \wedge \varphi_k) \wedge \varphi_l)$	0.74	18.96	11

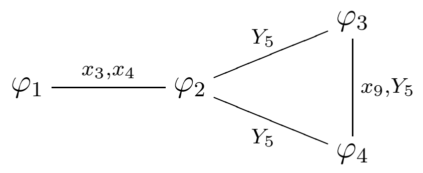
Figure 3.7: Performance (secs) Results w.r.t. Associativity

To take advantage of the structure of the input conjunction, we propose another heuristic that produces a more appropriate goal ordering. We use a structure called a *formula graph*: the nodes of the graph G_φ are the conjuncts of φ and the edges connect formulas that share (free) variables (the edge labels list the shared variables).

Example 3.45 Consider a formula $\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ where:

$$\begin{aligned}
\varphi_1 &= \exists x_{10} : (((\exists Y_{30} : ((x_{10} \in Y_{30}) \wedge (x_4 \in Y_{30}))) \wedge (\exists Y_{40} : (x_{10} \in Y_{40}))) \\
&\quad \wedge \neg(\exists Y_{20} : ((x_{10} \in Y_{10}) \wedge (x_3 \in Y_{20})))) \\
\varphi_2 &= \neg((\exists x_1 : ((x_1 \in Y_5) \wedge (x_2 \in Y_5))) \wedge ((x_3 \in Y_5) \wedge (x_4 \in Y_5))) \\
\varphi_3 &= \neg(x_9 \in Y_5) \\
\varphi_4 &= (x_9 \in Y_5)
\end{aligned}$$

The formula graph for φ is as follows:



For this heuristic we also need to estimate size of the automaton A_φ. We use only very simple estimation rules for the automata operations:

- $|A_{\neg\varphi}| = |A_\varphi|$

- $|A_{\varphi_1 \wedge \varphi_2}| = |A_{\varphi_1}| \times |A_{\varphi_2}|$
- $|A_{\exists \overline{x}:\varphi}| = 2^{|A_\varphi|}$

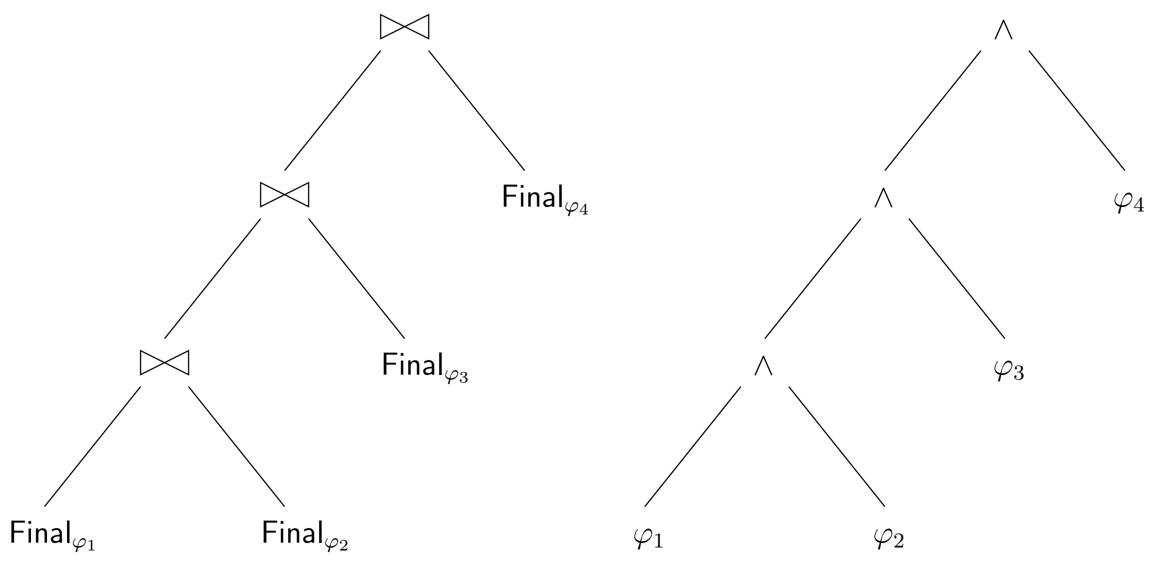
The goal ordering heuristics for a formula $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ constructs a left-associative permutation as follows: it starts from the conjunct that has the largest estimated automaton and then finds its neighbor with the largest automaton (alternatively selecting another conjunct if there are no conjuncts left that satisfy this criteria). This step is repeated until all the conjuncts are processed. Intuitively in the case where a conjunction is applied on a large automaton and a small automaton, when top-down evaluation is used, for every final state we find in the first automaton we check all the final states of the second one and see if they form a final state in the conjunction automaton. Since we iterate on a small sized automaton in this case, ordering the formulas starting from the large ones is heuristically better. The experimental results shown in Figure 3.9 support this optimization. In the table *heuristic time* is the response time of the program for the rewriting generated by the proposed heuristics, *best time* is the fastest response time among all the programs generated for the formula, and similarly *worst time* is the slowest response time. The experiments show that in many cases the heuristic achieves a performance close to the performance of the program for the best possible ordering.

Heuristics described for conjunctions of formulas can be closely related to join order optimization [14, 91] as shown by an example in Figure 3.8. The first heuristic which is choosing left associative parenthesizing can be related to choosing left deep plans in query optimization for join evaluation. The second heuristic on goal ordering can be related to join order selection. The proposed approach tries to minimize the number of tuples visited to find the first answer that satisfies the query.

3.4.2 Negated and Existential Formulas

In addition to conjunctive formulas, we also consider negated and existential formulas. Negations can be classified as negations of conjunctions and negations of existential formulas. The satisfiability problem for negations of conjunctions can be answered efficiently since it is reduced to finding a non-final state in a conjunct. The experimental results in Figure 3.10 show that the ordering of the conjuncts for negated conjunctions does not have much impact on the performace of the satisfiability query. In the table *best time* is the fastest response time among all the programs generated for the formula, and similarly *worst time* is the slowest response time.

On the other hand, the satisfiability problem for negations of existential formulas of the form $\varphi = \exists \overline{x} : \alpha$ can not be answered efficiently especially when the scope of the



$\bowtie$
$\bowtie$
$\mathsf{Final}_{\varphi_4}$
$\bowtie$
$\mathsf{Final}_{\varphi_3}$
$\mathsf{Final}_{\varphi_1}$
$\mathsf{Final}_{\varphi_2}$
$\wedge$
$\wedge$
φ_4
$\wedge$
φ_3
φ_1
φ_2

Figure 3.8: Relation between join ordering and formula rewriting

quantifier exceeds a certain limit. This is due to the fact that, we need to find all the final states of the determinized automaton representing the existential formula, $\exists\overline{x} : \alpha$, to find a final state in the negation automaton. To avoid this problem, we extended the rules for determinization to answer the satisfiability query for these types of formulas more efficiently. The extensions are given in Definition 3.46 and Definition 3.47.

Definition 3.46 The program $P'_{\exists\overline{x}:\alpha}$ consists of the program $P_{\exists\overline{x}:\alpha}$ and the following clauses

1. $\mathsf{Final\_Set}^{u}_{\exists\overline{x}:\alpha}(\langle n\rangle) \leftarrow \mathsf{Final}^{u}_{\exists\overline{x}:\alpha}(n)$
2. $\mathsf{Not\_Final}_{\exists\overline{x}:\alpha}(N) \leftarrow \mathsf{Final\_Set}^{u}_{\exists\overline{x}:\alpha}(N_1), \mathsf{Node}_{\exists\overline{x}:\alpha}(N), N \cap N_1 = \emptyset$

Definition 3.47 The program $P'_{\neg\exists\overline{x}:\alpha}$ consists of the program $P_{\neg\exists\overline{x}:\alpha}$ given by Definition 3.4 where the 3rd rule is updated with the following one:

1. $\mathsf{Final}_{\neg\exists\overline{x}:\alpha}(n) \leftarrow \mathsf{Not\_Final}_{\exists\overline{x}:\alpha}(n)$

Above extensions allow us to compute the final states in the negation automaton without computing all the final states of the determinization automaton. In Definition 3.46, the first rule computes the set of all the final states in the projection automaton representing the existential formula $\exists\overline{x} : \alpha$ and the second rule checks if the intersection of this set and the set representing a state of the determinized automaton is empty to find a final state in the negation automaton.

# of Conjuncts	Formula	Heuristic Time	Best Time	Worst Time
3	1	68.17	67.97	N/A
	2	68.45	68.45	N/A
	3	7.60	7.60	N/A
	4	94.46	1.04	N/A
	5	N/A	5.14	N/A
	6	0.42	0.42	3.18
4	1	1.06	0.56	N/A
	2	3.81	0.72	N/A
	3	0.66	0.64	N/A
5	1	12.61	0.94	N/A
	2	15.94	0.92	N/A
	3	2.6	0.50	N/A

Figure 3.9: Performance (secs) results on ordering

Experimental results presented in Figure 3.11 show that we have better results using the extended set of rules for determinization. By the extension, we increase the size of the set of formulas we can check for satisfiability.

Last, we consider existential formulas. We do not need to perform the determinization step after projection for these types of formulas which results in up to an exponential saving in space and time.

Bibliographical Notes

The automata-theoretic approach for monadic logics over finite words was developed by [9, 25, 89]. It was then extended to infinite words in [10], to finite trees in [87], and generalized to infinite trees in [73]. An extensive survey on automata and logics can be found in [88, 36]. The MONA system [40, 47] is an implementation of automata-based decision procedures for WS1S and WS2S. The deductive database system used in the experiments presented in this chapter is XSB [74, 80]. In addition, the query evaluation techniques: magic set rewriting and SLG resolution used in the decision procedures proposed in this chapter can be found in [4, 64] and in [17, 16] respectively.

# of Conjuncts	Formula	Best Time	Worst Time
4	1	0.01	0.07
	2	0.01	0.07
	3	0.01	0.07
	4	0.01	0.14
	5	0.01	0.07
5	1	0.01	0.07
	2	0.01	0.07
	3	0.01	0.07
6	1	0.01	0.05

Figure 3.10: Performance (secs) results on ordering for negated conjunctions

	#1	#2	#3	#4	#5
Previous	11.87	10.35	N/A	N/A	N/A
Extended	16.30	10.97	10.87	22.07	44.20

Figure 3.11: Performance (secs) w.r.t. increasing number of variables

Chapter 4

Logic Programming Approach to Decision Procedures for S1S

We show that an approach similar to the one we introduced in Chapter 3 can be used for implementing S1S decision procedures. However, in the case of S1S, an automaton on infinite words must be used. The complementation operation for automata on infinite words (e.g. Büchi automata) is considerably more complicated than that for automata on finite words. In this chapter, we provide a mapping for the complementation operation proposed by Kupferman and Vardi [53] to Datalog$^{\text{cv}}$ views which differs from the complementation operation on finite word automata used for deciding WS1S. Hence, given a S1S formula φ we create a Datalog$^{\text{cv}}$ program P_φ such that an answer to a reachability/transitive closure goal w.r.t. this program proves satisfiability of φ. We also show that we can use formula rewriting to transform negated conjunctions to formulas with disjunctions and use union operation on automata which represents the disjunction operation (when converting the satisfiability problem to the emptiness problem on automata) to optimize our decision procedure for various formulas.

4.1 Introduction

The complementation problem on Büchi automata has numerous applications in formal verification. Specification formalisms such as ETL [102] and μTL [94] involve complementation of Büchi automata, and the difficulty of complementing Büchi automata is an obstacle to practical use.

We propose a solution based on expressing the complementation operation for Büchi automata [53] as Datalog$^{\text{cv}}$ views which also extends our translation for WSnS to S1S.

We also propose an optimization for the satisfiability problem of formulas with negated conjunctions. The method is based on rewriting subformulas of the form $\neg(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k)$ as $(\neg\varphi_1 \vee \neg\varphi_2 \vee \ldots \vee \neg\varphi_k)$. We show that such a rewriting considerably reduces the state space for formulas with large negated conjunctions.

4.2 S1S and Automata Connection

First we show how to construct an automaton on infinite words that accepts models of a given formula.

Similar to the case of WS1S an automaton can be constructed for each atomic formula. The automaton $A_{\varphi\wedge\phi}$ is the product automaton of A_φ and A_ϕ and accepts $L(A_\varphi) \cap L(A_\phi)$, the satisfying interpretations of $\varphi \wedge \phi$. The automaton $A_{\exists x:\varphi}$, the projection automaton of A_φ, accepts satisfying interpretations of $\exists x : \varphi$.

The complementation operation on nondeterministic Büchi automata on infinite words (NBW) is not as simple as the one for automata on finite words used for deciding WS1S. An approach to this problem has been proposed by Kupferman and Vardi [53]. The approach for NBW complementation involves the following steps:

1. Dualize the transition function and the acceptance condition resulting in a Universal Co-Büchi Word Automaton (UCW) A',
2. Translate the UCW A' to a Non-deterministic Büchi Word Automaton (NBW) A''.

Theorem 4.1 [37, 53] *Let A be a UCW with n states. There is an NBW A' with at most $3^n \cdot (2n-1)^n$ states such that $L(A') = L(A)$.*

Let $A = \langle \Sigma, Q, S, \delta, F\rangle$ be a UCW where $\delta : Q \times \Sigma \to 2^Q$. There is an NBW $A' = \langle \Sigma, Q', S', \delta', F'\rangle$ such that $L(A) = L(A')$. For an integer k, let $[k] = \{0, 1, \ldots, k\}$, and let $[k]^{odd}$ and $[k]^{even}$ denote the set of odd and even numbers of $[k]$, respectively. A *level ranking* for A is a function $g : Q \to [2n-2]$, such that if $g(q)$ is odd, then $q \notin F$. Let $\mathcal{R}$ be the set of all level rankings. For a subset T of Q and a letter σ, $\delta(T, \sigma) = \bigcup_{s\in T} \delta(s, \sigma)$.

For two level rankings g and g' in $\mathcal{R}$ and a letter σ, g' *covers* $\langle g, \sigma\rangle$ if for all q and q' in Q, if $q' \in \delta(q, \sigma)$, then $g'(q') \leq g(q)$. For $g \in \mathcal{R}$, $odd(g) = \{q : g(q) \in [2n-2]^{odd}\}$.

Now, $A' = \langle \Sigma, Q', Q'_i, \delta', \alpha'\rangle$, where

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$
- $S' = \{S\} \times \{\emptyset\} \times \mathcal{R}$
- δ' is defined, for all $\langle T, O, g\rangle \in Q'$ and $\sigma \in \Sigma$ as follows

  - If $O \neq \emptyset$, then $\delta'(\langle T, O, g\rangle, \sigma) = \{\langle \delta(T, \sigma), \delta(O, \sigma) \setminus odd(g'), g'\rangle : g' \text{ covers } \langle g, \sigma\rangle\}$.
  - If $O = \emptyset$, then $\delta'(\langle T, O, g\rangle, \sigma) = \{\langle \delta(T, \sigma), \delta(T, \sigma) \setminus odd(g'), g'\rangle : g' \text{ covers } \langle g, \sigma\rangle\}$.

- $F' = 2^Q \times \{\emptyset\} \times \mathcal{R}$

Consider a state $\langle T, O, g\rangle \in Q'$, since $O \subseteq T$, there are at most 3^n pairs of T and O that can be members of the same state. In addition, there are at most $(2n-1)^n$ level rankings, hence the number of states in A' is at most $3^n \cdot (2n-1)^n$.

4.3 Representation of Automata

First, we fix the representation for automata that capture models of S1S formulas. Given a S1S formula φ with free variables $x_1, \ldots, x_k$ we define a Datalog$^{\text{cv}}$ program P_φ that defines the following predicates:

1. $\mathsf{Node}_\varphi(n)$ representing the nodes of A_φ,
2. $\mathsf{Start}_\varphi(n)$ representing the set of starting states,
3. $\mathsf{Final}_\varphi(n)$ representing the set of final states, and
4. $\mathsf{Trans}_\varphi(nf_1, nt_1, \overline{x})$ representing the transition relation.

where $\overline{x} = \{x_1, x_2, \ldots, x_k\}$ is the set of free variables of φ; concatenation of their binary valuations represents a letter of A_φ's alphabet.

4.4 Automata-theoretic Operations

In this section, we define the appropriate automata-theoretic operations. Conjunction and projection operations on Büchi automata can be represented by the Datalog$^{\text{cv}}$ programs given in Definition 3.7 and Definition 3.10. Hence we use the Datalog$^{\text{cv}}$ programs given in Chapter 3 for WS1S for the automata operations other than complementation.

The negation automaton which represents the negation of a given formula is defined as follows. Suppose P'_α defines the UCW A'_α, we define the program P''_α defining its NBW translation A''_α.

Definition 4.2 The program P'_α consists of the following clauses:

1. $\mathsf{Node}'_\alpha(n) \leftarrow \mathsf{Node}_\alpha(n)$
2. $\mathsf{Start}'_\alpha(n) \leftarrow \mathsf{Start}_\alpha(n)$

3. $\mathsf{Final}'_\alpha(n) \leftarrow \mathsf{Final}_\alpha(n)$
4. $\mathsf{Trans}'_\alpha(nf_1, \langle nt_1\rangle, \overline{x}) \leftarrow \mathsf{Trans}_\alpha(nf_1, nt_1, \overline{x})$

We represent the interval $0 \leq i < 2n-2$, and the sets of even and odd numbers in the interval, the set of functions $\mathcal{R}$, and *odd* defined in Section 4.2 by the following program R''_α.

Definition 4.3 The program R''_α consists of the following clauses.

1. $\mathsf{Number\_k}''_\alpha(0) \leftarrow$
   $\mathsf{Number\_k}''_\alpha(1) \leftarrow$
   $\ldots$
   $\mathsf{Number\_k}''_\alpha(n\_val-1) \leftarrow$
   $\mathsf{Odd\_k}''_\alpha(1) \leftarrow$
   $\mathsf{Odd\_k}''_\alpha(3) \leftarrow$
   $\ldots$
   $\mathsf{Odd\_k}''_\alpha(n\_val-2) \leftarrow$ (if $n-2$ is odd)
   $\mathsf{Odd\_k}''_\alpha(n\_val-1) \leftarrow$ (if $n-1$ is odd)
2. $\mathsf{Two\_k}''_\alpha(N) \leftarrow (q,k_1) \in N, (q,k_2) \in N, k_1 \neq k_2$
   $\mathsf{Final\_Odd}''_\alpha(N) \leftarrow (q,k) \in N, \mathsf{Final}'_\alpha(q), \mathsf{Odd\_k}''_\alpha(k)$
   $\mathsf{R\_Rel}''_\alpha(\langle(q,k)\rangle) \leftarrow \mathsf{Node}'_\alpha(q), \mathsf{Number\_k}''_\alpha(k)$
   $\mathsf{R\_Function}''_\alpha(N) \leftarrow \mathsf{R\_Rel}''_\alpha(M), N \subseteq M, \neg\mathsf{Two\_k}''_\alpha(N), \neg\mathsf{Final\_Odd}''_\alpha(N)$
3. $\mathsf{Odd}''_\alpha(G, \langle Q\rangle) \leftarrow \mathsf{R\_Function}''_\alpha(G), (Q,X) \in G, \mathsf{Odd\_k}''_\alpha(X)$

Here n_val is a constant equal to $2n-2$. We can use arithmetic to compactly represent the finite set of atomic rules given in the first rule. The value of n_val is given so we can use the following rules to define the predicates $\mathsf{Number\_k}''_\alpha$, and $\mathsf{Odd\_k}''_\alpha$.

$\mathsf{Number\_k}''_\alpha(0) \leftarrow$
$\mathsf{Number\_k}''_\alpha(N) \leftarrow \mathsf{Number\_k}''_\alpha(M), N = M+1, N < n\_val$
$\mathsf{Odd\_k}''_\alpha(1) \leftarrow$
$\mathsf{Odd\_k}''_\alpha(N) \leftarrow \mathsf{Odd\_k}''_\alpha(M), N = M+2, N < n\_val$

Lemma 4.4 *If P'_α represents the UCW A'_α then the predicate* $\mathsf{R\_Function}''_\alpha$ *in R''_α represents the set of functions $\mathcal{R}$.*

Proof: $\mathcal{R}$ is the set of functions $f : Q \to [2n-2]$ such that if $f(q)$ is odd then $q \notin F$. In R''_α $\mathsf{R\_Rel}''_\alpha$ represents $Q \times [2n-2]$. Also Node'_α represents Q, $\mathsf{Number\_k}''_\alpha$ represents $[2n-2]$, and $\mathsf{R\_Function}''_\alpha$ picks a set N from $2^{Q\times[2n-2]}$ such that N contains tuples (q,k)

where $q \in \mathsf{Node}'_\alpha$, $k \in \mathsf{Number\_k}''_\alpha$, and if $(q, k_1) \in N$ and $(q, k_2) \in N$ then $k_1 \neq k_2$, hence N represents a partial function $f : Q \to [2n-2]$. Further if $q \in \mathsf{Final}'_\alpha$ and $(q, k) \in N$ then $k \notin \mathsf{Odd\_k}''_\alpha$ which means if q represents an element from F then $f(q)$ is not odd. $\square$

Lemma 4.5 *If the predicate* $\mathsf{R\_Function}''_\alpha$ *in* R''_α *represents the set of functions* $\mathcal{R}$ *then the predicate* Odd''_α *in* R''_α *represents the function Odd.*

Proof: The predicate Odd''_α contains tuples $G, \langle Q \rangle$ such that $G \in \mathsf{R\_Function}''_\alpha, (Q, X) \in G$ and $X \in \mathsf{Odd\_k}''_\alpha$ hence the set represented by $\langle Q \rangle$ contains all the nodes q such that $(q, X) \in G$ where X is odd. As a result the predicate Odd''_α in R''_α represents the function Odd where $\langle Q \rangle$ represents $Odd(g)$ and G represents g given in the definition of Odd. $\square$

We now represent the *covers* relation defined in Section 4.2 by the following program C''_α.

Definition 4.6 The program C''_α consists of the following clauses.

1. $\mathsf{Not\_Covers}''_\alpha(G_1, (G_2, \overline{x})) \leftarrow \mathsf{R\_Function}''_\alpha(G_1), \mathsf{R\_Function}''_\alpha(G_2), \mathsf{Node}'_\alpha(q),$
   $\mathsf{Node}'_\alpha(t), \mathsf{Trans}'_\alpha(q, t, \overline{x}),$
   $(q, l) \in G_1, (t, k) \in G_2, k > l$
2. $\mathsf{Covers}''_\alpha(G_1, (G_2, \overline{x})) \leftarrow \mathsf{R\_Function}''_\alpha(G_1), \mathsf{R\_Function}''_\alpha(G_2), \mathsf{Trans}'_\alpha(q, t, \overline{x}),$
   $\neg\mathsf{Not\_Covers}''_\alpha(G_1, (G_2, \overline{x}))$

The binary relation $>$ is used in this definition which can also be defined using a finite set of atomic rules since the set of tuples this relation can have is finite.

Lemma 4.7 *If* P'_α *represents the UCW* A'_α *and the predicate* $\mathsf{R\_Function}''_\alpha$ *in* R''_α *represents the set of functions* $\mathcal{R}$ *then the predicate* Covers''_α *in* C''_α *represents the covers relation.*

Proof: The predicate $\mathsf{Not\_Covers}''_\alpha$ defines the tuples $(G_1, (G_2, \overline{x}))$ such that for $G_1 \in \mathsf{R\_Function}''_\alpha$ and $G_2 \in \mathsf{R\_Function}''_\alpha$, there is a transition $(q, t, \overline{x}) \in \mathsf{Trans}'_\alpha$, $(q, l) \in G_1$, $(t, k) \in G_2$ where $k > l$. Then $\mathsf{Not\_Covers}''_\alpha$ contains $(G_1, (G_2, \overline{x}))$ pairs such that G_1 does not cover $(G_2, \overline{x})$ according to the definition of *covers*. The predicate Covers''_α defines the tuples $(G_1, (G_2, \overline{x}))$ such that $G_1 \in \mathsf{R\_Function}''_\alpha$, $G_2 \in \mathsf{R\_Function}''_\alpha$, $(q, t, \overline{x}) \in \mathsf{Trans}'_\alpha$ such that $(G_1, (G_2, \overline{x})) \notin \mathsf{Not\_Covers}''_\alpha$, which means G_1 covers $(G_2, \overline{x})$ according to the definition of *covers*. Hence the predicate Covers''_α in C''_α represents the *covers* relation. where G_1 represents g', $(G_2, \overline{x})$ represents $\langle g, \sigma \rangle$. $\square$

Definition 4.8 Suppose P'_α defines the UCW A'_α resulted from dualizing the transition function and the acceptance condition of the NBW for α. The program P''_α consists of the following clauses added to the program $P'_\alpha \cup R''_\alpha \cup C''_\alpha$:

1. $\mathsf{Node\_Set}'_\alpha(\langle n\rangle) \leftarrow \mathsf{Node}'_\alpha(n)$
   $\mathsf{Node}''_\alpha((n_1, n_2, r)) \leftarrow \mathsf{Node\_Set}'_\alpha(n), n_1 \subseteq n, n_2 \subseteq n, \mathsf{R\_Function}''_\alpha(r)$
2. $\mathsf{Start}''_\alpha((\langle n_1\rangle, \{\}, r_1)) \leftarrow \mathsf{Start}_\alpha(n_1), \mathsf{R\_Function}''_\alpha(r_1)$
3. $\mathsf{Trans\_Set}'_\alpha(n, \langle s\rangle, \overline{x}) \leftarrow \mathsf{Trans}'_\alpha(n_1, s, \overline{x}), n_1 \in n$
   $\mathsf{Trans}''_\alpha((n_1, n_2, r_1), (s_1, s_2, g_1), \overline{x}) \leftarrow \mathsf{Node}''_\alpha((n_1, n_2, r_1)),$
   $\mathsf{Covers}''_\alpha(g_1, (r_1, \overline{x})), \mathsf{Trans\_Set}'_\alpha(n_1, s_1, \overline{x}),$
   $(n_2 = \emptyset \rightarrow \mathsf{Trans\_Set}'_\alpha(n_1, s_3, \overline{x}); \mathsf{Trans\_Set}'_\alpha(n_2, s_3, \overline{x})),$
   $\mathsf{Odd}''_\alpha(g_1, s), s_2 = s_3 \setminus s$
4. $\mathsf{Final}''_\alpha((n_1, \{\}, r_1)) \leftarrow \mathsf{Node\_Set}'_\alpha(n), n_1 \subseteq n, \mathsf{R\_Function}''_\alpha(r_1)$

Lemma 4.9 *If* $\mathsf{R\_Function}''_\alpha$ *in* R''_α *represents the set of functions* $\mathcal{R}$*, the predicate* Odd''_α *in* R''_α *represents the function Odd, and the predicate* Covers''_α *in* C''_α *represents the covers relation then* P''_α *represents* A''_α*, which is the automaton representing* $\neg\alpha$*.*

Proof: For a UCW $A'_\alpha = (N'_\alpha, S'_\alpha, \delta'_\alpha, F'_\alpha)$ we construct an NBW $A''_\alpha = (N''_\alpha, S''_\alpha, \delta''_\alpha, F''_\alpha)$. The construction of A''_α is given in Theorem 4.1. Given $n \in \mathsf{Node\_Set}'_\alpha$, $n_1 \subseteq n$ and $n_2 \subseteq n$ iff n_1 and n_2 each represent an element from 2^Q, $r_1 \in \mathsf{R\_Function}_\alpha$ iff r_1 represents a function from $\mathcal{R}$, $\mathsf{Trans\_Set}'_\alpha$ represents δ'_α where $\delta'_\alpha(T, \sigma) = \bigcup_{s\in T} \delta'_\alpha(s, \sigma)$. Further, $(g_1, (r_1, \overline{x})) \in \mathsf{Covers}_\alpha$ iff g_1 covers $\langle r_1, \overline{x}\rangle$ according to the definition of *covers*, and $(g_1, s) \in \mathsf{Odd}_\alpha$ iff s represents $Odd(g)$ and g_1 represents g given in the definition of Odd, we can conclude that Node''_α relation represents N''_α, Start''_α represents S''_α, Final''_α represents F''_α, and Trans''_α represents T''_α. Hence, P''_α represents A''_α, which is the automaton representing $\neg\alpha$. □

Definition 4.10 The program $P_{\neg\alpha}$ consists of the following clauses:

1. $\mathsf{Node}_{\neg\alpha}(n) \leftarrow \mathsf{Node}''_\alpha(n)$
2. $\mathsf{Start}_{\neg\alpha}(n) \leftarrow \mathsf{Start}''_\alpha(n)$
3. $\mathsf{Final}_{\neg\alpha}(n) \leftarrow \mathsf{Final}''_\alpha(n)$
4. $\mathsf{Trans}_{\neg\alpha}(nf_1, nt_1, \overline{x}) \leftarrow \mathsf{Trans}''_\alpha(nf_1, nt_1, \overline{x})$

Last, the test for emptiness of an automaton has to be defined: To find out whether the language accepted by an automaton A_α is non-empty and thus whether α is satisfiable, a *reachability (transitive closure)* query is used.

Definition 4.11 The following program TC_α computes the transitive closure of the transition function of A_α.

1. $\mathsf{TransClos}_\alpha(n, n) \leftarrow$

2. $\mathsf{TransClos}_\alpha(nf_1, nt_1) \leftarrow \mathsf{Trans}_\alpha(nf_1, nt_2, \overline{x}),\ \mathsf{TransClos}_\alpha(nt_2, nt_1)$

Note that the use of magic sets and/or SLG resolution automatically transforms the transitive closure query into a reachability query.

Theorem 4.12 *Let φ be a S1S formula. Then φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x), \mathsf{Final}_\varphi(y), \mathsf{TransClos}_\varphi(x,y), \mathsf{TransClos}_\varphi(y,y)$.*

<u>**Proof:**</u> We know that φ is satisfiable iff A_φ has a path from $s_\varphi \in S_\varphi$ to $f_\varphi \in F_\varphi$ and f_φ is visited infinitely often. Start_φ represents S_φ, and Final_φ represents F_φ. There is a path from $s_\varphi \in S_\varphi$ to $f_\varphi \in F_\varphi$ iff $x \in \mathsf{Start}_\varphi$, $y \in \mathsf{Final}_\varphi$, $(x,y) \in \mathsf{TransClos}_\varphi$, and f_φ is visited infinitely often iff $(y,y) \in \mathsf{TransClos}_\varphi$. Hence, φ is satisfiable if and only if $P_\varphi, TC_\varphi \models \mathsf{Start}_\varphi(x), \mathsf{Final}_\varphi(y), \mathsf{TransClos}_\varphi(x,y), \mathsf{TransClos}_\varphi(y,y)$. □

Example 4.13 Suppose that we have an S1S formula $\alpha = \neg(\alpha_1 \wedge \alpha_2)$, let A_{α_1} be the automaton for the subformula α_1 and A_{α_2} be the automaton for the subformula α_2, we can use the following logic program to construct the automaton A''_α:

$$
\begin{aligned}
&\mathsf{Node}_{\alpha_1\wedge\alpha_2}([n_1,n_2]) \leftarrow \mathsf{Node}_{\alpha_1}(n_1), \mathsf{Node}_{\alpha_2}(n_2)\\
&\mathsf{Start}_{\alpha_1\wedge\alpha_2}([n_1,n_2]) \leftarrow \mathsf{Start}_{\alpha_1}(n_1), \mathsf{Start}_{\alpha_2}(n_2)\\
&\mathsf{Final}_{\alpha_1\wedge\alpha_2}([n_1,n_2]) \leftarrow \mathsf{Final}_{\alpha_1}(n_1), \mathsf{Final}_{\alpha_2}(n_2)\\
&\mathsf{Trans}_{\alpha_1\wedge\alpha_2}([nf_1,nf_2],[nt_1,nt_2],\overline{x},\overline{y},\overline{z}) \leftarrow\\
&\qquad\qquad \mathsf{Trans}_{\alpha_1}(nf_1,nt_1,\overline{x},\overline{y}),\ \mathsf{Trans}_{\alpha_2}(nf_2,nt_2,\overline{y},\overline{z})
\end{aligned}
$$

This part computes the intersection automaton A_α representing the formula $\alpha_1 \wedge \alpha_2$ and its UCT translation A'_α is represented by P'_α as given in Definition 4.2.

$$
\begin{aligned}
&\mathsf{Number\_k}''_\alpha(0) \leftarrow\\
&\mathsf{Number\_k}''_\alpha(1) \leftarrow\\
&\ldots\\
&\mathsf{Number\_k}''_\alpha(n\_val - 1) \leftarrow\\
&\mathsf{Odd\_k}''_\alpha(1) \leftarrow\\
&\mathsf{Odd\_k}''_\alpha(3) \leftarrow\\
&\ldots\\
&\mathsf{Odd\_k}''_\alpha(n\_val - 2) \leftarrow \text{(if } n-2 \text{ is odd)}\\
&\mathsf{Odd\_k}''_\alpha(n\_val - 1) \leftarrow \text{(if } n-1 \text{ is odd)}\\
&\mathsf{Two\_k}''_\alpha(N) \leftarrow (q,k_1) \in N, (q,k_2) \in N, k_1 \neq k_2\\
&\mathsf{Final\_Odd}''_\alpha(N) \leftarrow (q,k) \in N, \mathsf{Final}'_\alpha(q), \mathsf{Odd\_k}''_\alpha(k)\\
&\mathsf{R\_Rel}''_\alpha(\langle(q,k)\rangle) \leftarrow \mathsf{Node}'_\alpha(q), \mathsf{Number\_k}''_\alpha(k)\\
&\mathsf{R\_Function}''_\alpha(N) \leftarrow \mathsf{R\_Rel}''_\alpha(M), N \subseteq M, \neg\mathsf{Two\_k}''_\alpha(N), \neg\mathsf{Final\_Odd}''_\alpha(N)\\
&\mathsf{Odd}''_\alpha(G,\langle Q\rangle) \leftarrow \mathsf{R\_Function}''_\alpha(G), (Q,X) \in G, \mathsf{Odd\_k}''_\alpha(X)
\end{aligned}
$$

This part computes the $\mathcal{R}$ and *Odd* functions for A''_α.

$$\begin{aligned}
&\mathsf{Not\_Covers}''_\alpha(G_1,(G_2,\overline{x})) \leftarrow \mathsf{R\_Function}''_\alpha(G_1), \mathsf{R\_Function}''_\alpha(G_2), \mathsf{Node}'_\alpha(q),\\
&\qquad \mathsf{Node}'_\alpha(t), \mathsf{Trans}'_\alpha(q,t,\overline{x}),\\
&\qquad (q,l) \in G_1, (t,k) \in G_2, k > l\\
&\mathsf{Covers}''_\alpha(G_1,(G_2,\overline{x})) \leftarrow \mathsf{R\_Function}''_\alpha(G_1), \mathsf{R\_Function}''_\alpha(G_2), \mathsf{Trans}'_\alpha(q,t,\overline{x}),\\
&\qquad \neg\mathsf{Not\_Covers}''_\alpha(G_1,(G_2,\overline{x}))
\end{aligned}$$

This part computes the *Covers* relation for A''_α.

$$\begin{aligned}
&\mathsf{Node\_Set}'_\alpha(\langle n\rangle) \leftarrow \mathsf{Node}'_\alpha(n)\\
&\mathsf{Node}''_\alpha((n_1,n_2,r)) \leftarrow \mathsf{Node\_Set}'_\alpha(n), n_1 \subseteq n, n_2 \subseteq n, \mathsf{R\_Function}''_\alpha(r)\\
&\mathsf{Start}''_\alpha((\langle n_1\rangle,\{\},r_1)) \leftarrow \mathsf{Start}_\alpha(n_1), \mathsf{R\_Function}''_\alpha(r_1)\\
&\mathsf{Trans\_Set}'_\alpha(n,\langle s\rangle,\overline{x}) \leftarrow \mathsf{Trans}'_\alpha(n_1,s,\overline{x}), n_1 \in n\\
&\mathsf{Trans}''_\alpha((n_1,n_2,r_1),(s_1,s_2,g_1),\overline{x}) \leftarrow \mathsf{Node}''_\alpha((n_1,n_2,r_1)),\\
&\qquad \mathsf{Covers}''_\alpha(g_1,(r_1,\overline{x})), \mathsf{Trans\_Set}'_\alpha(n_1,s_1,\overline{x}),\\
&\qquad (n_2 = \emptyset \rightarrow \mathsf{Trans\_Set}'_\alpha(n_1,s_3,\overline{x}); \mathsf{Trans\_Set}'_\alpha(n_2,s_3,\overline{x})),\\
&\qquad \mathsf{Odd}''_\alpha(g_1,s), s_2 = s_3 \setminus s\\
&\mathsf{Final}''_\alpha((n_1,\{\},r_1)) \leftarrow \mathsf{Node\_Set}'_\alpha(n), n_1 \subseteq n, \mathsf{R\_Function}''_\alpha(r_1)
\end{aligned}$$

This part computes the negation automaton A''_α representing α.

4.5 Optimization for Formulas with Negated Conjunctions

The proposed logic programming approach for the satisfiability problem of S1S formulas has the same advantages over the traditional automata-based algorithms as the approach for WS1S formulas. Hence the satisfiability questions for formulas with many free variables (i.e. conjunctions) can be solved more efficiently. On the other hand, the most complex operation in this case is the complementation operation, hence satisfiability problem for the formulas with large negated conjunctions is harder to solve.

We propose a method for the satisfiability problem of formulas with large negated conjunctions. The method is based on the transformation of such formulas to formulas with disjunction. Hence given a formula we have two transformation steps:

1. Convert the formula to a formula with projection, conjunction and negation operations

2. Convert the subformulas of the form $\neg(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k)$ to $(\neg\varphi_1 \vee \neg\varphi_2 \vee \ldots \vee \neg\varphi_k)$

Definition 4.14 The program $P_{\alpha_1 \vee \alpha_2}$ is defined as the union of P_{α_1} and P_{α_2} with the clauses

1. $\mathsf{Node}_{\alpha_1\vee\alpha_2}(n_0)$.
   $\mathsf{Node}_{\alpha_1\vee\alpha_2}(n) \leftarrow \mathsf{Node}_{\alpha_1}(n)$
   $\mathsf{Node}_{\alpha_1\vee\alpha_2}(n) \leftarrow \mathsf{Node}_{\alpha_2}(n)$
2. $\mathsf{Start}_{\alpha_1\vee\alpha_2}(n_0)$.
3. $\mathsf{Final}_{\alpha_1\vee\alpha_2}(n) \leftarrow \mathsf{Final}_{\alpha_1}(n)$
   $\mathsf{Final}_{\alpha_1\vee\alpha_2}(n) \leftarrow \mathsf{Final}_{\alpha_2}(n)$
4. $\mathsf{Trans}_{\alpha_1\vee\alpha_2}(n_0, nt_1, \epsilon) \leftarrow \mathsf{Start}_{\alpha_1}(nt_1)$
   $\mathsf{Trans}_{\alpha_1\vee\alpha_2}(n_0, nt_1, \epsilon) \leftarrow \mathsf{Start}_{\alpha_2}(nt_1)$
   $\mathsf{Trans}_{\alpha_1\vee\alpha_2}(nf_1, nt_1, \overline{x}) \leftarrow \mathsf{Trans}_{\alpha_1}(nf_1, nt_1, \overline{x})$
   $\mathsf{Trans}_{\alpha_1\vee\alpha_2}(nf_1, nt_1, \overline{x}) \leftarrow \mathsf{Trans}_{\alpha_2}(nf_1, nt_1, \overline{x})$

Since we use nondeterministic automata using disjunction does not add any determinization steps. In addition, this optimization results in large savings in state space for formulas of this type. Given a formula φ, for each conversion from subformulas of the form $\phi = \neg(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k)$ to $\phi' = (\neg\varphi_1 \vee \neg\varphi_2 \vee \ldots \vee \neg\varphi_k)$ such that the number of states in the autamaton accepting models of φ_1 is n_1, φ_2 is n_2, ..., φ_k is n_k, we can calculate estimated number of states $|A_\phi|$ and $|A_{\phi'}|$ of the automata A_ϕ and $A_{\phi'}$ by the following formulas:

$$|A_\phi| = 3^{n_1 \cdot n_2 \cdot \ldots \cdot n_k} \cdot 2(n_1 \cdot n_2 \cdot \ldots \cdot n_k - 1)^{n_1 \cdot n_2 \cdot \ldots \cdot n_k}$$
$$|A_{\phi'}| = 3^{n_1} \cdot 2(n_1 - 1)^{n_1} + 3^{n_2} \cdot 2(n_2 - 1)^{n_2} + \ldots + 3^{n_k} \cdot 2(n_k - 1)^{n_k}$$

The saving in state space is defined as $|A_\phi| \;/\; |A'_\phi|$. Similarly, we can estimate the number of transitions T_ϕ in A_ϕ and $T_{\phi'}$ in $A_{\phi'}$ where Σ_1 is the set of free variables in φ_1, Σ_2 is the set of free variables in φ_2, ..., Σ_k is the set of free variables in φ_k as follows:

$$|T_\phi| = 3^{n_1 \cdot n_2 \cdot \ldots \cdot n_k} \cdot 2(n_1 \cdot n_2 \cdot \ldots \cdot n_k - 1)^{n_1 \cdot n_2 \cdot \ldots \cdot n_k} \cdot 2^{|\Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_k|}$$
$$|T_{\phi'}| = 3^{n_1} \cdot 2(n_1 - 1)^{n_1} \cdot 2^{|\Sigma_1|} + 3^{n_2} \cdot 2(n_2 - 1)^{n_2} \cdot 2^{|\Sigma_2|} + \ldots + 3^{n_k} \cdot 2(n_k - 1)^{n_k} \cdot 2^{|\Sigma_k|}$$

Theorem 4.15 *For each k-way negated conjunction $\phi = \neg(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k)$ where each conjunct can be represented by an automaton with $O(n)$ states, the conversion to $\phi' = (\neg\varphi_1 \vee \neg\varphi_2 \vee \ldots \vee \neg\varphi_k)$ results in the order of $O((n_k)^{n_k})$ saving in state space.*

<u>**Proof:**</u> From theorem 4.1, the number of states in ϕ is bounded by $|A_\phi| = O(3^{n^k} \cdot 2(n^k - 1)^{n^k})$, the number of states in ϕ' is bounded by $|A_{\phi'}| = O(k \cdot 3^n \cdot 2(n-1)^n)$ hence the saving in state space is $O(3^{n^k} \cdot 2(n^k-1)^{n^k})/O(k \cdot 3^n \cdot 2(n-1)^n) = O((n_k)^{n_k})$. □

Example 4.16 Suppose that we have an S1S formula $\alpha = \neg\alpha_1 \vee \neg\alpha_2$, let A_{α_1} be the automaton for the subformula α_1 and A_{α_2} be the automaton for the subformula α_2, we can use the following logic program to construct the automaton A''_α:

```
$\mathsf{Number\_k}''_{\alpha_1}(0) \leftarrow$
$\mathsf{Number\_k}''_{\alpha_1}(1) \leftarrow$
...
$\mathsf{Number\_k}''_{\alpha_1}(n\_val - 1) \leftarrow$
$\mathsf{Odd\_k}''_{\alpha_1}(1) \leftarrow$
$\mathsf{Odd\_k}''_{\alpha_1}(3) \leftarrow$
...
$\mathsf{Odd\_k}''_{\alpha_1}(n\_val - 2) \leftarrow$ (if $n-2$ is odd)
$\mathsf{Odd\_k}''_{\alpha_1}(n\_val - 1) \leftarrow$ (if $n-1$ is odd)
$\mathsf{Two\_k}''_{\alpha_1}(N) \leftarrow (q,k_1) \in N, (q,k_2) \in N, k_1 \neq k_2$
$\mathsf{Final\_Odd}''_{\alpha_1}(N) \leftarrow (q,k) \in N, \mathsf{Final}'_{\alpha_1}(q), \mathsf{Odd\_k}''_{\alpha_1}(k)$
$\mathsf{R\_Rel}''_{\alpha_1}(\langle (q,k) \rangle) \leftarrow \mathsf{Node}'_{\alpha_1}(q), \mathsf{Number\_k}''_{\alpha_1}(k)$
$\mathsf{R\_Function}''_{\alpha_1}(N) \leftarrow \mathsf{R\_Rel}''_{\alpha_1}(M), N \subseteq M, \neg\mathsf{Two\_k}''_{\alpha_1}(N), \neg\mathsf{Final\_Odd}''_{\alpha_1}(N)$
$\mathsf{Odd}''_{\alpha_1}(G, \langle Q \rangle) \leftarrow \mathsf{R\_Function}''_{\alpha_1}(G), (Q,X) \in G, \mathsf{Odd\_k}''_{\alpha_1}(X)$
```

This part computes the $\mathcal{R}$ and Odd functions for A''_{α_1} representing $\neg\alpha_1$.

```
$\mathsf{Not\_Covers}''_{\alpha_1}(G_1, (G_2, \overline{x})) \leftarrow \mathsf{R\_Function}''_{\alpha_1}(G_1), \mathsf{R\_Function}''_{\alpha_1}(G_2), \mathsf{Node}'_{\alpha_1}(q),$
        $\mathsf{Node}'_{\alpha_1}(t), \mathsf{Trans}'_{\alpha_1}(q,t,\overline{x}),$
        $(q,l) \in G_1, (t,k) \in G_2, k > l$
$\mathsf{Covers}''_{\alpha_1}(G_1, (G_2, \overline{x})) \leftarrow \mathsf{R\_Function}''_{\alpha_1}(G_1), \mathsf{R\_Function}''_{\alpha_1}(G_2), \mathsf{Trans}'_{\alpha_1}(q,t,\overline{x}),$
        $\neg\mathsf{Not\_Covers}''_{\alpha_1}(G_1, (G_2, \overline{x}))$
```

This part computes the $Covers$ relation for A''_{α_1}.

$$\begin{array}{l}
\mathsf{Node\_Set}'_{\alpha_1}(\langle n\rangle) \leftarrow \mathsf{Node}'_{\alpha_1}(n)\\
\mathsf{Node}''_{\alpha_1}((n_1,n_2,r)) \leftarrow \mathsf{Node\_Set}'_{\alpha_1}(n), n_1 \subseteq n, n_2 \subseteq n, \mathsf{R\_Function}''_{\alpha_1}(r)\\
\mathsf{Start}''_{\alpha_1}((n_1,\{\},r_1)) \leftarrow \mathsf{Start}_{\alpha_1}(n_1), \mathsf{R\_Function}''_{\alpha_1}(r_1)\\
\mathsf{Trans\_Set}'_{\alpha_1}(n,\langle s\rangle,\overline{x}) \leftarrow \mathsf{Trans}'_{\alpha_1}(n_1,s,\overline{x}), n_1 \in n\\
\mathsf{Trans}''_{\alpha_1}((n_1,n_2,r_1),(s_1,s_2,g_1),\overline{x}) \leftarrow \mathsf{Node}''_{\alpha_1}((n_1,n_2,r_1)),\\
\qquad\qquad \mathsf{Covers}''_{\alpha_1}(g_1,(r_1,\overline{x})), \mathsf{Trans\_Set}'_{\alpha_1}(n_1,s_1,\overline{x}),\\
\qquad\qquad (n_2=\emptyset \rightarrow \mathsf{Trans\_Set}'_{\alpha_1}(n_1,s_3,\overline{x}); \mathsf{Trans\_Set}'_{\alpha_1}(n_2,s_3,\overline{x})),\\
\qquad\qquad \mathsf{Odd}_{\alpha_1}(g_1,s), s_2 = s_3 \setminus s\\
\mathsf{Final}''_{\alpha_1}((n_1,\{\},r_1)) \leftarrow \mathsf{Node\_Set}'_{\alpha_1}(n), n_1 \subseteq n, \mathsf{R\_Function}''_{\alpha_1}(r_1)
\end{array}$$

This part computes the negation automaton A''_{α_1} representing $\neg\alpha_1$.

$$\begin{array}{l}
\mathsf{Number\_k}''_{\alpha_2}(0) \leftarrow\\
\mathsf{Number\_k}''_{\alpha_2}(1) \leftarrow\\
\ldots\\
\mathsf{Number\_k}''_{\alpha_2}(n\_val-1) \leftarrow\\
\mathsf{Odd\_k}''_{\alpha_2}(1) \leftarrow\\
\mathsf{Odd\_k}''_{\alpha_2}(3) \leftarrow\\
\ldots\\
\mathsf{Odd\_k}''_{\alpha_2}(n\_val-2) \leftarrow (\text{if } n-2 \text{ is odd})\\
\mathsf{Odd\_k}''_{\alpha_2}(n\_val-1) \leftarrow (\text{if } n-1 \text{ is odd})\\
\mathsf{Two\_k}''_{\alpha_2}(N) \leftarrow (q,k_1) \in N, (q,k_2) \in N, k_1 \neq k_2\\
\mathsf{Two\_k}''_{\alpha_2}(N) \leftarrow (q,k_1) \in N, (q,k_2) \in N, k_1 \neq k_2\\
\mathsf{Final\_Odd}''_{\alpha_2}(N) \leftarrow (q,k) \in N, \mathsf{Final}'_{\alpha_2}(q), \mathsf{Odd\_k}''_{\alpha_2}(k)\\
\mathsf{R\_Rel}''_{\alpha_2}(\langle(q,k)\rangle) \leftarrow \mathsf{Node}'_{\alpha_2}(q), \mathsf{Number\_k}''_{\alpha_2}(k)\\
\mathsf{R\_Function}''_{\alpha_2}(N) \leftarrow \mathsf{R\_Rel}''_{\alpha_2}(M), N \subseteq M, \neg\mathsf{Two\_k}''_{\alpha_2}(N), \neg\mathsf{Final\_Odd}''_{\alpha_2}(N)\\
\mathsf{Odd}''_{\alpha_2}(G,\langle Q\rangle) \leftarrow \mathsf{R\_Function}''_{\alpha_2}(G), (Q,X) \in G, \mathsf{Odd\_k}''_{\alpha_2}(X)
\end{array}$$

This part computes the $\mathcal{R}$ and *Odd* functions for A''_{α_2}.

$$\begin{array}{l}
\mathsf{Not\_Covers}''_{\alpha_2}(G_1,(G_2,\overline{x})) \leftarrow \mathsf{R\_Function}''_{\alpha_2}(G_1), \mathsf{R\_Function}''_{\alpha_2}(G_2), \mathsf{Node}'_{\alpha_1}(q),\\
\qquad\qquad \mathsf{Node}'_{\alpha_2}(t), \mathsf{Trans}'_{\alpha_2}(q,t,\overline{x}),\\
\qquad\qquad (q,l) \in G_1, (t,k) \in G_2, k > l\\
\mathsf{Covers}''_{\alpha_2}(G_1,(G_2,\overline{x})) \leftarrow \mathsf{R\_Function}''_{\alpha_2}(G_1), \mathsf{R\_Function}''_{\alpha_2}(G_2), \mathsf{Trans}'_{\alpha_2}(q,t,\overline{x}),\\
\qquad\qquad \neg\mathsf{Not\_Covers}''_{\alpha_2}(G_1,(G_2,\overline{x}))
\end{array}$$

This part computes the *Covers* relation for A''_{α_2}.

$\mathsf{Node\_Set}'_{\alpha_2}(\langle n\rangle) \leftarrow \mathsf{Node}'_{\alpha_2}(n)$
$\mathsf{Node}''_{\alpha_2}((n_1, n_2, r)) \leftarrow \mathsf{Node\_Set}'_{\alpha_2}(n), n_1 \subseteq n, n_2 \subseteq n, \mathsf{R\_Function}''_{\alpha_2}(r)$
$\mathsf{Start}''_{\alpha_2}((\langle n_1\rangle, \{\}, r_1)) \leftarrow \mathsf{Start}_{\alpha_2}(n_1), \mathsf{R\_Function}''_{\alpha_2}(r_1)$
$\mathsf{Trans\_Set}'_{\alpha_2}(n, \langle s\rangle, \overline{x}) \leftarrow \mathsf{Trans}'_{\alpha_2}(n_1, s, \overline{x}), n_1 \in n$
$\mathsf{Trans}''_{\alpha_2}((n_1, n_2, r_1), (s_1, s_2, g_1), \overline{x}) \leftarrow \mathsf{Node}''_{\alpha_2}((n_1, n_2, r_1)),$
$\quad\quad \mathsf{Covers}''_{\alpha_2}(g_1, (r_1, \overline{x})), \mathsf{Trans\_Set}'_{\alpha_2}(n_1, s_1, \overline{x}),$
$\quad\quad (n_2 = \emptyset \rightarrow \mathsf{Trans\_Set}'_{\alpha_2}(n_1, s_3, \overline{x}); \mathsf{Trans\_Set}'_{\alpha_2}(n_2, s_3, \overline{x})),$
$\quad\quad \mathsf{Odd}_{\alpha_2}(g_1, s), s_2 = s_3 \setminus s$
$\mathsf{Final}''_{\alpha_2}((n_1, \{\}, r_1)) \leftarrow \mathsf{Node\_Set}'_{\alpha_2}(n), n_1 \subseteq n, \mathsf{R\_Function}''_{\alpha_2}(r_1)$

This part computes the negation automaton A''_{α_2} representing $\neg\alpha_2$.

$\mathsf{Node}''_{\alpha}(n_0).$
$\mathsf{Node}_{\alpha}(n) \leftarrow \mathsf{Node}''_{\alpha_1}(n)$
$\mathsf{Node}_{\alpha}(n) \leftarrow \mathsf{Node}''_{\alpha_2}(n)$
$\mathsf{Start}''_{\alpha}(n_0).$
$\mathsf{Final}''_{\alpha}(n) \leftarrow \mathsf{Final}''_{\alpha_1}(n)$
$\mathsf{Final}''_{\alpha}(n) \leftarrow \mathsf{Final}''_{\alpha_2}(n)$
$\mathsf{Trans}''_{\alpha}(n_0, nt_1, \epsilon) \leftarrow \mathsf{Start}''_{\alpha_1}(nt_1)$
$\mathsf{Trans}''_{\alpha}(n_0, nt_1, \epsilon) \leftarrow \mathsf{Start}''_{\alpha_2}(nt_1)$
$\mathsf{Trans}''_{\alpha}(nf_1, nt_1, \overline{x}) \leftarrow \mathsf{Trans}''_{\alpha_1}(nf_1, nt_1, \overline{x})$
$\mathsf{Trans}''_{\alpha}(nf_1, nt_1, \overline{x}) \leftarrow \mathsf{Trans}''_{\alpha_2}(nf_1, nt_1, \overline{x})$

This part computes the automaton A''_{α} representing $\alpha = \neg\alpha_1 \vee \neg\alpha_2$.

Consider the formula $\neg(\varphi_1 \wedge \varphi_2)$ given in example 4.13 and $\neg\varphi_1 \vee \neg\varphi_2$ given in this example such that the number of states in A_{φ_1} and A_{φ_2} is 3 and $\Sigma_1 = \Sigma_2$ where $|\Sigma_1| = 2$. Estimated number of states in $A_{\neg(\varphi_1\wedge\varphi_2)} = 3^{3^2} \cdot 2(3^2-1)^{3^2} = 3^9 \cdot 2^{37}$ whereas estimated number of states in $A_{\neg\varphi_1\vee\neg\varphi_2} = 2 \cdot 3^3 \cdot 2(3-1)^3 = 3^3 \cdot 2^4$. Estimated number of transitions in $A_{\neg(\varphi_1\wedge\varphi_2)} = 3^9 \cdot 2^{41}$ whereas estimated number of transitions in $A_{\neg\varphi_1\vee\neg\varphi_2} = 3^3 \cdot 2^8$.

Example 4.17 Consider a S1S formula $\phi = \neg(\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5)$ where $\phi' = \neg\varphi_1 \vee \neg\varphi_2 \vee \neg\varphi_3 \vee \neg\varphi_4 \vee \neg\varphi_5$ such that the number of states in A_{φ_1}, A_{φ_2}, A_{φ_3}, A_{φ_4}, and A_{φ_5} is 2 and $\Sigma_1 = \Sigma_2 = \ldots = \Sigma_5$ where $|\Sigma_1| = 2$. Estimated number of states in $A_\phi = 3^{2^5} \cdot 2(2^5-1)^{2^5} = 2 \cdot 93^{32}$ whereas estimated number of states in $A_{\phi'} = 5 \cdot 3^2 \cdot 2(2-1)^2 = 90$. Estimated number of transitions in $A_\phi = 8 \cdot 93^{32}$ whereas estimated number of transitions in $A_{\phi'} = 360$.

Heuristics we provided for conjunctive WS1S formulas can also be used for S1S formulas. In addition, we propose an optimization for negations of conjunctions of formulas of the form $\phi = \neg(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k)$ which can be thought of breaking the negation operation to smaller steps such that we can test for the satisfiability of ϕ by checking for the emptiness of automata, $A_{\neg\varphi_1}$, $A_{\neg\varphi_2}$, …, $A_{\neg\varphi_n}$, representing $\neg\varphi_1$, $\neg\varphi_2$, …, $\neg\varphi_n$ separately until we find a non-empty automaton or conclude that $A_{\neg\varphi_1}$, $A_{\neg\varphi_2}$, …, $A_{\neg\varphi_n}$ are all empty.

Bibliographical Notes

The complementation problem for Büchi automata that matches the lower bound has been solved by Safra [78]. In our work we use an impoved complementation construction proposed by Kupferman and Vardi [53] which also outlines the complementation constructions for Büchi automata and their complexity results.

Chapter 5

An Incremental Technique for μ-Calculus Decision Procedures

As in the case of WSnS and S1S conjunctive μ-calculus formulas play an important role in many settings such as reasoning in theories that describe system behavior using a conjunction of a large number of relatively simple constraints. In this chapter, we provide a decomposition technique for checking the satisfiability of conjunctive μ-calculus formulas. The satisfiability problem for a μ-calculus formula φ can be translated to the emptiness problem for an alternating parity tree automaton A. Our technique is based on decomposing the emptiness test procedure proposed by Kupferman and Vardi [55] for conjunctive formulas and, in turn, an incremental algorithm for checking the emptiness of an APT A constructed from a formula φ. We also outline a top-down approach that drives this incremental procedure.

Given a conjunctive μ-calculus formula $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ the incremental technique first constructs an automaton A_1 for φ_1 and checks for its emptiness, if A_1 is empty then the procedure stops. Otherwise it continues with automata for formulas $\varphi_1 \wedge \varphi_2$, …, φ applying the same technique and reusing the automaton computed in step i for computing the automaton in step $i + 1$.

5.1 Introduction

Propositional μ-calculus is often considered one of the *lingua franca* logical formalism among logics with EXPTIME decision procedures. Indeed, many other modal, dynamic, temporal, and description logics have been shown to be relatively easily encodable in μ-calculus [23, 49, 81].

The key technique to showing decidability and complexity bounds for μ-calculus is based on capturing *the language of models* of a given formula using an automaton constructed from the formula—usually an *alternating parity automaton*—that accepts infinite tree models of the formula [88, 97, 98]. Hence, testing for satisfiability reduces to testing for non-emptiness of an alternating parity automaton automaton.

The emptiness test for alternating parity automaton, in particular when based on Safra's determinization approach [78, 79], is difficult to implement. This issue, for μ-calculus formulas, was addressed by using simpler *Safraless* decision procedures based on transforming an alternating parity automaton to a non-deterministic Büchi automaton while preserving emptiness [55].

However, even this improvement does not yield a practical reasoning procedure. The difficulties *inherent in the automata-based approaches* are especially apparent when determining *logical consequences* of moderately large *theories* of the form $\{\varphi_1, \ldots \varphi_n\} \models \varphi$, are considered. Commonly, more local search techniques applied to this problem try to discover an inconsistency in the set $\{\varphi_1, \ldots \varphi_n, \neg\varphi\}$, which in practice rarely involves all the formulas φ_i in the input. Hence, the inconsistency can often be detected much more efficiently than using the automata-theoretic method which is constructing the automaton for the formula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi$ and then checking for its emptiness. This problem manifests itself in many important settings, in which theories that describe system behavior use a large number of relatively simple constraints, such as database schemes or UML diagrams specified using, e.g., an appropriate description logic [11, 12, 5, 13].

In our work, we explore techniques that attempt to remedy the above difficulties by proposing an incremental and interleaved approach to constructing the automaton corresponding to the logical implication problem while simultaneously testing for satisfiability of the so far constructed fragments. The main contributions of this work are as follows:

- we show how the decision problem can be split into a sequence of simpler problems,
- we show that in this incremental process, the larger problems can be constructed from the simpler ones, hence avoiding unnecessary recomputation, and
- we show how top-down query evaluation techniques enhanced with memoing can be used to drive the incremental computation.

5.2 From APT to NBT via UCT

The standard approach for checking the emptiness of an alternating parity tree automaton (APT) involves Safra's construction [78] which is complicated and not very suitable for efficient implementation. An alternative approach to this problem has been proposed by Kupferman and Vardi [55] and involves the following steps:

1. Translate the APT A representing a μ-calculus formula φ to a Universal Co-Büchi Tree Automaton (UCT) A',
2. Translate the UCT A' to a Non-deterministic Büchi Tree Automaton (NBT) A'', and
3. Check for emptiness of A''.

The above transformations only preserve emptiness for the automata, not the actual languages of trees accepted. This is, however, sufficient for deciding satisfiability. We modify this procedure to operate in an incremental fashion when the original alternating automaton represents a conjunction of L_μ formulas. First, we review the two main steps in the original construction [55]:

5.2.1 From APT to UCT

Consider an APT $A = \langle \Sigma, D, Q, S, \delta, F \rangle$, where $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$. A *restriction* of δ is a partial function $\eta : Q \to 2^{D \times Q}$. A restriction η is relevant to $\sigma \in \Sigma$ if for all $q \in Q$ for which $\delta(q, \sigma)$ is satisfiable, the set $\eta(q)$ satisfies $\delta(q, \sigma)$. Let R be the set of restrictions of δ.

For $A = \langle \Sigma, D, Q, S, \delta, F \rangle$ with $S = \{q_i\}$, $F = \{F_1, F_2, \ldots, F_{2h}\}$, and $F_0 = \emptyset$, the UCT is defined as $A' = \langle \Sigma', D, Q \times \{0, \ldots, h-1\}, \{\langle q_i, 0 \rangle\}, \delta', F' \rangle$ where:

- $\Sigma' \subseteq \Sigma \times R$ such that η is relevant to σ for all $\langle \sigma, \eta \rangle \in \Sigma'$.
- For every $q \in Q$, $\sigma \in \Sigma$, and $\eta \in R$:
  - $\delta'(\langle q, 0 \rangle, \langle \sigma, \eta \rangle) = \bigwedge_{0 \le i < h} \bigwedge_{(c,s) \in (\eta(q) \setminus (D \times F_{2i}))} (c, \langle s, i \rangle)$.
  - For every $1 \le i < h$, $\delta'(\langle q, i \rangle, \langle \sigma, \eta \rangle) = \bigwedge_{(c,s) \in (\eta(q) \setminus (D \times F_{2i}))} (c, \langle s, i \rangle)$.
- $F' = \bigcup_{0 \le i < h} (F_{2i+1} \times \{i\})$

Intuitively, the nondeterminism in A is removed in A' since Σ' contains all the pairs $\langle \sigma, \eta \rangle$ for which η is relevant to σ (η chooses from all the possible sets of atoms that satisfy δ). The automaton A' consists of h copies of A such that the ith copy checks if a path in a run of A' visits F_{2i} only finitely often then it also visits F_{2i+1} only finitely often by making sure that the run stays in the ith copy unless it has to move to a state from F_{2i}.

5.2.2 From UCT to NBT

Let $A' = \langle \Sigma', D', Q', S', \delta', F' \rangle$, and let $k = (2n!)n^{2n}3^n(n+1)/n!$. Let $\mathcal{R}$ be the set of functions $f : Q' \to \{0, \dots k\}$ in which $f(q)$ is even for all $q \in F'$. For $g \in \mathcal{R}$, let $odd(g) = \{q : g(q) \text{ is odd}\}$. The definition of $A'' = \langle \Sigma', D', Q'', S'', \delta'', F'' \rangle$ is given as follows:

- $Q'' = 2^{Q'} \times 2^{Q'} \times \mathcal{R}$
- $S'' = \{\langle \{q'_i\}, \emptyset, g_0 \rangle\}$, where g_0 maps all states to k.
- For $q \in Q'$, $\sigma \in \Sigma'$, and $c \in D'$, let $\gamma'(q, \sigma, c) = \delta'(q, \sigma) \cap (\{c\} \times Q)$. For two functions g and g' in $\mathcal{R}$, a letter σ, and direction $c \in D'$, we say that g' *covers* $\langle g, \sigma, c \rangle$ if for all q and q' in Q', if $q' \in \gamma'(q, \sigma, c)$. then $g'(q') \leq g(q)$. Then for all $\langle S, O, g \rangle \in Q''$ and $\sigma \in \Sigma'$, δ'' is defined as follows:
  - If $O \neq \emptyset$ then $\delta''(\langle S, O, g \rangle, \sigma)$
  $$= \bigwedge_{c \in D} \bigvee_{g_c \ covers \ \langle g, \sigma, c \rangle} \langle \gamma'(S, \sigma, c), \gamma'(O, \sigma, c) \setminus odd(g_c), g_c \rangle$$
  - If $O = \emptyset$ then $\delta''(\langle S, O, g \rangle, \sigma)$
  $$= \bigwedge_{c \in D} \bigvee_{g_c \ covers \ \langle g, \sigma, c \rangle} \langle \gamma'(S, \sigma, c), \gamma'(S, \sigma, c) \setminus odd(g_c), g_c \rangle$$
- $F'' = 2^{Q'} \times \{\emptyset\} \times \mathcal{R}$.

Intuitively, the automaton A'' is the result of a subset construction applied to A' such that for a run of A' that satisfies a particular co-Büchi condition it guesses the possible runs that satisfy its dual Büchi condition. The emptiness problem for NBT is much simpler than the emptiness problem for APT which is shown to be solved symbolically in quadratic time [58].

5.3 Decomposition of the APT to NBT Translation

In this section, we describe the proposed decomposition technique for a conjunction of formulas of the form $\varphi = \varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$. We know that there is an APT $A = \langle \Sigma, D, Q, S, \delta, F \rangle$ that accepts tree models of φ. To define this automaton, we need the following auxiliary definition:

Definition 5.1 The closure of a formula ϕ, $\mathsf{cl}(\phi)$ is the smallest set of formulas that satisfies the following:

- $\phi \in \mathsf{cl}(\phi)$.
- If $\phi_1 \wedge \phi_2 \in \mathsf{cl}(\phi)$, then $\phi_1 \in \mathsf{cl}(\phi)$ and $\phi_2 \in \mathsf{cl}(\phi)$.
- If $[a]\psi$ or $\neg\psi \in \mathsf{cl}(\phi)$ then $\psi \in \mathsf{cl}(\phi)$.
- If $\nu z.\psi \in \mathsf{cl}(\phi)$, then $\psi(\nu z.\psi) \in \mathsf{cl}(\phi)$ and $\psi \in \mathsf{cl}(\phi)$.

Now we define an alternating automaton $A_k = \langle \Sigma_k, D, Q_k, S_k, \delta_k, F_k \rangle$ for a subformula $\varphi' = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$ of φ as follows:

- $\Sigma_k = 2^{AP_k}$ where AP_k is the set of atomic propositions in φ',
- $S_k = \{\varphi'\}$,
- $Q_k = \mathsf{cl}(\varphi')$,
- for all $\sigma \in \Sigma_k$, $\delta(q, \sigma) \in \delta_k$ iff $q \in Q_k$, and
- $F_k = \{F_1 \cap Q_k, F_2 \cap Q_k, \ldots, F_{2h} \cap Q_k\}$.

Emptiness of A_k implies emptiness of A and, in turn, the unsatisfiability of the original formula φ, as A_k represents a subformula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$ of φ. Hence, we can stop checking for the emptiness of the automaton A early: whenever we reach an automaton A_k that is empty. Otherwise we use the following theorem to *extend* A_k to A_{k+1} without the need to recompute all the transitions from scratch:

Theorem 5.2 *Let $A'_k = \langle \Sigma'_k, D, Q_k \times \{0, \ldots h-1\}, \{\langle q_{i_k}, 0\rangle\}, \delta'_k, F'_k \rangle$ be the UCT translation of A_k, and $A' = \langle \Sigma', D, Q \times \{0, \ldots h-1\}, \{\langle q_i, 0\rangle\}, \delta', F' \rangle$ be the UCT translation of A. Then for every $q_k \in Q_k$, $\sigma_k \in \Sigma_k$, $\eta_k \in R_k$, $\eta_l \in R_l$:*
$\delta'(\langle q_k, i\rangle, \langle \sigma_k, \eta_k \cup \eta_l\rangle) = \delta'_k(\langle q_k, i\rangle, \langle \sigma_k, \eta_k\rangle)$ for all $0 \le i < h$ where R_k is the set of restrictions $\eta_k : Q_k \to 2^{D \times Q_k}$ such that for all $\langle \sigma_k, \eta_k \rangle \in \Sigma'_k$, η_k is relevant to σ_k, and R_l is the set of restrictions $\eta_l : Q \setminus Q_k \to 2^{D \times Q}$.

<u>Proof:</u> For every $q_k \in Q_k$, $\sigma_k \in \Sigma_k$, $\eta_k \in R_k$, $\eta_l \in R_l$:

- $$\begin{aligned}\delta'(\langle q_k,0\rangle,\langle\sigma_k,\eta_k\cup\eta_l\rangle) &= \textstyle\bigwedge_{0\le i<h}\bigwedge_{(c,s)\in(\eta_k(q_k)\cup\eta_l(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\\ &= \textstyle\bigwedge_{0\le i<h}\bigwedge_{(c,s)\in(\eta_k(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\wedge\\ &\qquad \textstyle\bigwedge_{0\le i<h}\bigwedge_{(c,s)\in(\eta_l(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\\ &= \textstyle\bigwedge_{0\le i<h}\bigwedge_{(c,s)\in(\eta_k(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\\ &= \textstyle\bigwedge_{0\le i<h}\bigwedge_{(c,s)\in(\eta_k(q_k)\setminus(D\times(F_{2i}\cap Q_k)))}(c,\langle s,i\rangle)\\ &= \delta'_k(\langle q_k,0\rangle,\langle\sigma_k,\eta_k\rangle),\text{ and}\end{aligned}$$
- for all $1\le i<h$ we have
$$\begin{aligned}\delta'(\langle q_k,i\rangle,\langle\sigma_k,\eta_k\cup\eta_l\rangle) &= \textstyle\bigwedge_{(c,s)\in(\eta_k(q_k)\cup\eta_l(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\\ &= \textstyle\bigwedge_{(c,s)\in(\eta_k(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\wedge\\ &\qquad \textstyle\bigwedge_{(c,s)\in(\eta_l(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\\ &= \textstyle\bigwedge_{(c,s)\in(\eta_k(q_k)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)\\ &= \textstyle\bigwedge_{(c,s)\in(\eta_k(q_k)\setminus(D\times(F_{2i}\cap Q_k)))}(c,\langle s,i\rangle)\\ &= \delta'_k(\langle q_k,i\rangle,\langle\sigma_k,\eta_k\rangle).\end{aligned}$$

□

Thus we can reuse the transitions computed for a UCT A'_k (i.e., for $\varphi_1\wedge\varphi_2\wedge\ldots\wedge\varphi_k$) when computing the transitions of A'_{k+1} for $\varphi_1\wedge\varphi_2\wedge\ldots\wedge\varphi_{k+1}$. Similar theorem holds for the UCT to NBT step:

Theorem 5.3 *Let $A''_k=\langle\Sigma'_k,D,Q''_k,S''_k,\delta''_k,F''_k\rangle$ be NBT translation of A'_k, and $A''=\langle\Sigma',D,Q'',S'',\delta'',F''\rangle$ be the NBT translation of A'.*
Then for all $\langle S,O,g\rangle\in Q''_k$, $\sigma'=\langle\sigma_k,\eta_k\rangle\in\Sigma'_k$, $\sigma=\langle\sigma_k,\eta_k\cup\eta_l\rangle\in\Sigma'$, $\delta''(\langle S,O,g\cup f\rangle,\sigma)=\delta''_k(\langle S,O,[g\cup f/g]\rangle,\sigma')$ where $g:Q'_k\to\{0,\ldots,k'\}$ ($k'=(2n_k!)n_k^{2n_k}3^{n_k}(n_k+1)/n_k!$ where n_k is the number of states in A'_k), and $f:Q'\setminus Q'_k\to\{0,\ldots,k\}$.

Proof: If $O\neq\emptyset$ then

$$\begin{aligned}&\delta''(\langle S,O,g\cup f\rangle,\sigma)\\ &\quad= \bigwedge_{c\in D}\ \bigvee_{\substack{g_c\,covers\ \langle g,\sigma,c\rangle\\ f_c\,covers\ \langle f,\sigma,c\rangle}}\langle\gamma'(S,\sigma,c),\gamma'(O,\sigma,c)\setminus odd(g_c\cup f_c),g_c\cup f_c\rangle\\ &\quad= \bigwedge_{c\in D}\ \bigvee_{\substack{g_c\ covers\ \langle g,\sigma,c\rangle\\ f_c\ covers\ \langle f,\sigma,c\rangle}}\langle\gamma'_k(S,\sigma',c),\gamma'_k(O,\sigma',c)\setminus odd(g_c),g_c\cup f_c\rangle\\ &\quad= \delta''_k(\langle S,O,[g\cup f/g]\rangle,\sigma')\end{aligned}$$

If $O = \emptyset$ then

$$\begin{aligned}
&\delta''(\langle S, O, g \cup f\rangle, \sigma) \\
&\quad = \bigwedge_{c\in D} \bigvee_{\substack{g_c \, covers \, \langle g,\sigma,c\rangle \\ f_c \, covers \, \langle f,\sigma,c\rangle}} \langle \gamma'(S,\sigma,c), \gamma'(S,\sigma,c) \setminus odd(g_c \cup f_c), g_c \cup f_c\rangle \\
&\quad = \bigwedge_{c\in D} \bigvee_{\substack{g_c \, covers \, \langle g,\sigma,c\rangle \\ f_c \, covers \, \langle f,\sigma,c\rangle}} \langle \gamma'_k(S,\sigma',c), \gamma'_k(S,\sigma',c) \setminus odd(g_c), g_c \cup f_c\rangle \\
&\quad = \delta''_k(\langle S, O, [g \cup f/g]\rangle, \sigma')
\end{aligned}$$

□

This result shows that we can reuse the transitions we compute for an NBT A''_k used for checking the satisfiability of $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$ when we are computing the transitions of A''_{k+1} for $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_{k+1}$.

Example 5.4 Consider a formula $\varphi = \varphi_1 \wedge \varphi_2$ such that $\varphi_1 = \nu x.(\psi \wedge \langle - \rangle x)$ $\varphi_2 = \neg \nu x.(\psi \wedge \langle - \rangle x)$ where $\psi = \mu y.(b \vee \langle - \rangle y)$. Let $\Sigma = \{a, b\}$, and $D = \{1, 2\}$, an APT accepting models (which are tree models that have at least one path with infinitely many b's) of φ_1 is $A_1 = \{\Sigma, D, Q_1, \{q_1\}, \delta_1, F_1\}$ where:

$Q_1 = \{q_0, q_1\}$
$\delta_1(q_0, a) = (1, q_0) \vee (2, q_0)$
$\delta_1(q_0, b) = (1, q_1) \vee (2, q_1)$
$\delta_1(q_1, a) = (1, q_0) \vee (2, q_0)$
$\delta_1(q_1, b) = (1, q_1) \vee (2, q_1)$
$F_1 = \{\{q_0\}, \{q_0, q_1\}, \{q_0, q_1\}, \{q_0, q_1\}\}$

APT for φ_2, $A_2 = \{\Sigma, D, Q_2, \{q_2\}, \delta_2, F_2\}$:

$Q_2 = \{q_2, q_3\}$
$\delta_2(q_2, a) = (1, q_2) \wedge (2, q_2)$
$\delta_2(q_2, b) = (1, q_3) \wedge (2, q_3)$
$\delta_2(q_3, a) = (1, q_2) \wedge (2, q_2)$
$\delta_2(q_3, b) = (1, q_3) \wedge (2, q_3)$
$F_2 = \{\{\}, \{q_2\}, \{q_2, q_3\}, \{q_2, q_3\}\}$

and the APT for φ is $A_3 = \{\Sigma, D, Q_3, \{q_4\}, \delta_3, F_3\}$:

$Q_3 = Q_1 \cup Q_2 \cup \{q_4\}$
$\delta_3 = \delta_1 \cup \delta_2$ plus the following transitions:

$\delta_3(q_4, a) = (1, q_0) \wedge (1, q_2)$
$\delta_3(q_4, b) = (1, q_0) \wedge (1, q_2)$
$F_3 = \{\{q_0\}, \{q_0, q_1, q_2\}, \{q_0, q_1, q_2, q_3\}, \{q_0, q_1, q_2, q_3\}\}$

Note that the index of A_3 is 4 and A_1 and A_2 have the same index as A_3 according to the definition of F_k (for k=1 and k=2 in this case). As a result some sets in F_1 and F_2 are repeated at the end.

The incremental strategy used for this formula first checks for the emptiness of A_1 (which is not empty), then checks for the emptiness of A_3 (while re-using the transitions computed for the UCT A'_1 and the NBT A''_1), e.g.: $\delta'_3(\langle q_0, 0\rangle, \langle a, \eta_1 \cup \eta_2\rangle) = \delta'_1(\langle q_0, 0\rangle, \langle a, \eta_1\rangle)$ for all $\eta_1 \in R_1$ and $\eta_2 \in R_2$. Here R_1 is the set of restrictions $\eta_1 : Q_1 \to 2^{D\times Q_1}$ such that for all $\langle \sigma_1, \eta_1\rangle \in \Sigma'_1$, η_1 is relevant to σ_1, and R_2 is the set of restrictions $\eta_2 : Q_3 \setminus Q_1 \to 2^{D\times Q_3}$.

5.4 The Algorithm

Let A_i be the APT for $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_i$, and let A'_i be the UCT translation of A_i, $A''_i[j]$ be the NBT translation of A'_i where $\mathcal{R}$ is the set of functions $f : Q'_i \to \{0, \ldots, j\}$ for $1 \leq i \leq n$.

The algorithm outlined in Figure 5.1 incrementally constructs automata $A''_i[j]$ for $1 \leq i \leq n$ and looks for the smallest j, $j_m \leq k$ such that $A''_i[j]$ is not empty reusing the automaton $A''_i[j]$ in the computation of $A''_i[j+1]$. If $A''_i[k]$ is empty it stops, if not it constructs $A''_{i+1}[j_m]$ reusing the automaton $A''_i[j_m]$. Hence we have two directions first we are checking for the emptiness of a particular automaton $A''_i[j]$ for $1 \leq j \leq k$, second we are checking for the emptiness of automata $A''_i[j]$ for $1 \leq i \leq n$. We are using the proposed incremental technique on computing automata reusing the previous automata in both directions.

Theorem 5.5 *If $A''_i[k']$ is not empty then $A''_{i-1}[k']$ is also not empty where $1 \leq k' \leq (2n!)n^{2n}3^n(n+1)/n!$ and n is the number of states in A'_i.*

<u>Proof:</u> Let $A'_i = \langle \Sigma, D, Q, \{\langle q_i, 0\rangle\}, \delta, F\rangle$, and $A'_{i-1} = \langle \Sigma_1, D, Q_1, \{\langle q_{i_1}, 0\rangle\}, \delta_1, F_1\rangle$. Starting state of A''_i is $q''_i = \langle \{\langle q_i, 0\rangle\}, \emptyset, g_0\rangle$, and starting state of A''_{i-1} is $q''_{i_1} = \langle \{\langle q_{i_1}, 0\rangle\}, \emptyset, g^1_0\rangle$ where $\mathcal{R}$ is the set of functions $f : Q \to \{0, \ldots k'\}$ and $g_0 \in \mathcal{R}$ and $g^1_0 \in \mathcal{R}$ map all the states in Q and Q_1 to k' respectively. Consider a path π where q''_i goes to $\langle S, O, g_c\rangle$. If we remove all the states $Q_2 = Q \setminus Q_1$ from π then we get a path π_2 where q''_i goes to $\langle S_2, O_2, g^1_c\rangle$ such that $\langle q_{i_1}, 0\rangle \in S_2$. Consider a path π_1 where q''_{i_1} goes to $\langle S_1, O_1, g^1_c\rangle$, if π_2 is accepting then π_1 is also accepting since $\{\langle q_{i_1}, 0\rangle\} \subseteq S_2$. □

```
1:  initial = 1
2:  for $i = 1$ to $n$ do
3:      construct $A_i$
4:      if $i > 1$ then
5:          construct $A'_i$ using $A'_{i-1}$
6:      end if
7:      $k = (2n!)n^{2n}3^n(n+1)/n!$ for $A'_i$ with $n$ states
8:      for $j = initial$ to $k - 1$ do
9:          construct $A''_i[j]$
10:         if $A''_i[j]$ is not empty then
11:             if $i = n$ then
12:                 return not empty
13:             else
14:                 initial = j
15:                 go to 2
16:             end if
17:         end if
18:     end for
19:     if $A''_i[k]$ is empty then
20:         return empty
21:     end if
22: end for
```

Figure 5.1: Pseudo-code for Incremental Satisfaction Algorithm.

This theorem shows that the smallest j such that $A''_{i-1}[j]$ is not empty is also the smallest possible j such that $A''_i[j]$ is not empty. As a consequence, when we are constructing $A''_i[j]$ we can start from the *last* j. Also, this means we can directly reuse the information computed at stage $i-1$.

5.5 A Top-down Approach to the APT to NBT Translation

We represent the general construction algorithm as a logic program and check the emptiness using a goal with respect to the program. The outline of the program for the construction of an NBT A''_β from an APT $A_\beta = \langle \Sigma, D, Q, S, \delta, F \rangle$ for a formula β is as follows:

5.5.1 Representation of APT

The representation for alternating parity tree automata A_β with an index h for a formula β is a Datalog$^{\text{cv}}$ program P_β that defines the following predicates:

1. $\mathsf{Dir}_\beta(n)$ representing the set of directions D,
2. $\mathsf{Node}_\beta(n)$ representing the set of nodes Q of A,
3. $\mathsf{Start}_\beta(n)$ representing the set of starting states S,
4. $\mathsf{Final}_\beta(i, N)$ representing the acceptance condition F where $0 \leq i < h$ and N is a set of nodes such that if we have $\mathsf{Final}_\beta(j, N_1)$ and $\mathsf{Final}_\beta(j+1, N_2)$ then $N_1 \subseteq N_2$. F is a finite set, hence the number for atomic rules defining Final_β is also finite.
5. $\mathsf{Trans}_\beta(n, N, a)$ representing the transition function δ where n is a node, N is a set of direction, node pairs and a is a letter from the alphabet.

5.5.2 Preprocessing of the Transition Relation

We preprocess the transition function δ of A_β to transform the $\mathcal{B}^+$ formulas to sets of nodes using the algorithm given in Figure 5.2. The procedure *process* called by the algorithm code is given in Figure 5.3.

5.5.3 APT to UCT Translation

Suppose P_β defines the APT A_β, we define the program P'_β defining its UCT translation A'_β. First we represent the set of restrictions and the *relevant* relation by the following program R'_β.

```
$index = 0$
for all $\delta(q, a) = \varphi$ do
    process$(q, a, \varphi)$
end for
```

Figure 5.2: Pseudo-code for preprocessing of δ

```
if $\varphi = (c_1, q_1) \wedge (c_2, q_2) \wedge \ldots \wedge (c_m, q_m)$ then
    add $\mathsf{Trans}_\beta(q, \{(c_1, q_1), (c_2, q_2), \ldots, (c_m, q_m)\}, a)$
else
    if $\varphi = (c_1, q_1)$ then
        add $\mathsf{Trans}_\beta(q, \{(c_1, q_1)\}, a)$
    else
        if $\varphi = \varphi_1 \wedge \varphi_2$ then
            add $\mathsf{Trans}_\beta(q, \{Q_{index}, Q_{index+1}\}, a)$
            process$(Q_{index}, a, \varphi_1)$
            process$(Q_{index+1}, a, \varphi_2)$
            $index = index + 2$
        else
            if $\varphi = \varphi_1 \vee \varphi_2$ then
                process$(q, a, \varphi_1)$
                process$(q, a, \varphi_2)$
            end if
        end if
    end if
end if
```

Figure 5.3: Pseudo-code for process(q, a, φ)

Definition 5.6 The program R'_β consists of the following clauses.

1. $\mathsf{DN\_Set}'_\beta(\langle(d,q)\rangle) \leftarrow \mathsf{Dir}_\beta(d), \mathsf{Node}_\beta(q)$
2. $\mathsf{Restrict\_El}'_\beta(q,S) \leftarrow \mathsf{Node}_\beta(q), \mathsf{DN\_Set}'_\beta(N), S \subseteq N$
   $\mathsf{Restrict}'_\beta(\langle(q,S)\rangle) \leftarrow \mathsf{Restrict\_El}'_\beta(q,S)$
   $\mathsf{Two\_S}'_\beta(N) \leftarrow (q,S_1) \in N, (q,S_2) \in N, S_1 \neq S_2$
   $\mathsf{Restrict\_Fn}'_\beta(M) \leftarrow \mathsf{Restrict}'_\beta(N), M \subseteq N, \neg\mathsf{Two\_S}'_\beta(M)$
3. $\mathsf{Not\_Relavant}'_\beta(R,a) \leftarrow \mathsf{Trans}_\beta(n,S_1,a)$, $\mathsf{Restrict\_Fn}'_\beta(R)$, $(n,S_2) \in R, \neg(S_1 \subseteq S_2)$
   $\mathsf{Relevant}'_\beta(R,a) \leftarrow \neg\mathsf{Not\_Relavant}'_\beta(R,a)$

Lemma 5.7 *If P_β represents A_β then R'_β represents the set of restrictions R and the relevant relation between the letters of the alphabet and the restrictions.*

Proof: From the definitions in section 5.2.1 a *restriction* of δ is a partial function $\eta : Q \rightarrow 2^{D\times Q}$. A restriction η is relevant to $\sigma \in \Sigma$ if for all $q \in Q$ for which $\delta(q,\sigma)$ is satisfiable, the set $\eta(q)$ satisfies $\delta(q,\sigma)$. In R'_β, $\mathsf{DN\_Set}'_\beta$ constructs the set $D \times Q$ since Dir_β represents D, and Node_β represents Q. The predicate $\mathsf{Restrict}'_\beta$ represents $Q \times 2^{D\times Q}$, and $\mathsf{Restrict\_Fn}'_\beta$ represents a restriction $\eta : Q \rightarrow 2^{D\times Q}$. The predicate $\mathsf{Relevant}'_\beta$ represents the relevant relation since for all $(n,S_1,a) \in \mathsf{Trans}_\beta$, $R \in \mathsf{Restrict\_Fn}'_\beta$, $(n,S_2) \in R, S_1 \subseteq S_2$ which means S_2 satisfies Trans_β for (n,S_1,a). □

The following program I'_β computes the interval $1 \leq i < h$ and $(i,2i)$ pairs for $0 \leq i < h$.

Definition 5.8 The program I'_β consists of the following clauses.

1. $\mathsf{Index}'_\beta(0) \leftarrow$
   $\mathsf{Index}'_\beta(1) \leftarrow$
   ...
   $\mathsf{Index}'_\beta(h\_val - 1) \leftarrow$
2. $\mathsf{Index\_Pair}'_\beta(0,0) \leftarrow$
   $\mathsf{Index\_Pair}'_\beta(1,2) \leftarrow$
   ...
   $\mathsf{Index\_Pair}'_\beta(h\_val - 2/2, h\_val - 2) \leftarrow$ (if $h\_val - 2$ is even)
   $\mathsf{Index\_Pair}'_\beta(h\_val - 1/2, h\_val - 1) \leftarrow$ (if $h\_val - 1$ is even)

Here, *h_val* is a constant and its value is the value of h. We can use arithmetic to compactly represent the finite sets of atomic rules given in the first and second parts of the definition. The value of *h_val* is given so we can also use the following rules to define the predicate Index'_β.

$\mathsf{Index}'_\beta(1) \leftarrow$
$\mathsf{Index}'_\beta(n) \leftarrow \mathsf{Index}'_\beta(m), n = m+1, n < h\_val$

$\mathsf{Index\_Pair}'_\beta(0,0) \leftarrow$
$\mathsf{Index\_Pair}'_\beta(n,m) \leftarrow \mathsf{Index}'_\beta(n), m = 2*m$

Since the set of values satisfying Index'_β is finite then the set of values satisfying $\mathsf{Index\_Pair}'_\beta$ which uses Index'_β in its definition is also finite.

Definition 5.9 The following program P'_β consists of the following clauses added to the program $P_\beta \cup R'_\beta \cup I'_\beta$.

1. $\mathsf{Node}'_\beta(n,i) \leftarrow \mathsf{Node}_\beta(n), \mathsf{Index}'_\beta(h), i = h-1$
2. $\mathsf{Start}'_\beta(n,0) \leftarrow \mathsf{Start}_\beta(n)$
3. $\mathsf{Dir\_Final}'_\beta(\langle(t,(c,s))\rangle) \leftarrow \mathsf{Dir}_\beta(c), \mathsf{Final}_\beta(t,S), s \in S$
   $\mathsf{Trans}'_\beta((n,0), \langle(c,(s,i))\rangle, (a,r)) \leftarrow \mathsf{Index\_Pair}'_\beta(i,t),\ \mathsf{Relevant}'_\beta(r,a),\ (n,m) \in r,$
   $(c,s) \in m,\ \neg\mathsf{Dir\_Final}'_\beta(t,(c,s))$
   $\mathsf{Trans}'_\beta((n,h), \langle(c,(s,h))\rangle, (a,r)) \leftarrow \mathsf{Index}'_\beta(h),\ \mathsf{Index\_Pair}'_\beta(h,t),\ \mathsf{Relevant}'_\beta(r,a),$
   $(n,m) \in r,\ (c,s) \in m,\ \neg\mathsf{Dir\_Final}'_\beta(t,(c,s))$
4. $\mathsf{Final}'_\beta(\langle(n,i)\rangle) \leftarrow \mathsf{Index\_Pair}'_\beta(i,t),\ l = t+1,\ \mathsf{Final}_\beta(l,s),\ n \in s$

Lemma 5.10 *If P_β defines the APT A_β and R'_β represents the set of restrictions R and the relevant relation between the letters of the alphabet and the restrictions then P'_β represents A'_β.*

Proof: Node'_β represents $Q \times \{0,\ldots,h-1\}$ since Node_β represents Q and Index'_β represents the interval $\{0,\ldots,h-1\}$. Start'_β represents $\{\langle q_i, 0\rangle\}$ since Start_β represents $\{q_i\}$. Trans'_β represents δ': For every $q \in Q$, $\sigma \in \Sigma$, and $\eta \in R$:

- $\delta'(\langle q,0\rangle, \langle\sigma,\eta\rangle) = \bigwedge_{0 \le i < h} \bigwedge_{(c,s)\in(\eta(q)\setminus(D\times F_{2i}))}(c, \langle s,i\rangle)$.
- For every $1 \le i < h$, $\delta'(\langle q,i\rangle, \langle\sigma,\eta\rangle) = \bigwedge_{(c,s)\in(\eta(q)\setminus(D\times F_{2i}))}(c, \langle s,i\rangle)$.

In P'_β, $\mathsf{Index\_Pair}'_\beta$ represents $(i,2i)$ pairs for $0 \le i < h$. The conjunction $\mathsf{Relevant}'_\beta(r,a)$, $(n,m) \in r$, $(c,s) \in m$, $\neg\mathsf{Dir\_Final}'_\beta(t,(c,s))$ compute $\bigwedge_{(c,s)\in(\eta(q)\setminus(D\times F_{2i}))}(c,\langle s,i\rangle)$ as a set of conjuncts. Final'_β represents F', it is given that $F' = \bigcup_{0\le i<h}(F_{2i+1} \times \{i\})$, $(i,t) \in \mathsf{Index\_Pair}'_\beta$, $l = t+1$ represent a pair $(i, 2i+1)$ and $(l,s) \in \mathsf{Final}_\beta$, $n \in s$ represent F_{2i+1}.
□

5.5.4 APT to UCT Decomposition

Let $\beta = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$, $\beta_k = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$, and $A'_k = \langle \Sigma'_k, D, Q_k \times \{0,\ldots,h-1\}, \langle\{q_{i_k}\},0\rangle, \delta'_k, F'_k\rangle$ be the UCT translation of A_{β_k}, $A' = \langle \Sigma', D, Q \times \{0,\ldots,h-1\}, \{\langle q_i, 0\rangle\}, \delta', F'\rangle$ be the UCT translation of A_β, and P_{β_k} represents A_{β_k}.

Definition 5.11 The program RD'_{β_k} consists of the following clauses.

1. $\mathsf{DN\_Set}'_\beta(\langle(d,q)\rangle) \leftarrow \mathsf{Dir}_\beta(d), \mathsf{Node}_\beta(q)$
2. $\mathsf{Restrict\_El}'_{\beta_k}(q,S) \leftarrow \mathsf{Node}_\beta(q), \neg\mathsf{Node}_{\beta_k}(q), \mathsf{DN\_Set}'_\beta(N), S \subseteq N$
   $\mathsf{Restrict\_I}'_{\beta_k}(\langle(q,S)\rangle) \leftarrow \mathsf{Restrict\_Set}'_{\beta_k}(q,S)$
   $\mathsf{Two\_S}'_{\beta_k}(N) \leftarrow (q,S_1) \in N, (q,S_2) \in N, S_1 \neq S_2$
   $\mathsf{Restrict\_Fn}'_{\beta_k}(M) \leftarrow \mathsf{Restrict}'_{\beta_k}(N), M \subseteq N, \neg\mathsf{Two\_S}'_{\beta_k}(M)$
   $\mathsf{Restrict\_El}'_\beta(q,S) \leftarrow \mathsf{Node}_\beta(q), \mathsf{DN\_Set}'_\beta(N), S \subseteq N$
   $\mathsf{Restrict}'_\beta(\langle(q,S)\rangle) \leftarrow \mathsf{Restrict\_El}'_\beta(q,S)$
   $\mathsf{Two\_S}'_\beta(N) \leftarrow (q,S_1) \in N, (q,S_2) \in N, S_1 \neq S_2$
   $\mathsf{Restrict\_Fn}'_\beta(M) \leftarrow \mathsf{Restrict}'_\beta(N), M \subseteq N, \neg\mathsf{Two\_S}'_\beta(M)$
3. $\mathsf{Not\_Relavant}'_\beta(R,a) \leftarrow \mathsf{Trans}_\beta(n,S_1,a),\ \mathsf{Restrict\_Fn}'_\beta(R),\ (n,S_2) \in R, \neg(S_1 \subseteq S_2)$
   $\mathsf{Relevant}'_\beta(R,a) \leftarrow \neg\mathsf{Not\_Relavant}'_\beta(R,a)$
   $\mathsf{Not\_Relavant}'_{\beta_k}(R,a) \leftarrow \mathsf{Trans}_{\beta_k}(n,S_1,a),\ \mathsf{Restrict\_Fn}'_{\beta_k}(R),\ (n,S_2) \in R, \neg(S_1 \subseteq S_2)$
   $\mathsf{Relevant}'_{\beta_k}(R,a) \leftarrow \neg\mathsf{Not\_Relavant}'_{\beta_k}(R,a)$

Definition 5.12 The following program PD'_β consists of the following clauses added to the program $P_\beta \cup RD'_{\beta_k} \cup (P'_{\beta_k} \setminus R'_{\beta_k}) \cup P'_\beta$.

1. For every $q_k \in \mathsf{Node}'_{\beta_k}$, $(a_k, r_k) \in \mathsf{Relevant}'_{\beta_k}$, $r_l \in \mathsf{Restrict\_I}'_{\beta_k}$:
   $\mathsf{TransD}'_\beta((q_k,h),t,(a_k, r_k \cup r_l)) \leftarrow \mathsf{Trans}'_{\beta_k}((q_k,h),t,(a_k,r_k))$
2. For every $q_l \in \mathsf{Node}'_\beta - \mathsf{Node}'_{\beta_k}$, $(a,r) \in \mathsf{Relevant}'_\beta$:
   $\mathsf{TransD}'_\beta((q_l,h),t,(a,r)) \leftarrow \mathsf{Trans}'_\beta((q_l,h),t,(a,r))$

If the incremental approach is used the tuples computed for $\mathsf{Trans}'_{\beta_k}$ when checking the emptiness of A'_{β_k} can be reused to compute a part of TransD'_β which represents the (decomposed) transition relation of A'_β.

5.5.5 UCT to NBT Translation

Suppose P'_β defines the UCT A'_β, we define the program P''_β defining its NBT translation A''_β. First we represent the interval $0 \leq i < k$, the sets of even and odd numbers in the interval. We then represent the set of functions $\mathcal{R}$, the functions g_0 and *odd* defined in section 5.2.2 by the following program R''_β.

Definition 5.13 The program R''_β consists of the following clauses.

1. $\mathsf{Number\_k}''_\beta(0) \leftarrow$
   $\mathsf{Number\_k}''_\beta(1) \leftarrow$
   ...
   $\mathsf{Number\_k}''_\beta(k\_val - 1) \leftarrow$
   $\mathsf{Even\_k}''_\beta(0) \leftarrow$
   $\mathsf{Even\_k}''_\beta(2) \leftarrow$
   ...
   $\mathsf{Even\_k}''_\beta(k\_val - 2) \leftarrow$ (if $k - 2$ is even)
   $\mathsf{Even\_k}''_\beta(k\_val - 1) \leftarrow$ (if $k - 1$ is even)
   $\mathsf{Odd\_k}''_\beta(1) \leftarrow$
   $\mathsf{Odd\_k}''_\beta(3) \leftarrow$
   ...
   $\mathsf{Odd\_k}''_\beta(k\_val - 2) \leftarrow$ (if $k - 2$ is odd)
   $\mathsf{Odd\_k}''_\beta(k\_val - 1) \leftarrow$ (if $k - 1$ is odd)
2. $\mathsf{Two\_k}''_\beta(N) \leftarrow (q, k_1) \in N, (q, k_2) \in N, k_1 \neq k_2$
   $\mathsf{R\_Rel}''_\beta(\langle(q, k)\rangle) \leftarrow \mathsf{Node}'_\beta(q), \mathsf{Number\_k}(k)$
   $\mathsf{R\_Function}''_\beta(N) \leftarrow \mathsf{R\_Rel}''_\beta(M), \mathsf{Final}'_\beta(S), N \subseteq M, \neg\mathsf{Two\_k}''_\beta(N)$
   $((q, k) \in N, (q, k) \in S \rightarrow \mathsf{Even\_k}''_\beta(k))$
3. $\mathsf{g\_Function}''_\beta(\langle(q, k\_val)\rangle) \leftarrow \mathsf{Node}'_\beta(q)$
4. $\mathsf{Odd}''_\beta(G, \langle Q\rangle) \leftarrow \mathsf{R\_Function}''_\beta(G), (Q, X) \in G, \mathsf{Odd\_k}''_\beta(X)$

The value of k_val is given so we can also use the following rules to define the predicates $\mathsf{Number\_k}''_\beta$, $\mathsf{Even\_k}''_\beta$, and $\mathsf{Odd\_k}''_\beta$.

$\mathsf{Number\_k}''_\beta(0)$
$\mathsf{Number\_k}''_\beta(N) \leftarrow \mathsf{Number\_k}''_\beta(M), N = M + 1, N < k\_val$
$\mathsf{Even\_k}''_\beta(0)$
$\mathsf{Even\_k}''_\beta(N) \leftarrow \mathsf{Even\_k}''_\beta(M), N = M + 2, N < k\_val$
$\mathsf{Odd\_k}''_\beta(1)$
$\mathsf{Odd\_k}''_\beta(N) \leftarrow \mathsf{Odd\_k}''_\beta(M), N = M + 2, N < k\_val$

Lemma 5.14 *Suppose P'_β defines the UCT A'_β then the predicate* $\mathsf{R\_Function}''_\beta$ *in R''_β represents the set of functions $\mathcal{R}$.*

Proof: $\mathcal{R}$ is the set of functions $f : Q \rightarrow \{0, \ldots, k\}$ in which $f(q)$ is even for all $q \in F$. In R''_β, $\mathsf{R\_Rel}''_\beta$ represents $Q \times [k]$. Also Node'_β represents Q, $\mathsf{Number\_k}''_\beta$ represents $[k]$, and

$\mathsf{R\_Function}''_\beta$ picks a set N from $2^{Q\times[k]}$ such that N contains tuples (q,k) where $q \in \mathsf{Node}'_\beta$, $k \in \mathsf{Number\_k}''_\beta$ and if $(q,k_1) \in N$ and $(q,k_2) \in N$ then $k_1 \neq k_2$, hence N represents a partial function $f : Q \to \{0,\ldots,k\}$. Further if $S \in \mathsf{Final}'_\beta$ and $(q,k) \in N$ and $(q,k) \in S$ then $k \in \mathsf{Even\_k}''_\beta$ which means if (q,k) represents an element from F then $f(q)$ is even. □

Lemma 5.15 *If P'_β defines the UCT A'_β then the predicate* $\mathsf{g\_Function}''_\beta$ *in R''_β represents g_0.*

Proof: The predicate $\mathsf{g\_Function}''_\beta$ is defined as a set with tuples (q, k_val), for all $q \in \mathsf{Node}'_\beta$ since k_val contains the value of k defined in the UCT to NBT translation in section 5.2.2, $\mathsf{g\_Function}''_\beta$ represents g_0 which maps all nodes to k. □

Lemma 5.16 *Let the predicate* $\mathsf{R\_Function}''_\beta$ *in R''_β represent the set of functions $\mathcal{R}$ then the predicate* Odd''_β *in R''_β represents the function Odd.*

Proof: The predicate Odd''_β contains tuples $G, \langle Q\rangle$ such that $G \in \mathsf{R\_Function}''_\beta$, $(Q,X) \in G$ and $X \in \mathsf{Odd\_k}''_\beta$ hence the set represented by $\langle Q\rangle$ contains all the nodes q such that $(q,X) \in G$ where X is odd. As a result the predicate Odd''_β in R''_β represents the function *Odd* where $\langle Q\rangle$ represents $Odd(g)$ and G represents g given in the definition of *Odd*. □

We now represent the *covers* relation defined in section 5.2.2 by the following program C''_β.

Definition 5.17 The program C'_β consists of the following clauses.

1. $\mathsf{Not\_Covers}''_\beta(G_1,(G_2,a,c)) \leftarrow \mathsf{R\_Function}''_\beta(G_1), \mathsf{R\_Function}''_\beta(G_2), \mathsf{Node}'_\beta(q_1,d_1),$
   $\mathsf{Node}'_\beta(q_2,d_2), \mathsf{Trans}'_\beta((q_1,d_1),(q_2,d_2),(a,c)),$
   $((q_1,d_1),x) \in G_1, ((q_2,d_2),y) \in G_2, y > x$
2. $\mathsf{Covers}''_\beta(G_1,(G_2,a,c)) \leftarrow \mathsf{R\_Function}''_\beta(G_1), \mathsf{R\_Function}''_\beta(G_2), \mathsf{Trans}'_\beta(q,t,(a,c)),$
   $\neg\mathsf{Not\_Covers}''_\beta(G_1,(G_2,a,c))$

Lemma 5.18 *Let P'_β defines the A'_β and the predicate* $\mathsf{R\_Function}''_\beta$ *in R''_β represents the set of functions $\mathcal{R}$ then the predicate* Covers''_β *in C''_β represents the covers relation.*

Proof: The predicate $\mathsf{Not\_Covers}''_\beta$ defines the tuples $(G_1,(G_2,a,c))$ such that $G_1 \in \mathsf{R\_Function}''_\beta$, $G_2 \in \mathsf{R\_Function}''_\beta$, there is a transition $((q_1,d_1),(q_2,d_2),(a,c)) \in \mathsf{Trans}'_\beta$, $((q_1,d_1),x) \in G_1$, $((q_2,d_2),y) \in G_2$ where $y > x$. Then $\mathsf{Not\_Covers}''_\beta$ contains $(G_1,(G_2,a,c))$ pairs such that G_1 does not cover (G_2,a,c) according to the definition of *covers*. The predicate Covers''_β defines the tuples $(G_1,(G_2,a,c))$ such that $G_1 \in \mathsf{R\_Function}''_\beta$, $G_2 \in \mathsf{R\_Function}''_\beta$, $(q,t,(a,c)) \in \mathsf{Trans}'_\beta$ such that $(G_1,(G_2,a,c)) \notin \mathsf{Not\_Covers}''_\beta$, which means

G_1 covers (G_2, a, c) according to the definition of *covers*. Hence the predicate Covers''_β in C''_β represents the *covers* relation where G_1 represents g', (G_2, a, c) represents $\langle g, \sigma, c\rangle$. $\square$

Definition 5.19 The program P''_β consists of the following clauses added to the program $P''_\beta \cup R''_\beta \cup C''_\beta$.

1. $\mathsf{Node\_Set}'_\beta(\langle(n,h)\rangle) \leftarrow \mathsf{Node}'_\beta(n,h)$
   $\mathsf{Node}''_\beta(n_1, n_2, r) \leftarrow \mathsf{Node\_Set}'_\beta(n), n_1 \subseteq n, n_2 \subseteq n, \mathsf{R\_Function}''_\beta(r_1)$
2. $\mathsf{Start}''_\beta(\{(n,h)\}, \{\}, r) \leftarrow \mathsf{Start}'_\beta(n,h), \mathsf{g\_Function}''_\beta(r)$
3. $\mathsf{Trans\_Set}''_\beta(n, \langle s\rangle, (a,c)) \leftarrow \mathsf{Trans}'_\beta(n_1, s, (a,c)), n_1 \in n$
   $\mathsf{Trans}''_\beta((s,o,g),(t_1,t_3,g_c),(a,c)) \leftarrow \mathsf{Dir}''_\beta(c),\ o \neq \emptyset,\ \mathsf{Node}''_\beta(s,o,g),\ \mathsf{Covers}''_\beta(g_c,(g,a,c)),$
   $\mathsf{Trans\_Set}''_\beta(s,t_1,(a,c)),\ \mathsf{Trans\_Set}''_\beta(o,t_2,(a,c)),\ \mathsf{Odd}''_\beta(g_c,m),\ t_3 = t_2 - m$
   $\mathsf{Trans}''_\beta((s,o,g),(t_1,t_3,g_c),(a,c)) \leftarrow \mathsf{Dir}''_\beta(c),\ o = \emptyset,\ \mathsf{Node}''_\beta(s,o,g),\ \mathsf{Covers}''_\beta(g_c,(g,a,c)),$
   $\mathsf{Trans\_Set}''_\beta(s,t_1,(a,c)),\ \mathsf{Trans\_Set}''_\beta(s,t_2,(a,c)),\ \mathsf{Odd}''_\beta(g_c,m),\ t_3 = t_2 - m$
4. $\mathsf{Final}''_\beta((n,\{\},r)) \leftarrow \mathsf{Node\_Set}'_\beta(m), n \subseteq m, \mathsf{R\_Function}''_\beta(r)$

Lemma 5.20 *Let P'_β defines the A'_β, the predicate $\mathsf{R\_Function}''_\beta$ in R''_β represents the set of functions $\mathcal{R}$ and the predicate Covers''_β in C''_β represents the covers relation then P''_β represents A''_β.*

Proof: Node''_β represents Q'' since n_1, and n_2 represent two subsets of Q, and $r_1 \in \mathsf{R\_Function}''_\beta$ represents a function from $\mathcal{R}$. Start''_β represents $S'' = \{\langle\{q'_i\}, \emptyset, g_0\rangle\}$ since Start'_β represents $\{q'_i\}$ and $r \in \mathsf{g\_Function}''_\beta$ represents g_0. For $q \in Q'$, $\sigma \in \Sigma'$, and $c \in D'$, let $\gamma'(q,\sigma,c) = \delta'(q,\sigma) \cap (\{c\} \times Q)$. For two functions g and g' in $\mathcal{R}$, a letter σ, and direction $c \in D'$, we say that g' *covers* $\langle g, \sigma, c\rangle$ if for all q and q' in Q', if $q' \in \gamma'(q,\sigma,c)$. then $g'(q') \leq g(q)$. Then for all $\langle S, O, g\rangle \in Q''$ and $\sigma \in \Sigma'$, δ'' is defined as follows:

- If $O \neq \emptyset$ then $\delta''(\langle S, O, g\rangle, \sigma)$

$$= \bigwedge_{c \in D} \bigvee_{g_c \ covers \ \langle g,\sigma,c\rangle} \langle \gamma'(S,\sigma,c), \gamma'(O,\sigma,c) \setminus odd(g_c), g_c\rangle$$

- If $O = \emptyset$ then $\delta''(\langle S, O, g\rangle, \sigma)$

$$= \bigwedge_{c \in D} \bigvee_{g_c \ covers \ \langle g,\sigma,c\rangle} \langle \gamma'(S,\sigma,c), \gamma'(S,\sigma,c) \setminus odd(g_c), g_c\rangle$$

Trans''_β represents δ'' where the term c represents a direction c, $\mathsf{Trans\_Set}''_\beta$ represents γ', the term s represents S, o represents O, Covers''_β represents the *covers* relation, and Odd''_β represents the *Odd* function. Final''_β represents F'' since n represents a subset of Q' and $r_1 \in \mathsf{R\_Function}''_\beta$ represents a function from $\mathcal{R}$. □

5.5.6 UCT to NBT Decomposition

Let $A''_{\beta_k} = \langle \Sigma'_k, D, Q''_k, S''_k, \delta''_k, F''_k \rangle$ be NBT translation of A'_{β_k}, and $A''_\beta = \langle \Sigma', D, Q'', S'', \delta'', F'' \rangle$ be the NBT translation of A'_β, and P''_{β_k} represents A''_{β_k}.

Definition 5.21 The program PD''_{β_k} consists of the following clauses to be added to $P''_{\beta_k} \cup P''_\beta$

1. For all $(s,o,g) \in \mathsf{Node}''_{\beta_k}$, $(a_k, r_k) \in \mathsf{Relevant}'_{\beta_k}$, $r_l \in \mathsf{Restrict\_I}'_{\beta_k}$, $g \in \mathsf{R\_Function}''_{\beta_k}$, $g \cup f \in \mathsf{R\_Function}''_\beta$:
   $\mathsf{TransD}''_\beta((s,o,g\cup f), t, (a, r_k \cup r_l)) \leftarrow \mathsf{Trans}''_{\beta_k}((s,o,g), t, (a, r_k))$
2. For all $(s,o,f) \in \mathsf{Node}''_\beta - \mathsf{Node}''_{\beta_k}$, $(a,r) \in \mathsf{Relevant}'_\beta$, $f \in \mathsf{R\_Function}''_\beta$:
   $\mathsf{TransD}''_\beta((s,o,f), t, (a,r)) \leftarrow \mathsf{Trans}''_\beta((s,o,f), t, (a,r))$

As in the case of APT to UCT decomposition the tuples computed for $\mathsf{Trans}''_{\beta_k}$ with the incremental approach can be reused to compute a part of TransD''_β which represents the (decomposed) transition relation of A''_β.

5.5.7 NBT Emptiness

Definition 5.22 Suppose P''_β defines the NBT A''_β, the program E''_β for deciding on the emptiness of A''_β on binary trees consists of the following clauses added to the program P''_β:

1. $\mathsf{TransB}''_\beta((s,o,g), a, n, m) \leftarrow \mathsf{Trans}''_\beta((s,o,g), n, (a,0)), \mathsf{Trans}''_\beta((s,o,g), m, (a,1))$
2. $\mathsf{Sub\_Tree}''_\beta(n_1, N) \leftarrow \mathsf{TransB}''_\beta(n_1, a, n_2, n_3), \mathsf{Final}''_\beta(n_2), \mathsf{Final}''_\beta(n_3), N = \{n_2\} \cup \{n_3\}$
   $\mathsf{Sub\_Tree}''_\beta(n_1, N) \leftarrow \mathsf{TransB}''_\beta(n_1, a, n_2, n_3), \mathsf{Final}''_\beta(n_2), \mathsf{Sub\_Tree}''_\beta(n_3, N_5),$
   $N = \{n_2\} \cup N_5$
   $\mathsf{Sub\_Tree}''_\beta(n_1, N) \leftarrow \mathsf{TransB}''_\beta(n_1, a, n_2, n_3), \mathsf{Final}''_\beta(n_3), \mathsf{Sub\_Tree}''_\beta(n_2, N_5),$
   $N = \{n_3\} \cup N_5$
   $\mathsf{Sub\_Tree}''_\beta(n_1, N) \leftarrow \mathsf{TransB}''_\beta(n_1, a, n_2, n_3), \mathsf{Sub\_Tree}''_\beta(n_2, N_4), \mathsf{Sub\_Tree}''_\beta(n_3, N_5),$
   $N = N_4 \cup N_5$
3. $\mathsf{GSub\_Tree}''_\beta(n_1, N_2) \leftarrow \mathsf{Start}''_\beta(n_1), \mathsf{Sub\_Tree}''_\beta(n_1, N_2)$
   $\mathsf{GSub\_Tree}''_\beta(s_1, N) \leftarrow \mathsf{GSub\_Tree}''_\beta(n_1, N_2), s_1 \in N_2\ \mathsf{Sub\_Tree}''_\beta(s_1, N)$

4. $\mathsf{Emp\_F}''_\beta(\langle n_1 \rangle) \leftarrow \mathsf{GSub\_Tree}''_\beta(n_1, N_2)$
   $\mathsf{Emp\_S}''_\beta(\langle N_2 \rangle) \leftarrow \mathsf{GSub\_Tree}''_\beta(n_1, N_2)$
   $\mathsf{Emp}''_\beta() \leftarrow \mathsf{Emp\_F}''_\beta(N_1),\ \mathsf{Emp\_S}''_\beta(N_2),\ M \in N_2, M \subseteq N_1$

The proposed NBT emptiness algorithm for a particular automaton checks if subtrees which have only final nodes in their leaves are repeated infinitely often. The emptiness query works top-down starting from the transitive closure of the initial state on these types of subtrees and stops checking when it makes certain that they are repeated infinitely often. This means that there is a tree accepted by the automaton. We compute only the transitions that we need to answer the emptiness query. For instance, to answer the emptiness query on an NBT automaton we only need to compute the transitions that are reachable from the starting state of the automaton.

Let $\beta = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$, then the following query answers the satisfiability question for β:
$\mathsf{Emp}''_{\varphi_1}()$, $\mathsf{Emp}''_{\varphi_1 \wedge \varphi_2}()$, …, $\mathsf{Emp}''_\beta()$

Example 5.23 Consider an NBT automaton A where

$\Sigma = \{a\}$, $D = \{1, 2\}$,
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $S = \{q_0\}$,
$\delta(q_0, a) = (1, q_1) \wedge (2, q_2)$,
$\delta(q_1, a) = (1, q_2) \wedge (2, q_3)$,
$\delta(q_2, a) = (1, q_1) \wedge (2, q_1)$,
$\delta(q_3, a) = (1, q_1) \wedge (2, q_3)$,
$\delta(q_4, a) = (1, q_5) \wedge (2, q_5)$,
$\delta(q_5, a) = (1, q_1) \wedge (2, q_1)$, and
$F = \{q_2, q_3\}$.

When we are running the emptiness algorithm on this automaton we only compute the first four transitions.

5.6 Heuristics

In this section, we provide several heuristics and optimizations that can be applied to the proposed technique. First, we explain the optimizations in translation of an APT A to a UCT A' which is an incremental technique on the alphabet we use for A'. Then we explain the optimizations in translation of a UCT A' to an NBT A'' which is an incremental technique on the size of the functions in $\mathcal{R}$ we use for A'' which is proposed in [55]. Finally, we describe the heuristics we can use for rewriting conjunctive formulas (i.e. reordering the subformulas in a conjunctive formula) so that we have a better chance for detecting possible contradictions faster.

5.6.1 Optimizations in APT to UCT Translation.

First we introduce an optimization used in the translation of APT to UCT. Since $\Sigma' \subseteq \Sigma \times R$ we can start the construction using a subset Σ'_1 of Σ'. We proceed with a larger subset, Σ'_2, if the satisfiability query is empty, and repeat enlarging the alphabet until either the query becomes non-empty or we reach to the set Σ'. We are also able to reuse the results in the next computation since $\Sigma'_1 \subseteq \Sigma'_2$.

Theorem 5.24 *Let $A'_1 = \langle \Sigma'_1, D, Q, S, \delta'_1, F \rangle$ and $A'_2 = \langle \Sigma'_2, D, Q, S, \delta'_2, F \rangle$ are UCT translations of an APT A using Σ'_1 as alphabet of A'_1 and using Σ'_2 as alphabet of A'_2. If $\Sigma'_1 \subseteq \Sigma'_2$, then $\delta'_1 \subseteq \delta'_2$.*

Proof: Since we define $\delta'_2(\langle q, i \rangle, \langle \sigma_2, \eta_2 \rangle)$ for every $q \in Q$, $\sigma_2 \in \Sigma_2$, $\eta_2 \in R_2$, and for all $0 \leq i < h$ where R_2 is the set of restrictions such that for all $\langle \sigma_2, \eta_2 \rangle \in \Sigma'_2$, η_2 is relevant to σ_2 the same way as $\delta'_1(\langle q, i \rangle, \langle \sigma_1, \eta_1 \rangle)$ for every $q \in Q$, $\sigma_1 \in \Sigma_1$, $\eta_1 \in R_1$, and for all $0 \leq i < h$ where R_1 is the set of restrictions such that for all $\langle \sigma_1, \eta_1 \rangle \in \Sigma'_1$, η_1 is relevant to σ_1 then if $\Sigma'_1 \subseteq \Sigma'_2$, $\delta'_1 \subseteq \delta'_2$. □

This incremental approach allows us to partition the alphabet Σ' to a sequence of sets:

$\Sigma'_1 \subseteq \Sigma'_2 \subseteq \ldots \subseteq \Sigma'$

which is especially useful when the alphabet is large such as the exponential alphabet of UCT in the size of the alphabet of APT after translation.

5.6.2 Optimizations in UCT to NBT Translation.

In the proposed translation of UCT to NBT we start from an initial value k_1 for k and increase this value up to k_2, as long as the satisfiability query is empty. We continue this process until either the automaton becomes non-empty or we reach the upper bound of $(2n!)n^{2n}3^n(n+1)/n!$ for n the number of states in the UCT automaton. This approach has been proposed in [55]. Our decomposition, however, allows an incremental implementation that reuses *the transitions* computed for k_1 in the subsequent construction for k_2.

Theorem 5.25 *Let $A''_1[k_1]$ and $A''_2[k_2]$ are NBT translations of an APT A, using k_1 as the maximum range of functions in $\mathcal{R}_1$ for A''_1 and k_2 as the maximum range of functions in $\mathcal{R}_2$ for A''_2. If $k_1 \leq k_2$, then $\delta''_1 \subseteq \delta''_2$.*

<u>**Proof:**</u> Since $\mathcal{R}_1$ is the set of functions $f_1 : Q' \to \{0, \dots, k_1\}$ and $\mathcal{R}_2$ is the set of functions $f_2 : Q' \to \{0, \dots, k_2\}$ and $k_1 \leq k_2$ then $\mathcal{R}_1 \subseteq \mathcal{R}_2$ which means $Q''_1 \subseteq Q''_2$. Thus $\delta''_1 \subseteq \delta''_2$. □

Example 5.26 Consider an alternating automaton A such that: $\Sigma = \{a\}$, $D = \{1, 2\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $S = \{q_0\}$, $\delta(q_0, a) = (1, q_1) \wedge (2, q_2)$, $\delta(q_1, a) = (1, q_3) \wedge (2, q_3)$, $\delta(q_2, a) = (1, q_3) \wedge (2, q_3)$, $\delta(q_3, a) = (1, q_3)$, and $F = \{\{\}, \{q_0, q_1, q_2, q_3\}\}$ We have calculated the actual number of transitions in the UCT translation of A, A' and the NBT translation of A, A'', and the number of transitions we need to answer the satisfiability query after we apply the above optimizations. The set of restrictions is R and the set of restrictions we used for answering the satisfiability query is R_1. The number of transitions computed for A' with R is 4×2^{32} and the number of transitions computed for A' with R_1 is 4. The results for the NBT translation are given in Figure 5.4 where $k = 2^{20} \cdot 42525$, and $k_1 = 1$.

# of transitions computed for $A''[k]$	$256 \cdot k^4$
# of transitions computed for $A''[k_1]$	256
# of transitions computed for $A''[k_1]$ with top-down evaluation	70

Figure 5.4: Number of transitions in the NBT automata $A''[k]$ and $A''[k_1]$.

5.6.3 Heuristics for Ordering of Conjunctive Formulas.

Consider a logical consequence question $\{\varphi_1, \varphi_2, \dots, \varphi_n\} \models \psi$, such that the formula ψ is already inconsistent with a subset of formulas in $\{\varphi_1, \varphi_2, \dots, \varphi_n\}$. As we use an incremental technique we can use rewriting heuristics to generate a formula $\neg\psi \wedge \varphi_{i_1} \wedge \varphi_{i_2} \wedge \dots \wedge \varphi_{i_n}$ such that $[i_1, i_2, \dots i_n]$ is a permutation of $[1, 2, \dots n]$. For instance, the formulas $\varphi_1, \varphi_2, \dots, \varphi_n$ can be ordered according to the number of free variables they share with ψ. Hence we improve our chances of finding a possible contradiction faster if we use this formula instead of the original one in the proposed algorithm. The following examples demonstrate the effect of ordering of the subformulas of a conjunctive formula.

Example 5.27 Consider a formula $\psi = \varphi \wedge \varphi_4$ where φ is the formula given in Example 5.4, $\varphi_4 = \nu x.(\psi \wedge \langle - \rangle x)$ such that $\psi = \mu y.(a \vee \langle - \rangle y)$, $\Sigma = \{a, b\}$, and $D = \{1, 2\}$, an APT for φ_4 is $A_4 = \{\Sigma, D, Q_4, \{q_5\}, \delta_4, F_4\}$ where:

$Q_4 = \{q_5, q_6\}$
$\delta_4(q_5, a) = (1, q_5) \vee (2, q_5)$

$\delta_4(q_5, b) = (1, q_6) \vee (2, q_6)$
$\delta_4(q_6, a) = (1, q_5) \vee (2, q_5)$
$\delta_4(q_6, b) = (1, q_6) \vee (2, q_6)$
$F_4 = \{\{q_6\}, \{q_5, q_6\}, \{q_5, q_6\}, \{q_5, q_6\}\}$

APT for ψ, $A_5 = \{D, \Sigma, Q_5, \{q_7\}, \delta_5, F_5\}$:

$Q_5 = Q_3 \cup Q_4 \cup \{q_7\}$
$\delta_5 = \delta_3 \cup \delta_4$ plus the following transitions:
$\delta_5(q_7, a) = (1, q_4) \wedge (1, q_5)$
$\delta_5(q_7, b) = (1, q_4) \wedge (1, q_5)$
$F_5 = \{\{q_0, q_6\}, \{q_0, q_1, q_2, q_5, q_6\}, \{q_0, q_1, q_2, q_3, q_5, q_6\}, \{q_0, q_1, q_2, q_3, q_5, q_6\}\}$

Using the proposed strategy we first check whether A_1 defined in Example 5.4 is empty (it is not empty), then we check the emptiness of A_3 which is empty and thus we do not need to construct A'_5 and A''_5. The estimated number of transitions is 10×2^{50} for A'_3, and 16×2^{128} for A'_5. The estimated number of transitions for A''_3 and A''_5 are given in Figure 5.5 where $k_3 = 20! \cdot 10^{20} \cdot 3^{10} \cdot 11/10!$, $k_5 = 32! \cdot 2^{128} \cdot 3^{16} \cdot 17/16!$.

estimated # of transitions for A''_3	$2 \times 2^{10} \times 2^{10} \times k_3^{10}$
estimated # of transitions for A''_5	$2 \times 2^{16} \times 2^{16} \times k_5^{16}$

Figure 5.5: Number of transitions in the NBT automata A''_3 and A''_5.

Example 5.28 Consider a logical consequence problem $\{\varphi_2, \varphi_3, \varphi_4, \varphi_5\} \models \varphi_1$ where φ_1 and φ_2 are given in Example 5.4, $\varphi_3 = \nu x.(\psi_1 \wedge \langle - \rangle x)$ such that $\psi_1 = \mu y.(a \vee \langle - \rangle y)$, $\Sigma_3 = \{a, b\}$, $\varphi_4 = \nu x.(\psi_2 \wedge \langle - \rangle x)$ such that $\psi_2 = \mu y.(c \vee \langle - \rangle y)$, $\Sigma_4 = \{c, b\}$, $\varphi_5 = \nu x.(\psi_3 \wedge \langle - \rangle x)$ such that $\psi_3 = \mu y.(d \vee \langle - \rangle y)$, $\Sigma_5 = \{d, b\}$, and $D = \{1, 2\}$, an APT for φ_3 is $A_3 = \{\Sigma, D, Q_3, \{q_5\}, \delta_3, F_3\}$ where:

$Q_3 = \{q_5, q_6\}$
$\delta_3(q_5, a) = (1, q_5) \vee (2, q_5)$
$\delta_3(q_5, b) = (1, q_6) \vee (2, q_6)$
$\delta_3(q_6, a) = (1, q_5) \vee (2, q_5)$
$\delta_3(q_6, b) = (1, q_6) \vee (2, q_6)$
$F_3 = \{\{q_6\}, \{q_5, q_6\}, \{q_5, q_6\}, \{q_5, q_6\}\}$

The APT A_4 for φ_4 and the APT A_5 for φ_5 are the same as A_3 except that the state names are changed and the letter a is replaced with c in A_4 and d in A_5, respectively.

Using the proposed strategy we first check if A_1 defined in Example 5.4 is empty (it is not empty), then we check the emptiness of the intersection automaton $A_{1,2}$ of A_1 and A_2 which is empty. Hence, we do not need to construct the complete intersection automaton A for A_1, A_2, A_3, A_4, and A_5. The estimated number of transitions for $A''_{1,2}$ and A'' are given in Figure 5.6 where $k_1 = 20! \cdot 10^{20} \cdot 3^{10} \cdot 11/10!$, $k_2 = 56! \cdot 28^{56} \cdot 3^{28} \cdot 29/28!$.

estimated # of transitions for $A''_{1,2}$	$2 \times 2^{10} \times 2^{10} \times k_1^{10}$
estimated # of transitions for A''	$5 \times 2^{28} \times 2^{28} \times k_2^{28}$

Figure 5.6: Number of transitions in the NBT automata $A''_{1,2}$ and A''.

Bibliographical Notes

Safra [78] described an optimal determinization construction for automata on infinite words. Alternating automata on infinite trees is introduced by [26, 63]. Safra's construction is shown to be resistant to efficient implementation [86]. The contruction used in this work which does not use Safra's contruction is introduced in [55]. An extension of Safraless decision algorithm that is amenable to implementation was proposed for LTL formulas [52] which also improved the complexity of the algorithm.

Chapter 6

Conclusions and Future Work

In our work, we introduced a translation technique that maps satisfiability questions for formulas in WSnS and S1S to query answering in Datalog$^{\text{cv}}$ and developed an incremental approach to an automata-based decision procedure for μ-calculus. We have also demonstrated how evaluation techniques used for answering queries on these programs can provide efficient decision procedures for second order logics. For developing decision procedures for WSnS and S1S using logic-automata connection we provide a Datalog$^{\text{cv}}$ representation of automata and automata-theoretic operations. We represent the automaton for a particular formula as Datalog$^{\text{cv}}$ rules and satisfiability of the formula as a Datalog$^{\text{cv}}$ query. The basic idea of our method is to represent our formulas in such a way that we can decide on them efficiently using the available techniques to construct and search only the part of the state space needed to answer the satisfiability queries on formulas. In our work, we classify formulas as $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$, $\varphi = \neg(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$, $\varphi = \neg(\exists \overline{x} : \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$, $\varphi = \exists \overline{x} : \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$, and propose heuristics and optimizations depending on the type of the formula. We study the impact of goal reordering and various other query optimization techniques on the performance of the decision procedures we introduce. The types of formulas we consider for the decision procedures we propose are given in Figure 6.1. Our results for WSnS show that our technique outperforms tools implemented for the same purpose for various types of formulas.

Future extensions of the proposed approach include extending the translation we propose for WSnS and S1S to other types of automata on infinite objects, e.g., to Rabin [73] and Alternating Automata [98], and on improving the upper complexity bounds by restricting the form of Datalog$^{\text{cv}}$ programs generated by the translation (when used for decision problems in, e.g., EXPTIME). In all these cases, the goal is to match the optimal theoretical bounds while avoiding the worst-case behavior (inherent in most automata-based techniques) in as many situations as possible.

	WSnS	S1S	μ-calculus
$\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$	✓	✓	✓
$\varphi = \neg(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$	✓	✓	
$\varphi = \neg(\exists\overline{x} : \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$	✓		
$\varphi = \exists\overline{x} : \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$	✓		

Figure 6.1: Summary of our results

Future research for the incremental technique we present for μ-calculus will follow several directions:

1. we attempt to reduce the part of the automaton needed to show satisfiability (or unsatisfiability) by introducing additional heuristics in the incremental construction,

2. for particular classes of problems, for which other techniques exhibit better performance due to reduced search space, we attempt to modify the proposed incremental approach to mimic those approaches,

3. we study how the proposed incremental technique can take advantage of the structure of problems formulated in more restricted formalisms such as description logics, and

4. we adopt our decomposition technique for alternating parity automata emptiness to alternating looping automata.

Bibliography

[1] S. Abiteboul and C. Beeri. The power of languages for the manipulation of complex values. *VLDB Journal*, 4(4):727–794, 1995.

[2] F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69:5–40, 2001.

[3] C. Beeri, S. Naqvi, O. Shmueli, and S. Tsur. Set construction in a logic database language. *Journal of Logic Programming*, 10(3&4):181–232, 1991.

[4] C. Beeri and R. Ramakrishnan. On the power of Magic. *Journal of Logic Programming*, 10(1/2/3&4):255–299, 1991.

[5] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML Class Diagrams using Description Logic Based Systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-44*, 2001.

[6] M. Biehl, N. Klarlund, and T. Rauhe. Algorithms for guided tree automata. In *First International Workshop on Implementing Automata, WIA '96, London, Ontario, Canada*, volume 1260 of *LNCS*. Springer Verlag, 1997.

[7] J. Bradfield and C. Stirling. *Modal Mu-Calculi*, chapter 12. Elsevier Science, 2006.

[8] R. E. Bryant. Symbolic boolean manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[9] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.

[10] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. 1960 Int. Congr. for Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

[11] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 84–89, 1999.

[12] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Description logics: Foundations for class-based knowledge representation. In *Proc. of the 17th IEEE Sym. on Logic in Computer Science (LICS 2002)*, pages 359–370, 2002.

[13] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer, 1998.

[14] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS*, pages 34–43, 1998.

[15] W. Chen, M. Kifer, and D. S. Warren. Hilog: A foundation for higher-order logic programming. *J. Log. Program.*, 15(3):187–230, 1993.

[16] W. Chen, T. Swift, and D. S. Warren. Efficient implementation of general logical queries. Technical report, SUNY at Stony Brook, 1993.

[17] W. Chen and D.S Warren. Query evaluation under the well-founded semantics. *PODS*, pages 168–179, 1993.

[18] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. A. Naqvi, S. Tsur, and C. Zaniolo. The LDL system prototype. *IEEE Trans. Knowl. Data Eng.*, 2(1):76–90, 1990.

[19] E. Clarke, D. Grumberg, and D. Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[20] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[21] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Computer-Aided Verification, Proc. 11th Int. Conference*, volume 1633, pages 249–260, July 1999.

[22] S. Dawson, C. R. Ramakrishnan, Steven Skiena, and Terrance Swift. Principles and practice of unification factoring. *TOPLAS*, 18(5):528–563, 1996.

[23] S. Demri and U. Sattler. Automata-theoretic decision procedures for information logics. *Fundam. Inform.*, 53(1):1–22, 2002.

[24] C. S. Jutla E. A. Emerson. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 368–377, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[25] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[26] E. A. Emerson. Automata, tableaux and temporal logics (extended abstract). In *Proceedings Conference on Logics of Programs, Brooklyn*, volume 193 of *LNCS*, pages 79–87. Springer-Verlag, 1985.

[27] E. A. Emerson. Temporal and modal logic. In *J. Van Leeuwen, editor, Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072, Cambridge, MA, USA, 1990. MIT Press.

[28] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 169–181. Springer-Verlag, 1980.

[29] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, FOCS'88, White Plains*, pages 328–337. IEEE Computer Society Press, Los Alamitos, CA, October 1988.

[30] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, 1991.

[31] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001.

[32] E. A. Emerson and A. P. Sistla. Deciding full branching time logics. *Information and Control*, 61(3):175–201, 1984.

[33] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[34] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Computer Aided Verification, Proc. 13th Int. Conference*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.

[35] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, Warsaw, October 1988.

[36] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. Springer, 2002.

[37] S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Vardi. On complementing nondeterministic Büchi automata, 2003.

[38] D. Harel. Dynamic logic. *In D. Gabbay and F. Guenthner (eds.), Handbook of Philosophical Logic, Volume II, Dordrecht: D. Reidel*, pages 497–604, 1984.

[39] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* Cambridge, MA: MIT Press, 2000.

[40] J. G. Henriksen, J. L. Jensen, M. E. Jörgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. MONA: Monadic second-order logic in practice. In *TACAS*, volume 1019 of *LNCS*, pages 89–110, 1995.

[41] J. Hladik and U. Sattler. A Translation of Looping Alternating Automata into Description Logics. In *International Conference on Automated Deduction (CADE 19)*, volume 2741 of *LNCS*, pages 90–105, 2003.

[42] P. Øhrstrøm and P. Hasle. *Temporal Logic: From Ancient Ideas to Artificial Intelligence.* Boston and London: Kluwer Academic Publishers, 1995.

[43] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *MFCS*, volume 969 of *LNCS*, pages 552–562, London, UK, 1995. Springer-Verlag.

[44] J. Jard and T. Jeron. On-line model-checking for finite temporal logic specifications. In *Automatic Verification Methods for Finite State Systems, Proc. Int. Workshop, Grenoble*, volume 407, pages 189–196, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[45] R. Kaivola. *Using Automata to Characterise Fixed Point Temporal Logics.* PhD thesis, University of Edinburgh, 1997.

[46] M. Kifer and G. Lausen. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. *SIGMOD Rec.*, 18(2):134–146, 1989.

[47] N. Klarlund. MONA & FIDO: The logic-automaton connection in practice. In *Computer Science Logic*, volume 1414 of *LNCS*, pages 311–326, London, UK, 1997. Springer-Verlag.

[48] N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. *Int. J. Found. Comput. Sci.*, 13(4):571–586, 2002.

[49] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[50] D. Kozen and R. Parikh. An elementary proof of the completeness of PDL. *Theoretical Computer Science*, 14:113–118, 1981.

[51] G. M. Kuper and M. Y. Vardi. The logical data model. *ACM Transactions On Database Systems*, 18:86–96, 1993.

[52] O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *CAV*, volume 4144 of *LNCS*, pages 31–44, 2006.

[53] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. In *Proceedings of the Fifth Israel Symposium on Theory of Computing and Systems, ISTCS'97*, pages 147–158, Los Alamitos, California, 1997. IEEE Computer Society Press.

[54] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of ACM*, 47(2):312–360, 2000.

[55] O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 531–540, Pittsburgh, October 2005.

[56] M. Liu. Relationlog: A typed extension to datalog with sets and tuples. *Journal of Logic Programming*, 36(3):271–299, 1998.

[57] M. Liu. Query processing in Relationlog. In *DEXA*, pages 342–351, 1999.

[58] P. Wolper M. Y. Vardi. Automata theoretic techniques for modal logics of programs: (extended abstract). In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 446–456, New York, NY, USA, 1984. ACM Press.

[59] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[60] A. R. Meyer. Weak monadic second order theory of successor is not elementary-recursive. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1973.

[61] M. Mukund. Linear-Time Temporal Logic and Büchi Automata. In *Tutorial talk, Winter School on Logic and Computer Science, ISI, Calcutta*, 1997.

[62] D. E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Symposium on Logic in Computer Science (LICS '88)*, pages 422–427, Washington, D.C., USA, July 1988. IEEE Computer Society Press.

[63] D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2-3):267–276, 1987.

[64] I. S. Mumick. *Query Optimization in Deductive and Relational Databases.* PhD thesis, Department of Computer Science, Stanford University, 1991.

[65] S. Naqvi and S. Tsur. *A Logical Language for Data and Knowledge Bases.* Computer Science Press, 1989.

[66] D. Niwinski. Fixed points vs. infinite generation. In *Symposium on Logic in Computer Science (LICS '88)*, pages 402–409, Washington, D.C., USA, July 1988. IEEE Computer Society Press.

[67] D. Park. Finiteness is μ-ineffable. *Theoretical Computer Science*, 3(2):173–181, 1976.

[68] V. R. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20:231–254, 1980.

[69] V. R. Pratt. A decidable μ-calculus: preliminary report. In *Proc. 29th IEEE Symposium on Foundation of Computer Science*, pages 421–427, 1981.

[70] A. N. Prior. *Papers on Time and Tense.* Oxford: Clarendon Press, 1957.

[71] A. N. Prior. *Past, Present and Future.* Oxford: Clarendon Press, 1957.

[72] A. N. Prior. *Time and Modality.* Oxford: Clarendon Press, 1957.

[73] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[74] I. V. Ramakrishnan, P. Rao, K. F. Sagonas, T. Swift, and D. S. Warren. Efficient tabling mechanisms for logic programs. In *International Conference on Logic Programming*, pages 697–711, 1995.

[75] R. Ramakrishnan, P. Bothner, D. Srivastava, and S. Sudarshan. CORAL - a database programming language. In *Workshop on Deductive Databases*, 1990.

[76] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL - Control, Relations and Logic. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pages 238–250, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[77] R. Ramakrishnan, D. Srivastava, S. Sudarshan, and P. Seshadri. The CORAL deductive system. *VLDB Journal*, 3(2):161–210, 1994.

[78] S. Safra. On the Complexity of ω-Automata. In *FOCS*, pages 319–327, 1988.

[79] S. Safra. Exponential Determinization for omega-Automata with Strong-Fairness Acceptance Condition (Extended Abstract). In *STOC*, pages 275–282, 1992.

[80] K. F. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In *SIGMOD Conference*, pages 442–453, 1994.

[81] U. Sattler and M. Y. Vardi. The Hybrid μ-Calculus. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *LNCS*, pages 76–91, 2001.

[82] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Computer-Aided Verification, Proc. 12th Int. Conference*, volume 1633, pages 247–263, 2000.

[83] L.J. Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, MIT Lab for Computer Science, 1974.

[84] R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, June 1989.

[85] T. Swift and D. S. Warren. Analysis of slg-wam evaluation of definite programs. In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming*, pages 219–235, Cambridge, MA, USA, 1994. MIT Press.

[86] S. Tasiran, R. Hojati, and R. K. Brayton. Language containment of non-deterministic omega -automata. In *Conference on Correct Hardware Design and Verification Methods*, pages 261–277, 1995.

[87] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–81, 1968.

[88] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3. Springer-Verlag New York, Inc. New York, NY, USA, 1997.

[89] B. A Trakhtenbrot. Finite Automata and Monadic Second order Logic. *Siberian Math Journal*, 3:101–131, 1962. Russian; English translation in: AMS Transl. 59 (1966), 23-55.

[90] S. Tsur and C. Zaniolo. LDL: A Logic-Based Data Language. In *VLDB '86: Proceedings of the 12th International Conference on Very Large Data Bases*, pages 33–41, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[91] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1&2. Computer Science Press, 1989.

[92] G. Unel and D. Toman. Deciding weak monadic second-order logics using complex-value datalog. In *Proc. LPAR (Short Paper)*, 2005.

[93] M. H. van Emden and R. Kowalski. The Semantics of Predicate Logic as Programming Language. *Journal of ACM*, 23(4):733–743, 1976.

[94] M. Y. Vardi. A temporal fixpoint calculus. In *POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 250–259, New York, NY, USA, 1988. ACM.

[95] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.

[96] M. Y. Vardi. Alternating automata: Unifying truth and validity for temporal logics. In *CADE*, pages 191–206, 1997.

[97] M. Y. Vardi. What makes Modal Logic so Robustly Decidable. In *Descriptive Complexity and Finite Models*. American Mathematical Society, 1997.

[98] M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, volume 1443 of *LNCS*, pages 628–641, 1998.

[99] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs (extended abstract). In *Proceedings 16th Annual ACM Symp. on the Theory of Computing, STOC'84*, pages 446–456. ACM Press, New York, 1984.

[100] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of the First Symposium on Logic in Computer Science*, pages 322–331, 1986.

[101] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[102] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.

[103] P. Wolper. Constructing automata from temporal logic formulas: A tutorial. *European Educational Forum: School on Formal Methods and Performance Analysis*, pages 261–277, 2000.