Local Reasoning for Parameterized First Order Protocols

by

Rylo Ashmore

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

Waterloo, Ontario, Canada, 2019

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

First Order Logic (FOL) is a powerful reasoning tool for program verification. Recent work on Ivy shows that FOL is well suited for verification of parameterized distributed systems. However, specifying many natural objects, such as a ring topology, in FOL is unexpectedly inconvenient. We present a framework based on FOL for specifying distributed multi-process protocols in a process-local manner together with an implicit network topology. In the specification framework, we provide an auto-active analysis technique to reason about the protocols locally, in a process-modular way. Our goal is to mirror the way designers often describe and reason about protocols. By hiding the topology behind the FOL structure, we simplify the modelling, but complicate the reasoning. To deal with that, we use an oracle for the topology to develop a sound and relatively complete proof rule that reduces reasoning about the implicit topology back to pure FOL. This completely avoids the need to axiomatize the topology. Using the rule, we establish a property that reduces verification to a fixed number of processes bounded by the size of local neighbourhoods. We show how to use the framework on a few examples, including leader elections on rings and trees.

Acknowledgements

First, I offer enormous thanks to my professional collaborators. Most importantly, my co-supervisors, Arie Gurfinkel and Richard Trefler offered guidance and motivation to make this work as rigorous as possible. Without this, there is no way this work would have been completed. I also offer gratitude to Oded Padon and Sharon Shoham for helping understand how Ivy works, and reviewing some of my work within Ivy. I am further grateful to the readers of this thesis, Grant Weddell and Derek Rayside.

Second, I thank my parents, Linda and Gerry Ashmore, for their continuing support thoughout my studies. Their support was invaluable to continuing studies in service of this thesis.

Thirdly, I present my gratefulness for the friends that I have met while at Waterloo, as well as those who maintained strong relationships from my undergraduate experience. While all of them are deeply appreciated, I make a special note for Ben Roque and Steph McIntyre, who accompanied me on a break from school for a road trip.

I would further like to acknowledge the work of the many lecturers and professors I have served as a TA for while at Waterloo. I greatly value the opportunity to hold office hours and help teach other students, and it is quite interesting seeing how many different ways core concepts can be expressed and permuted.

Finally, I acknowledge that this thesis is funded in part by a Cheriton scholarship.

Table of Contents

List of Figures vii

1 Introduction 1

2 Background 4

2.1 FOL syntax and semantics. . . . 4

2.2 FOL modulo structures . . . 5

2.3 First Order Transition Systems. . . . 5

3 First Order Protocols 8

3.1 Network Topology . . . 8

3.2 Example: Red-Black Rings . . . 9

3.3 FOP Definition . . . 10

4 Local Reasoning 14

4.1 Verifying FO-Protocols using First Order Logic . . . 14

4.2 Soundness and Completeness . . . 17

4.2.1 Soundness. . . . 17

4.2.2 Small model property. . . . 18

4.2.3 Relative Completeness. . . . 18

5 Case Studies **21**
5.1 Example: Leader Election Ring Protocol 21
5.2 Leader Election Tree Protocol 23
5.2.1 Modelling of Tree Leader Election 24
5.2.2 Verification of Tree Leader Election 26
5.2.3 Sanity Checks 28

6 Conclusion **31**
6.1 Related Work 31
6.2 Conclusion 32

References **33**

APPENDICES **36**

A symbols **37**

B Ivy Code **42**
B.1 Red-Black Rings Example Protocol 42
B.2 Ring Leader Election 45
B.3 Tree Leader Election 48

C btw Ring **52**
C.1 Finite Models and Ring Axioms 52

List of Figures

1.1 A description of a unidirectional ring in FOL as presented by Ivy [20]. . . . 2

3.1 An example of a topology and a protocol. 9
3.2 A FO-protocol description of the system from Fig. 3.1. 12
3.3 An FOTS of the protocol in Fig. 3.2. 12

4.1 Characteristics $\chi_{Red}(\emptyset, q)$ and $\chi_{Red}(Mod(p), q)$ for the $\mathcal{RBR}$ topology. . . . 16
4.2 The verification conditions VC_{Red} for the red process invariant. 16

5.1 A model of the Leader Election protocol as a FO-protocol. 22
5.2 Local inductive invariant $Inv_{lead}(x, y, z)$ for Leader Election from Fig. 5.1. . 23
5.3 A leader election protocol over trees. We use $\exists p' \neq p''$ to denote there exist two distinct elements p', p'', and $\exists! p' \cdot$ to denote that there exists one unique element satisfying the property. 24
5.4 A pair of example states of the tree protocol. 25

Chapter 1

Introduction

Auto-active[10] and automated verification engines are now commonly used to analyze the behavior of safety- and system-critical multi-process distributed systems. Applying the analysis techniques early in the design cycle has the added advantage that any errors or bugs found are less costly to fix than if one waits until the system is deployed. Therefore, it is typical to seek a proof of safety for *parametric* designs, where the number of participating program components is not yet determined, but the inter-process communciation fits a given pattern, as is common in routing or communication protocols, and other distributed systems.

Recently, Ivy [20] has been introduced as a novel auto-active verification technique (in the style of Dafny [10]) for reasoning about parameterized systems. Ivy models protocols in First Order Logic (FOL). The verification conditions are compiled (with user help) to a decidable fragment of FOL, called Effectively Propositional Reasoning (EPR) [21]. Ivy is automatic in the sense that the verification engineer only provides an inductive invariant. Furthermore, unlike Dafny, it guarantees that the verification is never stuck inside the decision procedure (verification conditions are decidable).

In representing a protocol in Ivy, an engineer must formally specify the entire protocol, including the topology. For instance, in verifying the leader election on a unidirectional ring, Ivy requires an explicit axiomatization of the ring topology, as shown in Fig. 1.1. The predicate $btw(x, y, z)$ means that a process y is between processes x and z in the ring; similarly, $next(a, b)$ means that b is an immediate neighbour of a on the ring. All (finite) rings satisfy the axioms in Fig. 1.1. The converse is not true in general. For instance, take the rationals $\mathbb{Q}$ and let $btw(x, y, z)$ be defined as $x < y < z \vee y < z < x \vee z < x < y$. All axioms of btw are satisfied, but the only consistent interpretation of $next$ is an empty

$$
\begin{aligned}
\forall x, y, z \cdot btw(x, y, z) &\Rightarrow btw(y, z, x) \\
\forall w, x, y, z \cdot btw(w, x, y) \wedge btw(w, y, z) &\Rightarrow btw(w, x, z) \\
\forall w, x, y \cdot btw(w, x, y) &\Rightarrow \neg btw(w, y, x) \\
\forall w, x, y \cdot distinct(w, x, y) &\Rightarrow (btw(w, x, y) \vee btw(w, y, x)) \\
\forall a, b \cdot (next(a, b) &\iff \forall x \cdot x \neq a \wedge x \neq b \Rightarrow btw(a, b, x))
\end{aligned}
$$

Figure 1.1: A description of a unidirectional ring in FOL as presented by Ivy [20].

set. This satisfies all the axioms, but does not define a ring. For the axioms in Fig. 1.1, all *finite* models of *btw* and *next* describe rings, as shown in Appendix C. This is not an issue for Ivy, since infinite models do not need to be considered for EPR. Such reasoning is non-trivial and is a burden on the verification engineer[1]. As another example, we were not able to come up with an axiomatization of rings of alternating red and black nodes (shown in Fig. 3.1a) within EPR. In general, a complete axiomatization of the topology might be hard to construct.

In this paper, we propose to address this problem by specifying the topology independently of process behaviour. We present a framework which separates the two and provides a clean way to express the topology. We then specify our transitions locally, as this is a natural and common way to define protocols. Once these preliminaries are done, we provide a process-local proof rule to verify properties of the system. To generate the proof rule, we offload topological knowledge to an oracle that can answer questions about the topology. Finally, we prove various properties of the proof rule.

In summary, the thesis makes the following contributions. First, in Sec. 3, we show how to model protocols locally in FOL. This is an alternative to the global modelling used in Ivy. Second, in Sec. 4.1, we show a proof rule with verification conditions (VC) in FOL, which are often in EPR. When the VC is in EPR, this gives an engineer a mechanical check of inductiveness. This allows reasoning about topology without axiomatizing it. Third, in Sec. 4.2, we show that our proof rule (a) satisfies a small model property, and (b) is relatively complete. The first guarantees the verification can be done on small process domains; the second ensures that our proof rule is relatively expressive.

We illustrate our approach on a few examples. First, as a running example, motivated by [17], is a protocol on rings of alternating red and black nodes. These rings globally have

[1]The proof given is by hand, a full formal proof verified by machine would be an even greater burden.

only rotational symmetry, however, they have substantial local symmetry [12, 16, 17]. This symmetry consists of two equivalence classes, one of red nodes, and one of black nodes. Second, in Sec. 5.1, we consider a modified version of the leader election protocol from Ivy [20]. This is of particular interest, since the local symmetry of [12, 16, 17] has not been applied to leader election. We thus extend [12, 16, 17] by both allowing more symmetries and infinite-state systems. Finally, we outline and verify a leader electin protocol on undirected binary trees motivated by [6]

Chapter 2

Background

This section will establish the basic ideas and notation needed in the rest of this work. In particular, we define the language of First Order Logic in Section 2.1 and append it with structures in Section 2.2. Finally, in Section 2.3, we define transition systems, building up to a definition of first-order transition systems and their specifications.

2.1 FOL syntax and semantics.

We assume some familiarity with the standard concepts of many sorted First Order Logic (FOL). A signature Σ consists of sorted predicates, functions, and constants. Terms are variables, constants, or (recursively) k-ary functions applied to k other terms of the correct sort. For every k-ary predicate P and k terms $t_1, \ldots, t_k$ of the appropriate sort for P, the formula $P(t_1, \ldots, t_k)$ is an atomic well-formed formula (wff). The full set of wffs are the boolean combinations of wffs and quantified (universally or existentially) wffs. Namely, if ψ and φ are wffs, then so are $(\psi \wedge \varphi)$, $(\psi \vee \varphi)$,$(\neg\psi)$, $(\psi \Rightarrow \varphi)$,$(\psi \iff \varphi)$, $(\forall x \cdot \psi)$, and $(\exists x \cdot \psi)$. A parse tree places the most recently used connective or quantifier at top, and sub-formulae underneath. Suppose the wff ψ has a parse tree as a subgraph of φ's parse tree. Then we say ψ is in the scope of φ. A variable x in an atomic formula ψ is bound if ψ is in the scope of a quantifier of x. A variable not bound is free. A wff with no free variables is called a sentence. For convenience, we often drop unnecessary parenthesis, and use $\top$ to denote true and $\bot$ to denote false.

An FOL interpretation $\mathcal{I}$ over a domain D assigns every k-ary predicate P a sort-appropriate semantic interpretation $\mathcal{I}(P) : D^k \to \{T, F\}$; to every k-ary function f a sort-appropriate interpretation $\mathcal{I}(f) : D^k \to D$, and to every constant c an element $\mathcal{I}(c) \in D$.

Given an interpretation $\mathcal{I}$ and a sentence ψ, then either ψ is true in $\mathcal{I}$ (denoted, $\mathcal{I} \models \psi$), or ψ is false in $\mathcal{I}$ (denoted $\mathcal{I} \not\models \psi$). The definition of the models relation is defined on the structure of the formula as usual, for example, $\mathcal{I} \models (\varphi \wedge \psi)$ iff $\mathcal{I} \models \varphi$ and $\mathcal{I} \models \psi$. We write $\models \varphi$ if for every interpretation $\mathcal{I}$, $\mathcal{I} \models \varphi$.

We write $\mathcal{I}(\Sigma')$ to denote a restriction of an interpretation $\mathcal{I}$ to a signature $\Sigma' \subseteq \Sigma$. Given disjoint signatures Σ,Σ' and corresponding interpretations $\mathcal{I}$, $\mathcal{I}'$ over a fixed domain D, we define $\mathcal{I} \oplus \mathcal{I}'$ to be an interpretation of $\Sigma \cup \Sigma'$ over domain D defined such that $(\mathcal{I} \oplus \mathcal{I}')(t) = \mathcal{I}(t)$ if $t \in \Sigma$, and $(\mathcal{I} \oplus \mathcal{I}')(t) = \mathcal{I}'(t)$ if $t \in \Sigma'$. Given interpretation $\mathcal{I}$ and sub-domain $D' \subseteq D$ of sort S, where D' contains all constants of sort S, we let $\mathcal{I}(D')$ be the interpretation where the sort S is restricted to domain D', and all other sorts remain the same.

2.2 FOL modulo structures

We use an extension of FOL to describe structures, namely graphs. In this case, the signature Σ is extended with some pre-defined functions and predicates, and the interpretations are restricted to particular intended interpretations of these additions to the signature. We identify a structure class $\mathcal{C}$ with its signature $\Sigma^{\mathcal{C}}$ and an intended interpretation. We write $FOL^{\mathcal{C}}$ for First Order Logic over the structure class $\mathcal{C}$. Common examples are FOL over strings, FOL over trees, and other finite structures.

A structure $\mathcal{S} = (D, \mathcal{I})$ is an intended interpretation $\mathcal{I}$ for structural predicates/functions $\Sigma^{\mathcal{C}}$ over an intended domain D. A set of structures is denoted $\mathcal{C}$. The syntax of $FOL^{\mathcal{C}}$ is given by the syntax for FOL with signature $\Sigma \uplus \Sigma^{\mathcal{C}}$ (where Σ is an arbitrary disjoint signature). For semantics, any FOL interpretation $\mathcal{I}$ over domain D (with structure $(D, \mathcal{I}^{\mathcal{C}}) \in \mathcal{C}$) of signature Σ leads to an $FOL^{\mathcal{C}}$ interpretation $\mathcal{I} \oplus \mathcal{I}^{\mathcal{C}}$ of the signature $\Sigma \uplus \Sigma^{\mathcal{C}}$. We write $\models_{\mathcal{C}} \varphi$ iff every $FOL^{\mathcal{C}}$ interpretation $\mathcal{I}$ satisfies $\mathcal{I} \models \varphi$. For our purposes, we further introduce a process sort *Proc* and require the intended domain D to be exactly the set of *Proc*-sorted elements, so that we put our intended structure on the processes.

2.3 First Order Transition Systems.

We use First Order Transitions Systems from Ivy [20, 19]. While the original definition was restricted to the EPR fragment of FOL, we do not require this. A transition system is a tuple $Tr = (S, S_0, R)$, where S is a set of states, $S_0 \subseteq S$ is a set of initial states, and

$R \subseteq S \times S$ is a transition relation. A trace π is a (finite or infinite) sequence of states $\pi = s_0 \cdots s_i \cdots$ such that $s_0 \in S_0$ and for every $0 \leq i < |\pi|$, $(s_i, s_{i+1}) \in R$, where $|\pi|$ denotes the length of π, or ∞ if π is infinite. A transition system may be augmented with a set $B \subseteq S$ of *bad* states. The system is safe iff all traces contain no bad states. A set of states I is inductive iff $S_0 \subseteq I$ and if $s \in I$ and $(s, s') \in R$, then $s' \in I$. Showing the existence of an inductive set I that is disjoint from bad set B suffices to show a transition system is safe.

A First-Order Transition System Specification (FOTSS) is a tuple $(\Sigma, \varphi_0, \tau)$ where Σ is an FOL signature, φ_0 is a sentence over Σ and τ is a sentence over $\Sigma \uplus \Sigma'$, where $\uplus$ denotes disjoint union and $\Sigma' = \{t' \mid t \in \Sigma\}$. The semantics of a FOTSS are given by First Order Transition Systems (FOTS). Let D be a fixed domain. A FOTSS $(\Sigma, \varphi_0, \tau)$ defines a FOTS over D as follows: $S = \{\mathcal{I} \mid \mathcal{I} \text{ is an FOL interpretation over } D\}$, $S_0 = \{\mathcal{I} \in S \mid \mathcal{I} \models \varphi_0\}$, and $R = \{(\mathcal{I}_1, \mathcal{I}_2) \in S \times S \mid \mathcal{I}_1 \oplus \mathcal{I}_2' \models \tau\}$, where $\mathcal{I}_2'$ interprets Σ'. We may augment a FOTSS with a FOL sentence Bad, giving bad states in the FOTS by $\mathcal{I} \in B$ iff $\mathcal{I} \models Bad$. A FOTSS is safe if all of its corresponding FOTS are safe, and is unsafe otherwise. That is, a FOTSS is unsafe if there exists at least one FOTS corresponding to it that has at least one execution that reaches a bad state. A common way to show a FOTSS is safe is to give a formula Inv such that $\models \varphi_0 \Rightarrow Inv$ and $\models Inv \wedge \tau \Rightarrow Inv'$. Then for any FOTS over domain D, the set $I \subseteq S$ given by $I = \{\mathcal{I} \in S \mid \mathcal{I} \models Inv\}$ is an inductive set, and $\models Inv \Rightarrow \neg Bad$ then suffices to show that the state sets I, B in the FOTS are disjoint. Finding an invariant Inv satisfying the three conditions above proves the system safe.

Example 2.3.1. Consider the following FOTSS:

$$\Sigma \triangleq \{Even, +, 1, var\} \qquad \varphi_0 \triangleq Even(var)$$
$$\tau \triangleq (var' = (var + 1) + 1) \wedge Unch(Even, +, 1) \qquad Bad \triangleq \neg Even(var)$$

where $Unch(Even, +, 1)$ means that $Even$, $+$, and 1 have identical interpretations in the pre- and post-states of τ.

Our intention is to model a program that starts with an even number in a variable var and increments var by 2 at every transition. It is an error if var ever becomes odd. A natural invariant to conjecture is $Inv \triangleq Even(var)$. However, since the signature is uninterpreted, the FOTSS does not model our intention.

For example, let $D = \{0, 1, 2\}$, $\mathcal{I}_0(Even) = \{1, 2\}$, $\mathcal{I}_0(1) = 1$, $\mathcal{I}_0(+)(a, b) = a + b \bmod 3$, and $\mathcal{I}_0(var) = 1$. Thus, $\mathcal{I}_0 \models \varphi_0$. Let $\mathcal{I}_1$ be the same as $\mathcal{I}_0$, except $\mathcal{I}_1(var) = 0$. Then, $\mathcal{I}_0 \oplus \mathcal{I}_1' \models \tau$ and $\mathcal{I}_1 \models Bad$. Thus, this FOTSS is unsafe.

One way to explicate our intention in Example 2.3.1 is to axiomatize the uninterpreted functions and relations in FOL as part of φ_0 and τ. Another alternative is to restrict their interpretation to a particular structure. This is the approach we take in this paper. We define a First-Order (relative to $\mathcal{C}$) Transition System Specification (FOCTSS).

We need to be able to talk about the structural objects in $\Sigma^{\mathcal{C}}$, and so we require that every FOCTSS $(\Sigma, \varphi_0, \tau)$ be an FOTSS with $\Sigma^{\mathcal{C}} \subseteq \Sigma$. Once we have these structural objects, any structure $(D, \mathcal{I}^{\mathcal{C}}) \in \mathcal{C}$ gives a FOCTS with states $\mathcal{I}$ where $\mathcal{I}(\Sigma^{\mathcal{C}}) = \mathcal{I}^{\mathcal{C}}$, initial states $\mathcal{I}$ where $\mathcal{I} \models \varphi_0$, transitions $(\mathcal{I}_1, \mathcal{I}_2)$ where $\mathcal{I}_1 \oplus \mathcal{I}_2' \models \tau$, and bad states $\mathcal{I}$ for which $\mathcal{I} \models \mathit{Bad}$. Essentially, FOCTSS/FOCTS are the same as FOTSS/FOTS, except we have pre-emptively chosen some $\mathcal{C}$. Then, all Σ are restricted to contain $\Sigma^{\mathcal{C}}$ and all interpretations $\mathcal{I}$ are restricted to interpret $\Sigma^{\mathcal{C}}$ according to some structure in $\mathcal{C}$.

Chapter 3

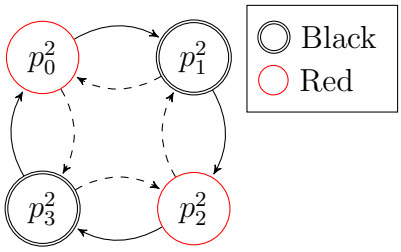
First Order Protocols

We introduce the notion of a *First-Order Protocol (FOP)* to simplify and restrict specifications in a FOTSS. We choose restrictions to make our protocols asynchronous compositions of processes over static network topologies. Each process description is relative to its process neighbourhood. For example, a process operating on a ring has access to its immediate left and right neighbours, and transitions are restricted to these processes. This simplifies the modelling.

3.1 Network Topology

We begin with formalizing the concept of a network topology. As a running example, consider a Red-Black-Ring (RBR) topology, whose instance with 4 processes is shown in Fig. 3.1a. Processes are connected in a ring of alternating Red and Black processes. Each process is connected to two neighbours using two links, labelled *left* and *right*, respectively. From the example it is clear how to extend this topology to rings of arbitrary (even) size.

To formalize this, we assume that there is a unique sort *Proc* for processes. Define $\Sigma^{\mathcal{C}} = \Sigma^{\mathcal{C}}_E \uplus \Sigma^{\mathcal{C}}_T$ to be a *topological signature*, where $\Sigma^{\mathcal{C}}_E$ is a set of unary *Proc*-sorted functions and $\Sigma^{\mathcal{C}}_T$ is a set of distinct k-ary *Proc*-sorted predicates. Functions (sometimes predicates) in $\Sigma^{\mathcal{C}}_E$ correspond to communication edges, such as *left* and *right* in our example. Predicates in $\Sigma^{\mathcal{C}}_T$ correspond to classes of processes, such as *Red* and *Black* in our example. For simplicity, we assume that all classes have the same arity k. We often omit k from the signature when it is contextually clear. We are now ready to define the concept of a network topology:



(a) Red-Black-Ring of 4 process. Dashed arrows are *right*, and solid are *left*.

$$
\begin{aligned}
\mathit{Init}: &\quad \mathit{var} \leftarrow \mathit{null} \\
\mathit{Tr}: &\quad \mathit{black} \Rightarrow \mathit{right.var} \leftarrow r \\
&\quad \mathit{red} \Rightarrow \mathit{right.var} \leftarrow b \\
\mathit{Bad}: &\quad \mathit{red} \wedge \mathit{var} = b
\end{aligned}
$$

(b) A simple protocol over Red-Black-Ring topology.

Figure 3.1: An example of a topology and a protocol.

Definition 3.1.1. A *network topology* $\mathcal{C}$ over a topological signature $\Sigma^{\mathcal{C}}$ is a collection of directed graphs $G = (V, E)$ augmented with an edge labelling $dir : E \to \Sigma_E^{\mathcal{C}}$ and k-node labelling $kind : V^k \to \Sigma_T^{\mathcal{C}}$. Given a node p in a graph $G = (V, E)$ from a network topology $\mathcal{C}$, the neighbourhood of p is defined as $nbd(p) = \{p\} \cup \{q \mid (p, q) \in E\}$. A network is *deterministic* if for every distinct pair $q, r \in nbd(p) \setminus \{p\}$, $dir(p, q) \neq dir(p, r)$. That is, each neighbour of p corresponds to a distinct name in Σ_E. If network topology $\mathcal{C}$ has any non-deterministic network $G \in \mathcal{C}$, then $\mathcal{C}$ is non-deterministic.

Given a deterministic network $G \in \mathcal{C}$, the intended interpretation of a predicate $P \in \Sigma_T^{\mathcal{C}}$ is the set of all nodes in the network topology labelled by P, and the intended interpretation of a function $f \in \Sigma_E^{\mathcal{C}}$ is such that $f(p) = q$ if an edge (p, q) is labelled by f and $f(p) = p$, otherwise.

Given a non-deterministic network $G \in \mathcal{C}$, the intended interpretation of a predicate $\Sigma_T^{\mathcal{C}}$ is the same as the deterministic case. The main difference is that $P \in \Sigma_E^{\mathcal{C}}$ are treated as relations rather than functions. Namely, for $P \in \Sigma_E^{\mathcal{C}}$, we have $P(p, q)$ iff $dir(p, q) = P$

Each graph G in a network topology $\mathcal{C}$ provides a possible intended interpretation for the sort of processes *Proc*, and the edge and node labelling provide the intended interpretation for predicates and functions in $\Sigma^{\mathcal{C}}$.

3.2 Example: Red-Black Rings

Example 3.2.1. For our running example, consider the protocol informally shown in Fig. 3.1b described by a set of guarded commands. The protocol is intended to be executed on the RBR topology shown in Fig. 3.1a. Initially, all processes start with their state

variable *var* set to a special constant *null*. Then, at each step, a non-deterministically chosen process, sends a color to its right. Every black process sends a red color r, and every red process sends a black color b. It is bad if a Red process ever gets a black color.

To formalize the topology, for each $n > 1$, let $G_n = (V_n, E_n)$, where $V_n = \{p_i^n \mid 0 \leq i < 2n\}$, and $E_n = \{(p_i^n, p_j^n) \mid |i - j| \bmod 2n = 1\}$. The edge labelling is given by $dir(p_i^n, p_j^n) = right$ if $j = (i{+}1) \mod n$ and *left* if $j = (i{-}1) \mod n$. Processes have colour $kind(p_i^n) = Red$ if i is even, and *Black* if i is odd. Finally, we define $\mathcal{RBR} = \{G_n \mid n \geq 2\}$ as the class of Red-Black Rings (RBR). □

Note that any set of graphs $\mathcal{G}$ with an upper bound on the out-degree of any vertex can be given a finite labelling according to the above definition. Further, if $\mathcal{G}$ has unbounded out-degree, an engineer may analyze a bounded sub-class $\mathcal{G}' \subset \mathcal{G}$ of the network topology.

3.3 FOP Definition

Once we have specified the topology, we want to establish how processes transition. We define the syntax and semantics of a protocol.

A protocol signature Σ is a disjoint union of a topological signature $\Sigma^{\mathcal{C}}$, a state signature Σ_S, and a background signature Σ_B. Recall that all functions and relations in $\Sigma^{\mathcal{C}}$ are of sort *Proc*. All elements of Σ_S have arity of at least 1 with the first and only the first argument of sort *Proc*. Elements of Σ_B do not allow arguments of sort *Proc* at all. Intuitively, elements of $\Sigma^{\mathcal{C}}$ describe how processes are connected, elements of Σ_S describe the current state of some process, and elements of Σ_B provide background theories, such as laws of arithmetic or uninterpreted functions.

For an interpretation $\mathcal{I}$, and a set of processes $P \subseteq \mathcal{I}(Proc)$, we write $\mathcal{I}(\Sigma_S)(P)$ for the interpretation $\mathcal{I}(\Sigma_S)$ restricted to processes in P. Similarly, we may retain topological knowledge and simply write $\mathcal{I}(P)$[1]. Intuitively, we look only at the states of P and ignore the states of all other processes.

Definition 3.3.1. A *First-Order Protocol* (FO-protocol) is a tuple $P = (\Sigma, Init(\vec{p}), Mod(p), TrLoc(p), \mathcal{C})$, where Σ is a protocol signature; $Init(\vec{p})$ is a formula with k free

[1]These definitions require no constants be in Σ, but we have already asserted that all elements of $\Sigma^{\mathcal{C}}$ have arity at least $\min(k, 2)$, Σ_S have arity greater than or equal to one, and Σ_B cannot contain process-sorted elements; thus constants are already dis-allowed for $k \geq 1$. The $k = 0$ case is pathological, however, since invariants will not quantify over any processes anyways.

variables $\vec{p}$ of sort *Proc*; $Mod(p)$ is a set of terms $\{t(p) \mid t \in dir(E)\} \cup \{p\}$; $TrLoc(p)$ is a formula over the signature $\Sigma \cup \Sigma'$ with free process variable p, and $\mathcal{C}$ is a deterministic network topology. Furthermore, $Init(\vec{p})$ is of the form $\bigwedge_{P \in \Sigma_T^{\mathcal{C}}} (P(\vec{p}) \Rightarrow Init_P(\vec{p}))$, and each $Init_P$ is a formula over $\Sigma \setminus \Sigma_{\mathcal{C}}$ (an initial state described without reference to topology for each relevant topological class); and terms of sort *Proc* occurring in $TrLoc(p)$ are a subset of $Mod(p)$.

For non-deterministic networks, $Mod(p)$ is not defined. Rather, the restriction on *TrLoc* is the following: If *Proc*-sorted term t appears in *TrLoc*, then it must be (at some higher level) of the form $\exists t \cdot P(p,t) \wedge \varphi(t)$ for some $P \in \Sigma_E^{\mathcal{C}}$. Namely, we non-deterministically choose a process t which satisfies an edge of type P and resolve the transition φ.

Note that the *semantic* local neighbourhood $nbd(p)$ and the set of *syntactic* terms in $Mod(p)$ have been connected. Namely, for every edge $(p,q) \in E$, there is a term $t(p) \in Mod(p)$ to refer to q, and for every term $t(p) \in Mod(p)$, we will refer to some process in the neighbourhood of p.

A formal description of our running example is given in Figure 3.2 as a FO-protocol. We define the signature including $\Sigma^{\mathcal{C}} = \{left, right, Red, Black\}$, the initial states $Init(p)$ in the restricted form, and modification set $Mod(p)$, where we allow processes to only write to their local neighbourhood. Next, we specify two kinds of transitions, a red t_r and a black t_b transition. Each writes to their right neighbour the colour they expect that process to be. Each process p does not change the *var* states of $p, left(p) \in Mod(p)$. Finally, we specify our local transitions $TrLoc(p)$ by allowing each of the sub-transitions. Note that all process-sorted terms in $TrLoc(p)$ are in $Mod(p) = \{left(p), p, right(p)\}$, and we are allowed to call on topological predicates in *TrLoc*, finishing our specification.

The semantics of a protocol P are given by a FOCTSS as shown in Fig. 3.3. The protocol signature Σ is the same in the FOCTSS as in the FOP. Initially, φ_0 requires that all k-tuples of a given topology satisfy a topology-specific initial state. Second, to take a transition τ, some process takes a local transition $TrLoc(p)$ modifying states of processes that can be described using the terms in $Mod(p)$. $Frame(p), Unch(y)$ guarantee that the transition does not affect local state of processes that are outside of $Mod(p)$. Finally, $UnMod$ makes all functions and predicates in the background signature retain their interpretation during the transition. Overall, this describes a general multiprocess asynchronous protocol.

For non-deterministic network topologies, $Frame(p) \triangleq UnMod \wedge \forall y \cdot \bigwedge_{P \in \Sigma_E^{\mathcal{C}}} \neg P(p,y) \Rightarrow Unch(y)$, since $Mod(p)$ is undefined.

$$
\begin{aligned}
Const &= \{null_{/0}, r_{/0}, b_{/0}\} \qquad Func = \{left_{/1}, right_{/1}, var_{/1}\} \\
Pred &= \{Red_{/1}, Black_{/1}, =_{/2}\} \qquad \Sigma = (Const, Func, Pred) \\
Init(p) &= (Red(p) \Rightarrow var(p) = null) \wedge (Black(p) \Rightarrow var(p) = null) \\
Mod(p) &= \{p, right(p), left(p)\} \\
t_r(p) &= var'(right(p)) = b \wedge var'(p) = var(p) \wedge var'(left(p)) = var(left(p)) \\
t_b(p) &= var'(right(p)) = r \wedge var'(p) = var(p) \wedge var'(left(p)) = var(left(p)) \\
TrLoc(p) &= (Red(p) \Rightarrow t_r(p)) \wedge (Black(p) \Rightarrow t_b(p))
\end{aligned}
$$

Figure 3.2: A FO-protocol description of the system from Fig. 3.1.

$$
\begin{aligned}
\varphi_0 \triangleq \forall \vec{p} \cdot Init(\vec{p}) \quad \tau \triangleq \exists p \cdot TrLoc(p) \wedge Frame(p) \\
Frame(p) \triangleq UnMod \wedge (\forall y \cdot y \notin Mod(p) \Rightarrow Unch(y))) \\
Unch(y) \triangleq \left(\bigwedge_{P \in Pred_S} \forall \vec{v} \cdot P(y, \vec{v}) \iff P'(y, \vec{v}) \right) \wedge \left(\bigwedge_{f \in Func_S} \forall \vec{v} \cdot f(y, \vec{v}) = f'(y, \vec{v}) \right) \\
UnMod \triangleq \left(\bigwedge_{P \in Pred_B} \forall \vec{v} \cdot P(\vec{v}) \iff P'(\vec{v}) \right) \wedge \left(\bigwedge_{f \in Func_B} \forall \vec{v} \cdot f(\vec{v}) = f'(\vec{v}) \right)
\end{aligned}
$$

Figure 3.3: An FOTS of the protocol in Fig. 3.2.

This definition of a FO-protocol places some added structure on the notion of FOTSS. It restricts how transition systems can be specified, which might seem like a drawback. On the contrary, the added structure provides two benefits. First, it removes the need for axiomatizing the network topology, since the topology is given semantically by $\mathcal{C}$. Second, the system guarantees that we model asynchronous composition of processes with local transitions – a common framework for specifying and reasoning about protocols.

To show safety of such a system, we will be concerned with invariants which only discuss a few processes, say $Inv(\vec{p})$ where $\vec{p} = p_1, \ldots, p_k$. Then our FO-invariants will be of the form $\forall \vec{p} \cdot Inv(\vec{p})$, and substituting φ_0 into our background, we find a natural check for when

a given formula is inductive:

$$InvOk(Inv) \triangleq ((\forall \vec{p} \cdot Init(\vec{p})) \Rightarrow (\forall \vec{p} \cdot Inv(\vec{p}))) \wedge (((\forall \vec{p} \cdot Inv(\vec{p})) \wedge \tau) \Rightarrow (\forall \vec{p} \cdot Inv'(\vec{p})))$$

Indeed, by unpacking definitions, one sees that $\models_{\mathcal{C}} InvOk$ means that every state on any trace of a FOCTS satisfies $\forall \vec{p} \cdot Inv(\vec{p})$, and thus it suffices to check that $\models_{\mathcal{C}} \forall \vec{p} \cdot Inv(\vec{p}) \Rightarrow \neg Bad$ to prove safety. We, however, will focus on the task of verifying a candidate formula as inductive or not.

To decide if a candidate is inductive or not requires reasoning in $FOL^{\mathcal{C}}$. However, reasoning about FOL extended with an arbitrary topology is difficult (or undecidable in general). We would like to reduce the verification problem to pure FOL. One solution is to axiomatize the topology in FOL – this is the approach taken by Ivy[20]. Another approach is to use properties of the topology to reduce reasoning about FO-protocols to FOL. This is similar to the use of topology to reduce reasoning about parameterized finite-state systems to reasoning about finite combinations of finite-state systems in [16]. In the next section, we show how this approach can be extended to FO-protocols.

Chapter 4

Local Reasoning

In this section, we present a technique for reducing verification of FO-protocols over a given network topology $\mathcal{C}$ to a decision problem in pure FOL. We assume that we are given a (modular) inductive invariant $\forall \vec{q} \cdot Inv(\vec{q})$ of the form $\left(\forall \vec{q} \cdot \bigwedge_{Top \in \Sigma_T^{\mathcal{C}}} Top(\vec{q}) \Rightarrow Inv_{Top}(\vec{q})\right)$. That is, Inv has a local inductive invariant $Inv_{Top(\vec{q})}$ for each topological class Top.

Given a First-Order Protocol and candidate invariant, we want to know if $\models_{\mathcal{C}} InvOk$. But deciding this is hard, and so we show that deciding validity of $InvOk$ can be done in pure FOL using modular verification conditions in the style of Owicki-Gries [18] and Paramaterized Compositional Model Checking [16].

4.1 Verifying FO-Protocols using First Order Logic

The input to our procedure is a formula Inv_{Top} over signature $\Sigma_B \uplus \Sigma_S$ for each topological class $Top \in \Sigma_T^{\mathcal{C}}$. The VC is a conjunction of sentences ensuring that for each tuple of processes $\vec{q}$ in a topological class Top, $Inv_{Top}(\vec{q})$ is true initially, is stable under a transition of one process in $\vec{q}$, and is stable under interference by any other process p whose execution might affect some $q_i \in \vec{q}$. If the VC is FOL-valid, an inductive invariant has been found. If not, there will be a local violation to inductiveness, which may correspond to a global violation.

Formally, $VC(Inv)$ is a conjunction of statements of the following two forms:

$$\forall \vec{q} \cdot (CrossInit_{Top}(\vec{q}) \Rightarrow Inv_{Top}(\vec{q})) \tag{4.1}$$

$$\forall p, \vec{q} \cdot ((CrossInv_{Top}(Mod(p), \vec{q}) \wedge \tau) \Rightarrow Inv'_{Top}(\vec{q})) \tag{4.2}$$

Statements of form (4.1) require that every local neighbourhood of $\vec{q}$ that satisfies all appropriate initial states also satisfies $\vec{q}$'s invariant. Statements of form (4.2) capture both transitions where $p = q_i$ for some i, or process p acts and modifies $q_i \in nbd(p)$, since p is quantified universally. All that remains is to formally construct the statements $CrossInit$, $CrossInv$. In order to do so, we construct a local characteristic formula $\chi_{Top}(A, \vec{q})$ of a process $\vec{q}$ (satisfying $Top(\vec{q})$) and neighbourhood A for each topological class $Top \in \Sigma_T^{\mathcal{C}}$. Intuitively, we aim for $\chi(A, \vec{q})$ to encode the available local neighbourhoods of processes in A and $\vec{q}$ in $\mathcal{C}$.

Let $\chi_{Top}(A, \vec{q})$ be the strongest formula that satisfies $\models_{\mathcal{C}} \forall \vec{q} \cdot Top(\vec{q}) \Rightarrow \chi_{Top}(A, \vec{q})$, subject to the following syntactic restrictions. A formula is a candidate for $\chi_{Top}(A, \vec{q})$ when it is (1) over signature $\Sigma_T^{\mathcal{C}} \cup \Sigma_E^{\mathcal{C}} \cup \{=\}$, (2) contains only terms $A \cup \{q_i \mid q_i \in \vec{q}\}$, and (3) is in CNF and all literals from $\Sigma_T^{\mathcal{C}}$ appear in positive form. The syntactic restrictions are to capture when elements of $A, \vec{q}$ satisfy various topological notions given by signature $\Sigma_E^{\mathcal{C}} \cup \{=\}$. We also never force some processes to be outside of some topological class. Intuitively, χ is a formula that captures all topological knowledge derivable from the topology given that we know that $Top(\vec{q})$ holds. For instance, in $\mathcal{RBR}$, we have $\chi_{Red}(\emptyset, q) = Red(q)$, while expanding this for $A = \{left(p), p, right(p)\}$ results in the following formula. We drop some trivial statements and convert away from CNF for readability. For instance, $left$, $right$ are inverse functions.

$$\begin{aligned}\chi_{Red}(\{left(p), p, right(p)\}, q) = Red(q) \wedge distinct(left(p), p, right(p)) \wedge \\ ((Red(left(p)) \wedge Black(p) \wedge Red(right(p)) \wedge p \neq q) \vee \\ (Black(left(p)) \wedge Red(p) \wedge Black(right(p)) \wedge distinct(left(p), right(p), q)))\end{aligned}$$

These characteristics are illustrated in Figure 4.1. When we just look at $\chi_{Red}(\emptyset, q)$, we find q is red. However, if we expand our local reasoning to the characteristic $\chi_{Red}(Mod(p), q)$, we find that there are two options given by $\mathcal{RBR}$. One option is p is red, and $q = p$ is optional (dotted lines), while $q \neq left(p), right(p)$. Alternatively, p is black, and $q \neq p$, but q could be $left(p), right(p)$, or neither.

Once we have $\chi_{Top}(A, \vec{q})$, we can define our statements $CrossInit_{Top}$, $CrossInv_{Top}$. First, $CrossInit_{Top}(\vec{q})$ is obtained from $\chi_{Top}(\emptyset, \vec{q})$ by replacing every instance of $Top_i(\vec{q})$ with $Init_{Top_i}(\vec{q})$. We build our interference constraints in a similar way. We construct $CrossInv_{Top}(\vec{q})$ by modifying $\chi_{Top}(Mod(p), \vec{q})$[1]. Namely, we obtain $CrossInv_{Top}(Mod(p), \vec{q})$ from $\chi_{Top}(Mod(p), \vec{q})$ by replacing every instance of $Top_i(\vec{q})$ with $(Top_i(\vec{q}) \wedge Inv_{Top_i}(\vec{q}))$.

[1]For non-deterministic topologies, an engineer may make terms for each element of a local neighbourhood and explicitly tell an engine to assume they are connected by the relevant edge type, then using the explicitly named terms instead of $Mod(p)$.

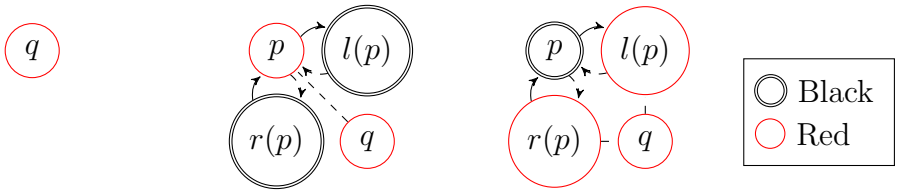


Figure 4.1: Characteristics $\chi_{Red}(\emptyset, q)$ and $\chi_{Red}(Mod(p), q)$ for the $\mathcal{RBR}$ topology.

$$\forall p \cdot Init_{red}(p) \Rightarrow Inv_{red}(p) \tag{4.3}$$

$$\begin{aligned}\forall p, q \cdot (&Red(q) \wedge Inv_{red}(q) \wedge Red(left(p)) \wedge Inv_{red}(left(p)) \wedge \\ &Black(p) \wedge Inv_{black}(p) \wedge Red(right(p)) \wedge Inv_{red}(right(p)) \wedge \\ &p \neq q \wedge distinct(left(p), p, right(p))) \Rightarrow Inv'_{red}(q)\end{aligned} \tag{4.4}$$

$$\begin{aligned}\forall p, q \cdot (&Red(q) \wedge Inv_{red}(q) \wedge Black(left(p)) \wedge Inv_{black}(left(p)) \wedge \\ &Red(p) \wedge Inv_{red}(p) \wedge Black(right(p)) \wedge Inv_{black}(right(p)) \wedge \\ &distinct(left(p), right(p), q) \wedge distinct(left(p), p, right(p))) \Rightarrow Inv'_{red}(q)\end{aligned} \tag{4.5}$$

$$VC_{P,1}(Inv_{red}, Inv_{black}) \triangleq (4.3) \wedge (4.4) \wedge (4.5) \tag{4.6}$$

Figure 4.2: The verification conditions VC_{Red} for the red process invariant.

Example 4.1.1. The VC generated by the $\mathcal{RBR}$ topology may be partitioned into VC_{Red} and VC_{Black}, each consisting of the statements whose conclusions are Inv_{Red}, Inv'_{Red} and $Inv_{black}, Inv'_{black}$, respectively. VC_{Red} is shown in Fig. 4.2. The conditions for VC_{Black} are symmetric. One can check that

$$Inv_{red}(p) \triangleq var(p) \neq b \qquad\qquad Inv_{black}(p) \triangleq \top$$

is an inductive invariant for the protocol in Fig. 3.1. □

In practice, the role of the χ-computing oracle can be filled by a verification engineer. A description of local neighbourhoods starts by allowing all possible neighbourhoods. Then, an engineer may dismiss local configurations that cannot occur on the topology as they occur.

4.2 Soundness and Completeness

In this section, we present soundness and relative completeness of our verification procedure from Section 4.1.

4.2.1 Soundness.

To show soundness, we present a model-theoretic argument to show that whenever the verification condition from Section 4.1 is valid in FOL, then the condition *InvOk* is valid in FOL extended with the given topology $\mathcal{C}$.

Theorem 4.2.1. *Given a FO-protocol P and a local invariant per topological class $Inv_{Top_1}(\vec{p}), \ldots, Inv_{Top_n}(\vec{p})$, if $\vDash VC(Inv)$, then $\vDash_{\mathcal{C}} InvOk(Inv)$.*

Proof. Assume $\models VC(Inv)$. We show that $InvOk(Inv)$ is valid in $FOL^{\mathcal{C}}$ by showing that any pair of $FOL^{\mathcal{C}}$ interpretations $\mathcal{I}$ and $\mathcal{I}'$ satisfy $VC(Inv)$ as FOL interpretations, and this is strong enough to guarantee $\mathcal{I} \oplus \mathcal{I}' \models Inv Ok(Inv)$, and thus $\mathcal{I} \oplus \mathcal{I}' \models_{\mathcal{C}} InvOk(Inv)$.

Let $\mathcal{I}, \mathcal{I}'$ be $FOL^{\mathcal{C}}$ interpretations over some $G = (V, E) \in \mathcal{C}$. Then $\mathcal{I} \oplus \mathcal{I}' \models VC(Inv)$ because $VC(Inv)$ is valid and $\mathcal{I} \oplus \mathcal{I}'$ is an FOL interpretation.

We first show that $\mathcal{I} \models (\forall \vec{p} \cdot Init(\vec{p}) \Rightarrow \forall \vec{p} \cdot Inv(\vec{p}))$. Suppose that $\mathcal{I} \vDash \forall \vec{p} \cdot Init(\vec{p})$. Let $\vec{p}$ be an aribtrary tuple in G. If $\mathcal{I} \models \neg Top_i(\vec{p})$ for every $Top_i \in \Sigma_T$, then $Inv(\vec{p})$ follows vacuously. Otherwise, suppose $\mathcal{I} \models Top_i(\vec{p})$. Then by definition of χ, we obtain $\mathcal{I} \models \chi_{Top_i}(\emptyset, \vec{p})$ since $\mathcal{I}$ is a $\mathcal{C}$ interpretation, so $\mathcal{I} \models Top_i(\vec{p}) \Rightarrow \chi_{Top_i}(\emptyset, \vec{p})$. Since $\mathcal{I} \models \forall \vec{p} \cdot Init(\vec{p})$, this gives us that $\mathcal{I} \models CrossInit(\vec{p})$ (for any $Top_j(\vec{p}')$ in $\chi_{Top_i}(\emptyset, \vec{p})$, find that $Init(\vec{p}')$, and thus $Top_j(\vec{p}')$ implies $Init_{Top_j}(\vec{p}')$, giving $CrossInit$). Since $\mathcal{I} \models CrossInit_{Top_i}(\vec{p})$ and $\mathcal{I} \models VC$, we get $\mathcal{I} \models CrossInit_{Top_i}(\vec{p}) \Rightarrow Inv_{Top_i}(\vec{p})$, finally giving us $\mathcal{I} \models Inv_{Top_i}(\vec{p})$, as desired.

Second, we show that $\mathcal{I} \oplus \mathcal{I}' \models (\forall \vec{p} \cdot Inv(\vec{p})) \wedge \tau \Rightarrow (\forall \vec{p} \cdot Inv(\vec{p}))$. Suppose that $\mathcal{I} \models \forall \vec{p} \cdot Inv(\vec{p})$ and $\mathcal{I} \oplus \mathcal{I}' \models TrLoc(p) \wedge Frame(p)$ for some $p \in V$. We show that $\mathcal{I}' \models \forall \vec{q} \cdot Inv'(\vec{q})$. Let $\vec{q} \in V^k$ be an arbitrary process tuple. If $\mathcal{I}' \not\models Top_i(\vec{q})$ for all $1 \leq i \leq n$, then $\mathcal{I}' \models Inv'(\vec{q})$ vacuously. Suppose $\mathcal{I}' \models Top_i(\vec{q})$ for some $Top_i \in \Sigma_T$. Then since $\mathcal{I}$ is a $\mathcal{C}$ interpretation, $\mathcal{I} \models Top_i(\vec{q}) \Rightarrow \chi_{Top_i}(Mod(p), \vec{q})$, and so $\mathcal{I} \models \chi_{Top_i}(Mod(p), \vec{q})$. Again by instantiating $\forall \vec{p} \cdot Inv(\vec{p})$ on terms in $Mod(p), \vec{q}$, we may obtain that $\mathcal{I} \models CrossInv(Mod(p), \vec{q})$. Combined, we have $\mathcal{I} \oplus \mathcal{I}' \models CrossInv(Mod(p), \vec{q}) \wedge \tau$. Applying VC finally gives $Inv_{Top_i}(\vec{q})$. Thus both conjuncts of $InvOk(Inv)$ are satisfied, giving our result.

□

Intuitively, the correctness of Theorem 4.2.1 follows from the fact that any interpretation under $FOL^{\mathcal{C}}$ is also an interpretation under FOL, and all preconditions generated for VC are true under any $FOL^{\mathcal{C}}$ interpretation.

4.2.2 Small model property.

Checking validity of universally quantified statements in FOL is in the fragment EPR, and thus we obtain a result saying that we only need to consider models of a given size. This means that a FOL solver needs to only reason about finitely many elements of sort *Proc*. It further means that topologies such as $\mathcal{RBR}$ may be difficult to compile to EPR in Ivy, but our methodology guarantees our verifications will be in EPR.

Theorem 4.2.2. *If* $\models VC(Inv)$ *for all process domains of size at most* $|Mod(p)|+k$*, then* $\models_{\mathcal{C}} InvOk(Inv)$*.*

Proof. By contrapositive, suppose $\not\models_{\mathcal{C}} InvOk(Inv)$. Then, by Theorem 4.2.1, $\not\models VC(Inv)$. Let $\mathcal{I} \oplus \mathcal{I}'$ be a falsifying interpretation. It contains an assignment to $Mod(p)$ and $\vec{q}$, or to $\vec{p}$ that makes at least one statement in $VC(Inv)$ false. Then $\mathcal{I} \oplus \mathcal{I}'(Mod(p) \cup \vec{q})$ or $\mathcal{I}(\vec{p})$ is also a counter-model to $VC(Inv)$, but with at most $|Mod(p)|+k$ elements of sort *Proc*. □

4.2.3 Relative Completeness.

We show that our method is relatively complete for local invariants that satisfy the *completability* condition. Let $\varphi(\vec{p})$ be a formula of the form $\bigwedge_{i=1}^{n}(Top_i(\vec{p}) \Rightarrow \varphi_{Top_i}(\vec{p}))$ with $\varphi_{Top_i}(\vec{p})$ over the signature $\Sigma_S \cup \Sigma_B$. Intuitively, $\varphi(\vec{p})$ is *completable* if every interpretation $\mathcal{I}$ that satisfies $\forall \vec{p} \cdot \varphi(\vec{p})$ and is consistent with some $\mathcal{C}$-interpretation $\mathcal{I}_G$ can be extended to a full $\mathcal{C}$-interpretation that satisfies $\forall \vec{p} \cdot \varphi(\vec{p})$. Formally, φ is *completable* relative to topology $\mathcal{C}$ iff for every interpretation $\mathcal{I}$ with domain $U \subseteq V$ for $G = (V, E) \in \mathcal{C}$ with an intended interpretation $\mathcal{I}_G$ such that $(\mathcal{I} \uplus \mathcal{I}_G)(U) \models \forall \vec{p} \cdot \varphi(\vec{p})$, there exists an interpretation $\mathcal{J}$ with domain V s.t. $(\mathcal{J} \uplus \mathcal{I}_G) \models \forall \vec{p} \cdot \varphi$ and $\mathcal{I}(U) = \mathcal{J}(U)$. To obtain relative completeness, we use a lemma for when a FOL interpretation can be lifted to a $\mathcal{C}$ interpretation.

Lemma 4.2.1. *If FOL interpretation* $\mathcal{I}$ *of signature* $\Sigma^{\mathcal{C}}$ *satisfies* $\mathcal{I} \models \chi_{Top}(A, \vec{q})$*, then there exists a* $\mathcal{C}$ *interpretation* $\mathcal{J}$ *of the same signature with* $\mathcal{J} \models \chi_{Top}(A, \vec{q})$ *and* $\mathcal{I} \models t_i = t_j$ *iff* $\mathcal{J} \models t_i = t_j$ *for terms* $t_i, t_j \in A \cup \vec{q}$*.*

Proof. Let $\mathcal{I} \models \chi_{Top}(A, \vec{q})$. Let $\varphi(A, \vec{q})$ be the conjunction of all atomic formulae over the signature $\{=\}$ and statements $\neg Top_j(\vec{q}')$ that is true of elements of $A, \vec{q}$ in interpretation $\mathcal{I}$. If no $\mathcal{C}$ interpretation $\mathcal{J} \models Top(\vec{q}) \wedge \varphi(A, \vec{q})$, then we can add the clause $\neg\varphi(A, \vec{q})$ to $\chi_{Top}(A, \vec{q})$, thus strengthening it (this is stronger since $\mathcal{I} \models Top(\vec{q}), \not\models \neg\varphi(A, \vec{q})$, and is true of every interpretation modelling $Top(\vec{q})$). However, this violates the assumptions that χ_{Top} is as strong as possible. Thus, some $\mathcal{J} \models Top(\vec{q}) \wedge \varphi(A, \vec{q})$. Note that $\mathcal{J}$ already satisfies $t_i = t_j$ iff $\mathcal{I}$ satisfies $t_i = t_j$ since every statement of $=, \neq$ is included in $\varphi(A, \vec{q})$. Finally, since $\mathcal{J}$ is a $\mathcal{C}$ interpretation and $\mathcal{J} \models Top(\vec{q})$, then $\mathcal{J} \models \chi_{Top}(A, \vec{q})$ by definition. □

Theorem 4.2.3. *Given an FO-protocol P, if $\models_{\mathcal{C}} InvOk(Inv)$ and both $Inv(\vec{p})$ and $Init(\vec{p})$ are completable relative to $\mathcal{C}$, then $\models VC(Inv)$.*

Proof. By contra-positive, we show that given a completable local invariant $Inv(\vec{p})$, if $VC(Inv)$ is falsifiable in FOL, then $InvOk(Inv)$ is falsifiable in $FOL^{\mathcal{C}}$. Suppose $VC(Inv)$ is not valid, and let $\mathcal{I} \oplus \mathcal{I}'$ by such that $\mathcal{I} \oplus \mathcal{I}' \not\models VC(Inv)$. We consider two cases – a violation initially or inductively.

Case 1: Initialization: For some processes $\vec{p} = (p_1, \ldots, p_k)$ and $1 \leq i \leq |\Sigma_T^{\mathcal{C}}|$, $\mathcal{I} \models CrossInit_{Top_i}(\vec{p})$ and $\mathcal{I} \not\models Inv_{Top_i}(\vec{p})$. Modify $\mathcal{I}(\Sigma_T)$ for every $\vec{q}$ so that $Top_j(\vec{q})$ is interpreted to be true iff $Init_{Top_j}(\vec{q})$ is true. Noting that all initial conditions are outside of the signature $\Sigma_T^{\mathcal{C}}$, we observe that this is done without loss of generality. Since our construction maintains $Top_i(\vec{q}) \Rightarrow Init_{Top_i}(\vec{q})$ for all $\vec{q} \subseteq \{p_i\}^k$, we maintain $\mathcal{I} \models CrossInit_{Top_i}(\vec{p})$. Thus we conclude now that $\mathcal{I} \models \chi_{Top_i}(\emptyset, \vec{p})$. Applying Lemma 4.2.1 to $\mathcal{I}(\Sigma^{\mathcal{C}})$, we get a $\mathcal{C}$ interpretation $\mathcal{J} \models \chi_{Top_i}(\emptyset, \vec{p}^{\mathcal{C}})$. Since this model has the same equalities of terms $\vec{p}^{\mathcal{C}}$ in $\mathcal{J}$ as $\vec{p}$ in $\mathcal{I}$, we may copy the states $\mathcal{I}(\Sigma_S)(p_i)$ to $\mathcal{J}(\Sigma_S)(p_i^{\mathcal{C}})$. Set $\mathcal{J}(\Sigma_B) = \mathcal{I}(\Sigma_B)$. Since $Init$ is completable by assumption, we complete $\mathcal{J}(\Sigma_S \cup \Sigma_B)(\vec{p})$ to $\mathcal{J}(\Sigma_S \cup \Sigma_B)$, completing our construction of $\mathcal{J}$ interpreting $\Sigma^{\mathcal{C}} \cup \Sigma_S \cup \Sigma_B$. Note that $\mathcal{J} \models \forall\vec{p} \cdot Init(\vec{p})$, but $\mathcal{J} \models Top_i(\vec{p}^{\mathcal{C}}) \wedge \neg Inv_{Top_i}(\vec{p}^{\mathcal{C}})$, thus showing that $InvOk(Inv)$ is falsifable in $FOL^{\mathcal{C}}$.

Case 2: Inductiveness: For some $p, \vec{q}$, and $1 \leq i \leq |\Sigma_T^{\mathcal{C}}|$, we have $\mathcal{I} \models CrossInv_{Top_i}(Mod(p), \vec{q})$, $(\mathcal{I} \oplus \mathcal{I}') \models TrLoc(p) \wedge Frame(p)$, and $\mathcal{I}' \not\models Inv_{Top_i}(\vec{q})$. By construction, $\models CrossInv(Mod(p), \vec{q}) \Rightarrow \chi_{Top_i}(Mod(p), \vec{q})$. Applying Lemma 4.2.1 to $\mathcal{I}(\Sigma^{\mathcal{C}}) \models \chi_{Top_i}(Mod(p), \vec{q})$, we get a $\mathcal{C}$ interpretation of $\Sigma_T^{\mathcal{C}}$, $\mathcal{J} \models \chi_{Top_i}(Mod(p^{\mathcal{C}}), \vec{q}^{\mathcal{C}})$. We extend this to a full model $\mathcal{J} \oplus \mathcal{J}'$ of signature $\Sigma^{\mathcal{C}} \cup \Sigma_S \cup \Sigma_B$, and its primed copy. We set $\mathcal{J}'(\Sigma^{\mathcal{C}}) = \mathcal{J}(\Sigma^{\mathcal{C}})$. Then, since $\mathcal{J}$ and $\mathcal{I}$, and $\mathcal{J}'$ and $\mathcal{I}'$ share equalties across terms in $Mod(p) \cup \vec{q}$ and $Mod(p^{\mathcal{C}}) \cup \vec{q}^{\mathcal{C}}$, we can lift states from terms $t \in Mod(p) \cup \vec{q}$

by $\mathcal{J}(\Sigma_S \cup \Sigma_B)(t^{\mathcal{C}}) \triangleq \mathcal{I}(\Sigma_S \cup \Sigma_B)(t)$ and $\mathcal{J}'(\Sigma'_S)(t^{\mathcal{C}}) \triangleq \mathcal{I}'(\Sigma'_S)(t)$. Since Inv is completable, we complete $\mathcal{J}(\Sigma_S \cup \Sigma_B)(Mod(p^{\mathcal{C}} \cup \vec{q}^{\mathcal{C}})$ to $\mathcal{J}(\Sigma_S \cup \Sigma_B)$ and clone the completion to $\mathcal{J}'(\Sigma_S \cup \Sigma_B)(V \setminus (Mod(p^{\mathcal{C}}) \cup \vec{q}^{\mathcal{C}}))$. Overall, this completes the interpretation $\mathcal{J} \oplus \mathcal{J}'$.

Note that $\mathcal{J} \models \forall \vec{p} \cdot Inv(\vec{p})$ by construction. Similarly, $\mathcal{J} \oplus \mathcal{J}' \models \tau$ since $\mathcal{I} \oplus \mathcal{I}' \models \tau(p)$ and $Mod(p)$ terms are lifted directly from $\mathcal{I}$ and $\mathcal{I}'$ to $\mathcal{J}$ and $\mathcal{J}'$. Finally, $\mathcal{J}' \models \neg Inv'_{Top_i}(\vec{q})$ since $\mathcal{J}'(\Sigma_S)$ is lifted directly from $\mathcal{I}'(\Sigma_S \cup \Sigma_B)$, which is the language of invariants. Thus, we have shown that $InvOk(Inv)$ is falsifiable in $FOL^{\mathcal{C}}$ in this case as well.

□

How restrictive is the requirement of *completability*? Intuitively, suppose a protocol is very restrictive about how processes interact. Then the system is likely sufficiently intricate that trying to reason locally may be difficult independant of our FOL methodology. For instance, the invariant we later find for leader election is not completable. However, if equivalence classes are small, then most reasonable formulae satisfy the completability condition.

Theorem 4.2.4. *If $Inv_{Top_i}(p)$ is satisfiable over any domain for each $1 \leq i \leq n$ and topological predicates are of arity $k = 1$, then $Inv(p)$ is completable.*

Proof. Let $Inv_i(p)$ be satisfiable for each $1 \leq i \leq n$. Then let $\mathcal{I}(V')$ be an interpretation of $\Sigma_B \uplus \Sigma_S$ over domain $V' \subseteq V$ for $G = (V, E) \in \mathcal{C}$, such that $\mathcal{I}(V') \models \forall p \cdot Inv(p)$. Let $p \in V \setminus V'$, suppose $\mathcal{I}_G \models Top_i(p)$ for some $1 \leq i \leq n$. Then choose $\mathcal{J}(p) \models Inv_{Top_i}(p)$ since $Inv_{Top_i}(p)$ is satisfiable. Otherwise, if $\mathcal{I}_G \not\models Top_i(p)$ for all $1 \leq i \leq n$, then $\mathcal{J}(p)$ is chosen arbitrarily. In either case, $\mathcal{J} \models Inv(p)$. Make this construction for each $p \in V \setminus V'$. Finally, for $p \in V'$, define $\mathcal{J}(p) = \mathcal{I}(p)$. Then $\mathcal{J}$ completes the partial interpretation $\mathcal{I}$. □

Theorem 4.2.4 can be generalized to the case where the topological kinds Σ_T are non-overlapping, and individually completable, where by individually completable, we mean that if $Top(\vec{p})$ and process states of $\vec{p}' \subset \vec{p}$ are given, then there is a way to satisfy $Inv(\vec{p})$ without changing the states of $\vec{p}'$.

Chapter 5

Case Studies

We proceed to illustrate our technique on two full-scale protocols; leader election on both rings and trees. Leader election on rings is used to demonstrate the usefulness of Ivy[20], although we modify the protocol slightly for our use. Leader election on trees provides a fresh topology which demonstrates that we are not simply using the same axiomatization found in Ivy.

5.1 Example: Leader Election Ring Protocol

In this section, we illustrate our approach by applying it to the well-known leader election protocol [4]. This is essentially the same protocol used to illustrate Ivy in [20]. The goal of the protocol is to choose a unique leader on a ring. Each process sends messages to its neighbour on one side and receives messages from a neighbour on the other side. Initially, all processes start with distinct identifiers, *id*, that are totally ordered. Processes pass *id*s around the ring and declare themselves the leader if they ever receive their own *id*.

We implement this behaviour by providing each process a comparison variable *comp*. Processes then pass the maximum between *id* and *comp* to the next process's *comp* variable. A process whose *id* and *comp* have the same value is the leader. The desired safety property is that there is never more than one leader in the protocol.

In [20], the protocol is modelled by a global transition system. The system maintains a bag of messages for each process. At each step, a currently waiting message is selected and processed according to the program of the protocol (or a fresh message is generated). The

$$Const \triangleq \{0_{/0}\} \quad Func \triangleq \{next_{/1}, id_{/1}, comp_{/1}\} \quad Pred \triangleq \{\leq_{/2}, =_{/2}, btw_{/3}\} \quad \mathcal{C} \triangleq \mathcal{BTW}$$
$$\Sigma \triangleq (Const, Func, Pred) \quad LO_0(\leq) \triangleq LO(\leq) \wedge \forall x \cdot 0 \leq x \quad Mod(p) \triangleq \{p, next(p)\}$$
$$Init(p) \triangleq (LO_0(\leq) \wedge btw(x, y, z) \Rightarrow (distinct(id(x), id(y), id(z)) \wedge 0 < id(x) \wedge comp(x) = 0))$$
$$\tau_1(p) \triangleq (id(p) \leq comp(p) \Rightarrow (comp'(next(p)) = comp(p)))$$
$$\tau_2(p) \triangleq (comp(p) \leq id(p) \Rightarrow (comp'(next(p)) = id(p)))$$
$$TrLoc(p) \triangleq (id(p) = id'(p) \wedge comp(p) = comp'(p) \wedge id'(next(p)) = id(next(p)) \wedge \tau_1(p) \wedge \tau_2(p))$$

Figure 5.1: A model of the Leader Election protocol as a FO-protocol.

network topology is axiomatized, as shown in Section 1. Here, we present a local model of the protocol and verify it locally.

Network topology. The leader election protocol operates on a ring of size at least 3. For $n \geq 3$, let $G_n = (V_n, E_n)$, where $V_n = \{p_i^n \mid 0 \leq i < n\}$ and $E_n = \{(p_i^n, p_j^n) \mid 0 \leq i < n, j = i + 1 \bmod n\}$. Let $\Sigma_E = \{next\}$ and $\Sigma_T = \{btw\}$, where btw is a ternary relation such that $btw(p_i^n, p_j^n, p_k^n)$ iff $i < j < k$, $j < k < i$, or $k < i < j$. Finally, the network topology is $\mathcal{BTW} = \{G_n \mid n \geq 3\}$. Note that while $\mathcal{BTW}$ can be axiomatized in FOL (for finite models), we do not require such an axiomatization. The definition is purely semantic, no theorem prover sees it.

A formal specification of the leader election as an FO-protocol is shown in Fig. 5.1, where $LO(\leq)$ is an axiomatization of total order from [20], and $x < y$ stands for $x \leq y \wedge x \neq y$. The model follows closely the informal description of the protocol given above. The safety property is $\neg Bad$, where $Bad = \exists x, y, z \cdot btw(x, y, z) \wedge id(x) = comp(x) \wedge id(y) = comp(y)$. That is, a bad state is reached when two processes that participate in the btw relation are both leaders.

A local invariant Inv_{lead} based on the invariant from [20] is shown in Fig. 5.2. The invariant first says if an id passes from y to x through z, then it must witness $id(y) \geq id(z)$ to do so. Second, the invariant says that if a process is a leader, then it has a maximum id. Finally, the invariant asserts our safety property.

This invariant was found interactively with Ivy by seeking local violations to the invariant. Our protocol's btw is uninterpreted, while Ivy's btw is explicitly axiomatized. The inductive check assumes that the processes $p, next(p), \vec{q}$ all satisfy a finite instantiation of

$$(btw(x,y,z) \wedge id(y) = comp(x)) \Rightarrow (id(z) \leq id(y))$$
$$(btw(x,y,z) \wedge id(x) = comp(x)) \Rightarrow (id(y) \leq id(x) \wedge id(z) \leq id(x))$$
$$(btw(x,y,z) \wedge id(x) = comp(x) \wedge id(y) = comp(y)) \Rightarrow x = y$$

Figure 5.2: Local inductive invariant $Inv_{lead}(x,y,z)$ for Leader Election from Fig. 5.1.

the ring axioms (this could be done by the developer as needed if an axiomatization is unknown, and this is guaranteed to terminate as there are finitely many relevant terms), and $btw(\vec{q})$. Once the invariants are provided, the check of inductiveness is mechanical[1]. Overall, this presents a natural way to model protocols for engineers that reason locally.

An uncompletable invariant The invariant for the leader election is not completable. To see this, we present a partial interpretation $\mathcal{I}$ over $\{p_0^3, p_2^3\} \subseteq V_3$ from G_3 with no extension. We choose $\mathcal{I}(\leq)$ to be $\leq$ over $\mathbb{N}$, as intended. Then we choose $\mathcal{I}(id)$ to map $p_0^3 \mapsto 1$ and $p_2^3 \mapsto 2$. We also choose $\mathcal{I}(comp)$ to map $p_0^3 \mapsto 0$ and $p_2^3 \mapsto 1$. Since no tuple satisfies btw, this vacuously satisfies all invariants thus far. Let $\mathcal{J}$ be a $\mathcal{BTW}$ interpretation agreeing on p_0^3, p_2^3. Consider $n = id(p_1^3)$. We know $id(p_1^3) \neq 0, 1, 2$ since we require distinct ids across the new btw relation. But we also have $id(p_0^3) = comp(p_2^3)$ and thus to satisfy Inv we must have $1 = id(p_2^3) \geq id(p_1^3) = n$. Thus we seek an $n \in \mathbb{N}$ such that $1 \geq n$, but $n \neq 0, 1$, which cannot exist. Thus Inv is uncompletable.

5.2 Leader Election Tree Protocol

As a second protocol, we analyze another system designed to elect a leader in a network. We assume as a pre-requisite that a set of processes are arranged in some undirected tree. The protocol is then described in pseudo-code in Figure 5.3. In words, when a process is scheduled, it will count how many of it's edges remain undirected (clearly, initally all of them). If there are more than one, the process executes a `skip` command. If there is exactly one, it directs that edge away from itself, forming a parent relation. If there are no undirected edges and the process has an outgoing parent relation, it executes a `skip`

[1]Ivy verifications for the examples presented here can be found in Appendix B, or github.com/ashmorer/fopExamples contains both global and local verifications.

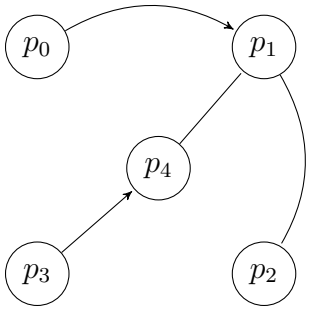
$$
\begin{aligned}
\mathit{Init}: &\quad \mathit{var} \leftarrow \mathit{null} \\
\mathit{Tr}: &\quad \exists p' \neq p'' \cdot \mathit{undir}(p,p') \wedge \mathit{undir}(p,p'') \Rightarrow \texttt{skip} \\
&\quad \exists! p' \cdot \mathit{undir}(p,p') \Rightarrow \mathit{parent}(p,p') \leftarrow \top \\
&\quad \neg\exists p' \cdot \mathit{undir}(p,p') \wedge \exists p' \cdot \mathit{parent}(p,p') \Rightarrow \texttt{skip} \\
&\quad \neg\exists p' \cdot \mathit{undir}(p,p') \wedge \neg\exists p' \cdot \mathit{parent}(p,p') \Rightarrow \mathit{leader} \leftarrow \top \\
\mathit{Bad}: &\quad \exists p,p' \cdot \mathit{leader}(p) \wedge \mathit{leader}(p') \wedge p \neq p'
\end{aligned}
$$

Figure 5.3: A leader election protocol over trees. We use $\exists p' \neq p''$ to denote there exist two distinct elements p', p'', and $\exists! p' \cdot$ to denote that there exists one unique element satisfying the property.

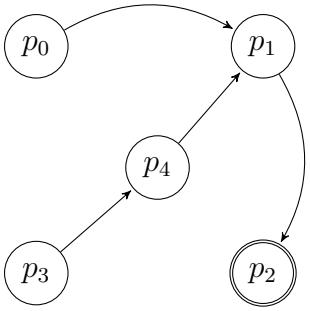
command. Finally, if there are no undirected edges and there is no outgoing parent relation, the process declares itself the leader. Two example states of the protocol are presented in Figure 5.4 Intuitively, each directed edge should be viewed as a parent relation, and the protocol builds a directed tree from the bottom-up. Only when the tree is finished being directed does some node have all edges incident to it directed to it, and there is indeed an unambiguous root (leader) of the tree. Because we require bounded neighbourhoods, we will assume all nodes are degree 3 or less, thus allowing for binary trees.

5.2.1 Modelling of Tree Leader Election

We first define a FOP of this system. Because all edges are treated the same, we use a non-deterministic FOP. First, the topology chosen is $\mathcal{TREE}3 = \{(V,E) \mid |E| = |V| - 1, \forall v \in V \cdot 1 \leq d(v) \leq 3, (V,E) \text{ is a connected undirected graph}\}$. Note that $1 \leq d(v)$ for all v means we are disallowing the singleton tree. We annotate the graphs with $\mathit{edge}(x,y)$ iff $\{x,y\} \in E$, and $\mathit{btw}(x,y,z)$ if y is on the x,z-path. We introduce two state relations $\mathit{parent}(x,y), \mathit{leader}(x)$ to complete the signature for when edges are directed, and when a process declares itself the leader. We use one further helper predicate, $\mathit{ancestor}(x,y)$ to denote the transitive closure of $\mathit{parent}(x,y)$. Notationally, we write $\vec{p}, \vec{q}$ to denote $\{p_1,p_2,p_3\}$ and $\{q_1,q_2,q_3\}$, respectively. We let $\mathit{Mod}(p) = \{p\} \cup \vec{p}$ denote up to three processes (they may be aliasing as one another), each satisfying $\mathit{edge}(p,p_i)$ for $p_i \in \vec{p}$. Finally, we must define $\mathit{Init}(\vec{q})$, $\mathit{TrLoc}(p)$.



(a) Example state of tree protocol. Note that p_0, p_1, p_3 will all `skip` if scheduled, while p_2, p_4 will write p_1 as their parent if scheduled.



(b) Following Figure 5.4a, if the next three processes scheduled are p_4, p_1, and then p_2, then the protocol will elect process p_2 as the leader.

Figure 5.4: A pair of example states of the tree protocol.

$$
\begin{aligned}
Init_{btw}(q_1, q_2, q_3) &\triangleq \bigwedge_{i,j=1}^{3} (\neg parent(q_i, q_j) \wedge \neg leader(q_i)) \\
\tau_{leader}(p) &\triangleq \left(\bigwedge_{i=1}^{3} parent(p_i, p) \right) \iff leader'(p) \\
\tau_{parent}(p, p_i) &\triangleq Unch(parent(p_i, p)) \wedge \\
&\quad (parent'(p, p_i) \Rightarrow \tau_{ancestor}(p, p_i)) \wedge \\
&\quad (\neg parent'(p, p_i) \Rightarrow Unch(ancestor, p, p_i)) \wedge \\
&\quad \left(\left(\neg parent(p_i, p) \wedge \bigwedge_{j \neq i} (p_i = p_j \vee parent(p_j, p)) \right) \iff \right. \\
&\qquad \left. parent'(p, p_i) \right) \\
\tau_{ancestor}(p, p_i) &\triangleq \forall x, y \cdot ancestor'(x, y) \iff \\
&\qquad (ancestor(x, y) \vee (x = p \wedge ancestor(p_i, y)) \vee \\
&\qquad\quad (ancestor(x, p) \wedge y = p_i) \wedge (x = p \wedge y = p_i)) \\
TrLoc(p) &\triangleq \tau_{leader}(p, p_1, p_2, p_3) \wedge \bigwedge_{i=1}^{3} edge(p, p_i) \\
&\quad \bigvee_{i=1}^{3} \left(\tau_{parent}(p, p_i) \wedge \bigwedge_{p_j \neq p_i} Unch(parent(p, p_j), parent(p_j, p)) \right)
\end{aligned}
$$

Initially, we require no parent relations, nor leaders. For transitions, τ_{leader} sets $leader'$ to be true iff it is the parent of all incident processes. The τ_{parent} transition sets a parent relation from p to p_i iff it doesn't already point the other way, and p_i is the only choice for p to make. The transition also calls $\tau_{ancestor}$ to update the ancestor predicate to be the transitive closure when the new edge is added (if it is added). Finally, the overall local transition calls the leader transition τ_{leader}, and non-deterministically chooses $\tau_{parent}(p, p_i)$ for some $p_i \in \vec{p}$, and updates its parent relation. Meanwhile, the $p_j \neq p_i$ are not changed.

Note that one part of our system is not local, namely $\tau_{ancestor}$. If p gains p_i as a parent, then all descendants of p will gain p_i as an ancestor, regardless of how local to p they are. However, this is acceptable since any counter-example to induction (CTI) in the global protocol will still appear only on terms $p, \vec{p}, \vec{q}$. Therefore, a CTI will occur as well when this transition's quantification is just instantiated on $p, \vec{p}, \vec{q}$, and thus our verification will still be sound.

5.2.2 Verification of Tree Leader Election

We wish to investigate whether or not this FOP satisfies our safety property: $\forall x, y \cdot (leader(x) \wedge leader(y)) \Rightarrow x = y$. In order to do this, we seek an invariant and investigate if the following VC's are valid.

$$\forall \vec{q} \cdot CrossInit_{btw}(\vec{q}) \Rightarrow Inv_{btw}(\vec{q}) \tag{5.1}$$

$$\forall p, \vec{p}, \vec{q} \cdot (CrossInv_{btw}(p, \vec{p}, \vec{q}) \wedge \tau(p, \vec{p})) \Rightarrow Inv'_{btw}(\vec{q}) \tag{5.2}$$

In order to make $CrossInit$, $CrossInv$, we construct a sufficiently strong χ_{btw} in a few ways. We first use global axioms to be instantiated on the objects $p, \vec{p}, \vec{q}$. We then provide some local assumptions particular to the objects $p, \vec{p}, \vec{q}$. First, we list our global axioms.

$$\begin{aligned}
\forall x, y \cdot edge(x, y) &\Rightarrow edge(y, x) \\
\forall x, y, z \cdot btw(x, y, z) &\Rightarrow btw(z, y, x) \\
\forall x, y, z \cdot btw(x, y, z) &\Rightarrow distinct(x, y, z) \\
\forall w, x, y, z \cdot (btw(w, x, y) \wedge btw(w, y, z)) &\Rightarrow (btw(w, x, z) \wedge btw(x, y, z)) \\
\forall x, y, z \cdot (edge(x, y) \wedge edge(y, z) \wedge x \neq z) &\Rightarrow btw(x, y, z) \\
\forall w, x, y, z \cdot (edge(w, x) \wedge btw(x, y, z) \wedge w \neq y) &\Rightarrow btw(w, y, z)
\end{aligned}$$

For notation, we will use $x_0 \rightsquigarrow \cdots \rightsquigarrow x_n$ to denote $distinct(\{x_i\}$, and there is a x_0, x_n-path that includes the elements $x_0, \ldots, x_n$ in exactly that order. For the first two, both the $edge$, btw relations are symmetric (as our networks are undirected). Axiom three asserts that btw disallows duplicates. Next, we claim that two sub-path $w \rightsquigarrow x \rightsquigarrow y$ and $w \rightsquigarrow y \rightsquigarrow z$ imply that a $w \rightsquigarrow x \rightsquigarrow y \rightsquigarrow z$ path exists, and thus $btw(w, x, z), btw(x, y, z)$ must both hold. Fifth, two $(x, y), (y, z)$ edges with $x \neq z$ forces the x, z path to go through y, thus $btw(x, y, z)$. Finally, if $x \rightsquigarrow y \rightsquigarrow z$ and w is in the local neighbourhood of x, then we have one of three cases: (1) $w \rightsquigarrow x \rightsquigarrow y \rightsquigarrow z$, (2) $x \rightsquigarrow w \rightsquigarrow y \rightsquigarrow z$, (3) $x \rightsquigarrow w = y \rightsquigarrow z$. When we also assume $w \neq y$, we disallow case (3), and in both cases (1) and (2), we have $btw(x, y, z)$, the only conclusion we assert.

Secondly, we make the following local assumptions:

$$\bigwedge_{p_i \in \vec{p}} edge(p, p_i)$$

$$\forall y \cdot \left(\bigvee_{p_i \in \vec{p}} (btw(p, p_i, y) \vee p_i = y) \right) \vee y = p$$

$$btw(\vec{q})$$

First, we assume the terms p_i are indeed incident to p. Secondly, we assume that any process y is either p, one of the p_i, or down one branch of the p_i (when we view process p as a root). Finally, we make the obvious assumption that the $\vec{q}$ processes we wish to observe satisfy $btw(\vec{q})$.

Finally, we will check the following invariants:

$$\begin{aligned}
&\forall x, y \cdot (leader(x) \wedge leader(y)) \Rightarrow x = y \\
&\forall x, y \cdot ancestor(x, y) \Rightarrow \neg ancestor(y, x) \\
&\forall x, y \cdot leader(x) \Rightarrow \neg ancestor(x, y) \\
&\forall x, y, z \cdot (btw(x, y, z) \wedge ancestor(y, z)) \Rightarrow ancestor(x, y) \\
&\forall x, y, z \cdot (ancestor(x, y) \wedge ancestor(y, z)) \Rightarrow ancestor(x, z) \\
&\forall x, y \cdot parent(x, y) \Rightarrow ancestor(x, y) \\
&\forall x, y \cdot (ancestor(x, y) \wedge edge(x, y)) \Rightarrow parent(x, y) \\
&\forall x, y \cdot (leader(x) \wedge edge(y, x)) \Rightarrow parent(y, x) \\
&\forall x, y, z \cdot (ancestor(x, z) \wedge btw(x, y, z)) \Rightarrow (ancestor(x, y) \wedge ancestor(y, z))
\end{aligned}$$

First, we assert our safety property, that no two processes can be leaders. Next, we assert that the *ancestor* relation is one-directional. Third, no leader has any ancestors. Fourth, if y has z as an ancestor, and y is between x, z, then x must have y as an ancestor already. This is because y should not set a parent until $ancestor(x, y)$ holds. Next we assert that *ancestor* is transitive. The next two say that *parent* forms a base case of *ancestor*. Next-to-last, a leader is a parent to it's local neighbourhood. Finally, if $ancestor(x, z)$, then *ancestor* holds from x to any internal node, and *ancestor* also holds from any internal node to z.

5.2.3 Sanity Checks

Throughout the process of creating the model and verification of the tree leader election, it can be easy for an engineer to accidentally assume $\bot$ (false) in the model. This results in vacuously verifying safety of the protocol. In order to convince ourselves that we have actually modelled and verified the protocol correctly, we discuss a few sanity checks one can take to check that the model is indeed correct. We broadly use two techniques; minimizing the inputs to verification, and modifying the model with intentional bugs.

First, we minimize the inputs to verification. There are two main locations where we make assumptions; *Inv* preconditions and χ_{Top} topological assertions. By removing parts of each and seeing if the file still verifies, we can reduce the verification to a minimal set of topological assumptions χ_{Top} and pre-conditions in *Inv*. If not many topological assumptions are needed, but an engineer suspects the network topology is necessary for correctness of the protocol, then this may suggest a flaw in the verification. Similarly, if not many state pre-conditions need to be assumed (and checked in post-conditions), the state may not be terribly relevant to the correctness of the protocol, which suggests that assumptions on the topology may be excessively strong.

Second, we re-model the system with intentional bugs, while not changing our local assumptions for the network topology, nor the invariant found. The simplest way to modify the transitions is to change $TrLoc(p) \triangleq \top$. In Ivy, this is equivalent to running a $P(\vec{x}) := *$; command for all state predicates P. This says that during every transition, all processes arbitrarily change state. More complex intentional bugs may be added, but they are model-dependant. We outline a few example bugs intentionally introduced in the leader election on tree protocols.

First, in τ_{parent}, we check that $\neg parent(p_i, p)$ holds before we write $parent'(p, p_i)$. If we remove this check, it results in two errors (this matches our intuition, as without this, the protocol may correctly elect a leader p, and then have a root node p fire the $\tau_{parent}(p, p_i)$

action, adding an extra parent node from p to p_i, thus causing two directions of *parent*, potentially causing the p_i node to also declare itself a leader.

Second, in modifying τ_{parent}, we may remove the check that all $p_j \neq p_i$ point to p. This also leads fails, since internal nodes without parent relations may start setting parent relations, rather than building up from the bottom of the tree. It is trivial then to construct sub-paths of the form $u \to v \leftarrow x \to y \leftarrow z$ to, which will easily lead to both v, y considering themself leaders.

Our final informed modification is to update *ancestor* not as the transitive closure of *parent*, but backwards, as if we were trying to define *descendant*. Consider a state where $\mathit{parent}(x, y) \wedge \mathit{ancestor}(x, y)$ holds. Our invariants remain unchanged, so this is an acceptable pre-condition for a transition. Then, if we schedule x to re-write its parent relation, it will do so, but it will also introduce an $\mathit{ancestor}(y, x)$ relation, thus violating our invariant that *ancestor* is anti-symmetric. This is needed, since, as discussed above, we need our directed edges to only go in one direction.

The three bugs described again did indeed result in Ivy coming up with counter-models, as did the simpler modifications of setting any one of the state predicates (*parent*, *ancestor*, *leader*) to arbitrarily change. Some other arbitrary modifications to the transition definition were considered as well, but still verified the protocol. These changes tend to fall into two sets: (a) systems that would not be live, but still retain safety, or (b) remove redundant checks for our particular verification method.

An example of the first case is if τ_{parent} is modified so that we require $\bigwedge_{j \neq i} \mathit{parent}(p_j, p)$, without allowing the option of $p_i = p_j$. Essentially, we only write $\mathit{parent}(p, p_i)$ when p is the parent of all of its neighbours (before, it was all neighbours minus one). This may initially seem like it will trigger bad states, since if $\mathit{parent}(p, p_i)$ requires $\mathit{parent}(p_i, p)$, then we will result in bi-directional *parent* relations. However, this system will actually never make progress, as some *parent* relation must exist before writing any further *parent* relations. Unfortunately for this system, no such initial *parent* relations exist. Fortunately for our model, a system that makes no progress is inherently safe. Unfortunately for our sanity, we are reminded that we have no guarantee of liveness for our model.

The second case is really a modelling quirk of how we choose to model our system within Ivy. In particular, we break our system down into individual local actions such as $\mathit{setParent}(n1, n2)$, $\mathit{trLoc}(p, p1, p2, p3)$, $\mathit{trC}(p, p1, p2, p3, q1, q2, q3)$. The first two correspond to $\tau_{parent}(p, p_i)$, $\mathit{TrLoc}(p)$, and the third corresponds to our inductive VC check (as we take a transition in network topology $\mathcal{C}$). As such, we assume that $\mathit{edge}(p, p_i)$ holds for $i \in \{1, 2, 3\}$ in trC, but then also check that $\mathit{edge}(n1, n2)$ when we reach the *setParent* action. In both places it is appropriate to assume, since it should be a pre-condition to

setParent, and it is a derived consequence in χ_{Top}. However, taken all together, they do result in redundant assumptions. Indeed, if we remove both of these, we result in a model that fails.

Overall, increasing our confidence in the model requires an intimate knowledge of the intent of the protocol. This allows an engineer to be able to make meaningful predictions about which modifications will affect the system's behaviour in which ways.

Chapter 6

Conclusion

6.1 Related Work

Finite-state parameterized verification is undecidable[2]. We have shown how analysis techniques for parametric distributed systems composed of components running on locally symmetric topologies, introduced in [12, 13, 14, 16, 17], can be generalized and applied within a First Order Logic based theorem proving engine.

We based our description of leader election on Ivy's [20]. However, the analysis carried out in Ivy [20] is global, while the analysis given in this thesis is local, where the local structures reason about triples of processes in the ring.

There has been extensive work on proving properties of parametric, distributed protocols. In particular the work in [1] offers an alternative approach to parametric program analysis based on "views". In that work, cut off points are calculated during program analysis. As another example, in [12, 16, 17] the "cut-offs" are based on the program topology and the local structural symmetries amongst the nodes of the process interconnection networks.

The notion of a "cutoff" proof of safety for a parametric family of programs was first introduced by [7]. For example, in [7], if a ring of 3 processes satisfies a parametric property then the property must hold for all rings with at least three nodes. The technique used here is somewhat different; rather than needing to check a ring of 3 processes, we check all pseudo-rings of a given size.

Local symmetry reduction for multi-process networks and parametric families of networks generalizes work on "global" symmetry reduction introduced by [8] and [5]. Lo-

cal symmetry is, in general, an abstraction technique that can offer exponentially more reduction than global symmetry. In particular, ring structures are globaly rotationally symmetric, but for isomorphic processes may be fully-locally symmetric [16, 17].

Recent work [22] has focused on *modular* reasoning in the proof or analysis of distributed systems. In the current work, the modularity in the proof is driven by a natural modularity in the program structures. In particular, for programs of several processes proofs are structured by modules that are local to a neighborhood of one or more processes [12, 16, 17].

6.2 Conclusion

We have presented a framework for specifying protocols in a process-local manner with topology factored out. We show that verification is reducible to FOL with an oracle to answer local questions about the topology. This reduction results in a decidable VC when the background theories are decidable. This cleanly separates the reasoning about the topology from that of the states of the processes.

Many open questions remain. Further work may investigate our methodology on other protocols and topologies, implement oracles for common topologies, and explore complexity of the generated characteristic formulae. Finally, we restricted ourselves to static topologies of bounded degree. Handling dynamic or unbounded topologies, for example in the AODV protocol [15], is left open.

References

[1] Parosh Abdulla, Frédéric Haziza, and Lukáš Holík. Parameterized verification through view abstraction. *International Journal on Software Tools for Technology Transfer*, 18(5):495–516, Oct 2016.

[2] K R Apt and D C Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, May 1986.

[3] Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.

[4] Ernest Chang and Rosemary Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283, May 1979.

[5] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Form. Methods Syst. Des.*, 9(1-2):77–104, August 1996.

[6] Marco Devillers, W. O. David Griffioen, Judi Romijn, and Frits W. Vaandrager. Verification of a leader election protocol: Formal methods applied to IEEE 1394. *Formal Methods in System Design*, 16(3):307–320, 2000.

[7] E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 85–94, New York, NY, USA, 1995. ACM.

[8] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Form. Methods Syst. Des.*, 9(1-2):105–131, August 1996.

[9] Arie Gurfinkel, Sharon Shoham, and Yuri Meshman. Smt-based verification of parameterized systems. In *Proceedings of the 24th ACM SIGSOFT International Symposium*

on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016, pages 338–348, 2016.

[10] K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10, pages 348–370, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] K. Rustan M. Leino and Micha Moskal. Usable auto-active verification, 2010.

[12] Kedar S. Namjoshi and Richard J. Trefler. Local symmetry and compositional verification. In *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, pages 348–362, 2012.

[13] Kedar S. Namjoshi and Richard J. Trefler. Uncovering symmetries in irregular process networks. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 496–514, 2013.

[14] Kedar S. Namjoshi and Richard J. Trefler. Analysis of dynamic process networks. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 164–178, 2015.

[15] Kedar S. Namjoshi and Richard J. Trefler. Loop freedom in aodvv2. In *Formal Techniques for Distributed Objects, Components, and Systems - 35th IFIP WG 6.1 International Conference, FORTE 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings*, pages 98–112, 2015.

[16] Kedar S. Namjoshi and Richard J. Trefler. Parameterized compositional model checking. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 589–606, 2016.

[17] Kedar S. Namjoshi and Richard J. Trefler. Symmetry reduction for the local mu-calculus. In *Tools and Algorithms for the Construction and Analysis of Systems -*

24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II, pages 379–395, 2018.

[18] Susan S. Owicki and David Gries. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976.

[19] Oded Padon, Jochen Hoenicke, Giuliano Losa, Andreas Podelski, Mooly Sagiv, and Sharon Shoham. Reducing liveness to safety in first-order logic. *PACMPL*, 2(POPL):26:1–26:33, 2018.

[20] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 614–630, 2016.

[21] Ruzica Piskac, Leonardo Mendonça de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *J. Autom. Reasoning*, 44(4):401–424, 2010.

[22] Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, Mooly Sagiv, Sharon Shoham, James R. Wilcox, and Doug Woos. Modularity for decidability of deductive verification with applications to distributed systems. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2018, pages 662–677, New York, NY, USA, 2018. ACM.

APPENDICES

Appendix A

Glossary of Symbols and Terms

Bad - A FOL formula capturing which interpretations of a FOTSS are unsafe.

$\mathcal{BTW}$ - The topological class of unidirectional rings over signature $\Sigma = \{btw, next\}$.

btw - A trinary predicate where $btw(x, y, z)$ denotes that x, y, z are distinct, and there is an x, z path passing through y with no repetitions.

B - A set of states of a transition system denoting bad states to not be reached.

$\mathcal{C}$ - A symbol denoting an arbitrary network topology class. This will consist of a set of FO-labelled graphs.

$CrossInit_{Top}$ - A FOL formula capturing when a local neighbourhood of some processes $\vec{q}$ (satisfying *Top*) satisfies all relevant initial conditions.

$CrossInv_{Top}$ - A FOL formula capturing when a local neighbourhood of some processes $\vec{q}$ (satisfying *Top*) satisfies all relevant invariants.

dir - Part of the FOL labelling of first-order graphs. This maps every edge in a given graph to a FO function or predicate in the signature.

distinct - A shorthand predicate denoting that all arguments are distinct.

D - A domain of a FOL interpretation.

EPR - Effectively Propositional Reasoning (also known as the Bernays-Schönfinkel class), a decidable fragment of FOL.

$FOL^{\mathcal{C}}$ - FOL with a semantic restriction on the objects in $\Sigma^{\mathcal{C}}$.

FOL - First Order Logic, an underlying language throughout this paper. It consists of predicates, functions, relations, the connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, and the quantifiers $\forall x \cdot, \exists x \cdot$.

FOCTSS - First Order (relative to topology $\mathcal{C}$) Transition System Specification. A protocol specification relative to a topology. Given a domain to execute on (from $\mathcal{C}$), obtain a FOCTS.

FOCTS - First Order (relative to topology $\mathcal{C}$) Transition System. Given a FOCTSS and a graph in $\mathcal{C}$, one obtains a FOCTS (fundamentally a transition system) with semantics given by the FOCTSS.

FOP - First Order Protocol. A tuple $(\Sigma, \mathit{Init}(\vec{p}), \mathit{Mod}(p), \mathit{TrLoc}(p), \mathcal{C})$ containing signature Σ, local initial states $\mathit{Init}(\vec{p})$, modification set $\mathit{Mod}(p)$, local transition relation $\mathit{TrLoc}(p)$, and network topology $\mathcal{C}$. Semantics are given by a FOCTSS built from the FOP.

FOTSS - First Order Transition System Specification. A specification for how to generate a transition system given an arbitrary first order domain.

FOTS - First Order Transition Sytem. A transition system generated from a FOTSS given a domain.

Frame - A predicate macro denoting that every process out of a local set is unmodified.

G - A graph (typically for this context a graph augmented with a FO-labelling from some topology class $\mathcal{C}$).

Init - A FOL formula capturing the local initial conditions for a FOP.

InvOk - A $FOL^{\mathcal{C}}$ formula whose validity directly corresponds to the inductiveness

of a proposed local invariant.

Inv_{Top} - A (proposed) local invariant for the topological kind Top.

Inv - A FOL formula capturing an invariant, a set of states which (if verified) overapproximate the reachable state space. We often seek inductive invariants. May be local or global, depending on context.

I - A set of states overapproximating the reachable state space of a transition system (also often is inductive).

$\mathcal{I}, \mathcal{J}$ - Symbols denoting arbitrary FOL interpretations. Often when $\mathcal{J}$ is used, it is a $FOL^{\mathcal{C}}$ interpretation built from $\mathcal{I}$.

kind - Part of the FOL labelling of first-order graphs. This maps some k-tuples in a given graph to a FO term in the signature.

Mod - A set of syntactic terms representing the local neighbourhood of a given process.

nbd - A set of (semantic) processes representing the local neighbourhood of a given process.

Proc - A sort of FOP's version of FOL representing which universe elements are processes.

$p, \vec{p}, p^{\mathcal{C}}$ - A process, a tuple of processes (often a triple in our examples), a process further specified to belong to a graph of $\mathcal{C}$

$\mathcal{RBR}$ - A topological class of rings of alternating red and black nodes. This class has the signature $\Sigma^{\mathcal{RBR}} = \{Red, Black, left, right\}$.

R - A relation over states of a transition system denoting which states are allowed to transition where.

`skip` - A symbol representing that a program semantics should be to leave all variables unchanged.

S_0 - A set of initial states of a transition system.

S - A set of states for a transition system.

Top - A symbol representing an arbitrary topological kind.

Transition System - A tuple of (S, S_0, R) with state set S, initial states S_0, and transition relation R.

$\mathcal{TREE}3$ - A topological class of undirected trees with at least one edge and maximum degree less than or equal to three. This captures non-trivial undirected binary trees.

TrLoc - A FOL formula capturing the transition of a process modifying its local neighbourhood.

Unch - A FOL shorthand denoting that all processes contained as arguments should retain the same semantics as the previous state.

UnMod - A FOL shorthand denoting that all predicates and functions contained as arguments should retain the same semantics as the previous state.

VC - Short for Verification Conditions, a set of formulae whose validity implies the inductiveness of an invariant or safety of a system.

π - A path of a transition system, $\pi = s_0 \dots s_i \dots$ for some sequence of states (s_i). May be finite or infinite.

$\Sigma^{\mathcal{C}}$ - A signature of a network topology class. It decomposes into two objects, $\Sigma^{\mathcal{C}}_E \uplus \Sigma^{\mathcal{C}}_T$, respectively, the edge labellings and topological kind labellings, of the topology $\mathcal{C}$.

Σ - A signature of FOL. A set of functions, predicates, and terms.

τ - A FOL formula capturing a global transition relation. Sometimes $\tau(p)$ is used to denote a local transition occuring with $\mathit{TrLoc}(p) \wedge \mathit{Frame}(p)$.

φ_0 - A FOL formula capturing the set of global initial states.

$\chi_{Top}(A, \vec{q})$ - A FOL formula intuitively expressing what is derivable about $A, \vec{q}$ given that $Top(\vec{q})$ holds. The common cases are $A = \emptyset$ and $A = Mod(p)$.

$\emptyset$ - The empty set, a set consisting of no elements.

$\models_{\mathcal{C}}$ - The models symbol, but with all interpreations restricted to be those allowed in $FOL^{\mathcal{C}}$.

$\models$ - The models symbol. Writing $\mathcal{I} \models \varphi$ means that interpretation $\mathcal{I}$ evaluates FOL formula φ to be true. Writing $\models \varphi$ means all FOL $\mathcal{I}$ satisfy $\mathcal{I} \models \varphi$.

$\oplus$ - A symbol used to denote the merging of two non-conflicting FOL interpretations.

$\uplus$ - A symbol denoting disjoint union of two sets.

Appendix B

Ivy Code for Verifying FOPs

B.1 Red-Black Rings Example Protocol

This appendix section contains the code for verifying the example $\mathcal{RBR}$ protocol locally.

```
#lang ivy1.7

type node
type colour = {0,r,b}

axiom 0~=r & 0~=b & r~=b

#Give colours to the processes
relation black(X:node)
relation red(X:node)

#define left/right
individual left(X:node) : node
individual right(X:node): node

#give var field to r/b nodes
individual var1(X:node) : colour
```

```
after init
{
    var1(X) := 0;
}

action t(n1:node,n2:node) = {
    assume right(n1)=n2;
    if(black(n1))
    {
        var1(n2) := r
    }
    else
    {
        var1(n2) := b
    }
}

action trRed(p:node, q:node) =
{
    #limit ourselves to small models
    assume N=p | N=left(p) | N=right(p) | N=q;
    assume red(N) <-> ~black(N);
    assume red(p) -> black(right(p));

    assume red(q);
    call t(p,right(p));
}

action trBlack(p:node, q:node) =
{
    #limit ourselves to small models
    assume N=p | N=left(p) | N=right(p) | N=q;
    assume red(N) <-> ~black(N);
    assume red(p) -> black(right(p));

    assume black(q);
    call t(p,right(p));
}
```

```
export trRed
export trBlack

#Safety property
conjecture [safety] red(X) -> var1(X)~=b
```

B.2 Ring Leader Election

This appendix section contains the code for verifying leader election on rings locally.

```
#lang ivy1.7
#axiomatize total order for ids
module total_order(r) =
{
    axiom r(X,X)                          #reflexivity
    axiom r(X,Y) & r(Y,Z) -> r(X,Z)       #transitivity
    axiom r(X,Y) & r(Y,X) -> X = Y        #anti-symmetry
    axiom r(X,Y) | r(Y,X)                 #total order
    axiom r(0,X)                          #minimal element 0
}

type node
type id

#Put nodes on an uninterpreted ring
individual next(X:node) : node
relation btw(X:node, Y:node, Z:node)

relation le(X:id, Y:id)
instantiate total_order(le)

#Get personal id idn and comparison id comp.
individual idn(X:node) : id
individual comp(X:node): id

axiom idn(X) ~= 0
#unique id's
axiom idn(X)=idn(Y) -> X=Y

after init
{
    comp(X) := 0;
}
```

```
action trLoc(n1:node,n2:node) = {
    assume next(n1)=n2;
    if le(idn(n1),comp(n1))
    {
        #id(n)<=comp(n1), pass comp(n1)
        comp(n2) := comp(n1);
    }
    else
    {
        #id(n1)>comp(n1), pass idn(n1)
        comp(n2) := idn(n1);
    }
}

action trC(p:node, q1:node, q2:node, q3:node) =
{
    #limit ourselves to small models
    assume N=p|N=next(p)|N=q1|N=q2|N=q3;

    #finite instantiation of ring axioms
    assume ((W=p|W=next(p)|W=q1|W=q2|W=q3) &
            (X=p|X=next(p)|X=q1|X=q2|X=q3) &
            (Y=p|Y=next(p)|Y=q1|Y=q2|Y=q3) &
            (Z=p|Z=next(p)|Z=q1|Z=q2|Z=q3)) ->
            (   (btw(X,Y,Z) -> btw(Y,Z,X)) &
                (btw(X,Y,Z) -> ~btw(X,Z,Y)) &
                ((X~=Y & X~=Z & Y~=Z) -> (btw(X,Y,Z) | btw(X,Z,Y))) &
                ((btw(W,X,Y) & btw(W,Y,Z)) -> btw(W,X,Z)));
    #next axiom instantiated on p,next(p),qi
    assume q1~=p & q1~=next(p) -> btw(p,next(p),q1);
    assume q2~=p & q2~=next(p) -> btw(p,next(p),q2);
    assume q3~=p & q3~=next(p) -> btw(p,next(p),q3);

    assume  btw(q1,q2,q3);
    call trLoc(p,next(p));
}
```

```
export trC

# The safety property
conjecture [safety]  btw(X,Y,Z) -> ((comp(X) = idn(X) & comp(Y) = idn(Y)) -> X = Y)

#helper invariants
conjecture [maxId] btw(X,Y,Z) -> (comp(X) = idn(X) -> (le(idn(Y),idn(X)) & le(idn(Z),id
conjecture [bypass] btw(X,Y,Z) -> (idn(X)=comp(Z) -> le(idn(Y),idn(X)))
```

B.3 Tree Leader Election

Below is the code for the core verification of the leader election on tree topologies, as well as the modification structure used for sanity testing.

```
#lang ivy 1.7

type node

# true iff there is an edge between X and Y
relation edge(X:node, Y:node)

# edge is symmetric and has no self-loops
axiom edge(X,Y) -> edge(Y,X)
axiom ~edge(X, X)

# true iff the edge between X and Y is directed X -> Y (Y is X's parent)
relation parent(X:node, Y:node)

# true iff the direction of all the parent relations on the X,Y-path is X --> Y
# (X has Y as an ancestor)
relation ancestor(X:node, Y:node)

# true if X is a leader
relation leader(X:node)

# true if (a) distinct(X, Y, Z) , and (b) Y is on the unique path between X and Z
relation btw(X:node, Y:node, Z:node)

#btw is symmetric
axiom (btw(X,Y,Z) -> btw(Z,Y,X))
axiom btw(X,Y,Z) -> (X~=Y & X~=Z & Y~=Z)
#axiom btw(X,Y,Z) -> ~btw(X,Z,Y)
#axiom btw(X,Y,Z) -> ~edge(X,Z)
axiom (btw(W,X,Y) & btw(W,Y,Z)) -> (btw(W,X,Z) & btw(X,Y,Z))
axiom (edge(X,Y) & edge(Y,Z) & X~=Z) -> btw(X,Y,Z)
axiom (edge(W,X) & btw(X,Y,Z) & W~=Y) -> btw(W,Y,Z)
#either WXYZ or XWYZ, either way, WYZ holds
```

```
after init
{
    assume ~parent(X,Y);
    assume ~ancestor(X, Y);
    assume ~leader(X);
}

action setLeader(p:node) = {
    if forall Z:node . (edge(p, Z) -> parent(Z, p))
    {
        leader(p) := true;
    }
}

action setParent(n1:node, n2:node) =
{
    assume edge(n1, n2) & n1 ~= n2;
    #if n2 could be n1's parent and has n2 is n1's unambiguous choice for parent.
    if ~parent(n2, n1) & forall X:node . (X ~= n2 -> (edge(X, n1) -> parent(X, n1)))
    {
        parent(n1, n2) := true;
        ancestor(X,Y) := ancestor(X,Y) | (ancestor(X,n1) & Y=n2) |
            (X=n1 & ancestor(n2, Y)) | (X=n1 & Y=n2);
    }
}

#this action is a bug if it is any of the following single modifications are made:
#n1~=n2 not assumed -> OK
#edge(n1, n2) not assumed -> OK
#~parent(n2, n1) not checked -> 2 errors
#edge(X, n1) -> parent(X, n1) for n2 as well -> OK
#no edge(X, n1) check -> OK
#parent(X, n1) replaced by true -> 1 fail
#parent(X, n1) replaced by parent(n1, X) -> OK
#parent(n1,n2) := *; -> 2 fails
#ancestor updated backwards -> 5 fails
```

```
action setParentBug(n1:node, n2:node) =
{
    assume edge(n1, n2) & n1~=n2;
    if ~parent(n2, n1) & forall X:node . (X~=n2 -> (edge(X, n1) ->  parent(X, n1)))
    {
        parent(n1, n2) := true;
        ancestor(X,Y) := ancestor(X,Y) | (ancestor(X,n1) & Y=n2) |
            (X=n1 & ancestor(n2, Y)) | (X=n1 & Y=n2);
        #ancestor(X,Y) := ancestor(X,Y) | (ancestor(X,n2) & Y=n1) |
            (X=n2 & ancestor(n1, Y)) | (X=n2 & Y=n1);
    }
}

action trLoc(p:node, p1:node, p2:node, p3:node) =
{
    call setLeader(p);
    if *
    {
        if *
        {
            #call setParent(p, p1);
            call setParentBug(p, p1);
        }
        else
        {
            call setParent(p, p2);
        }
    }
    else
    {
    call setParent(p, p3);
    }
}

action trLocBad(p:node, p1:node, p2:node, p3:node) =
{
    #all commented -> OK
    #Each line alone generates 3, 7, 3 fails respectively
```

```
    #all together, 9 fails
    parent(X, Y) := *;
    ancestor(X, Y) := *;
    leader(X) := *;
}

action trC(p:node, p1:node, p2:node, p3:node, q1:node, q2:node, q3:node) = {

    #local neighbourhood of p
    assume edge(p, p1) & edge(p, p2) & edge(p, p3);

    #all processes y are in neighbourhood of p, or
    # are on p1, p2, or p3 branch of tree.
    assume  (btw(p,p1,Y) | btw(p,p2,Y) | btw(p,p3,Y) | Y=p1 | Y=p2 | Y=p3 | Y = p);

    #observing neighbourhood q1,q2,q3 satisfying btw
    assume btw(q1,q2,q3);

    call trLoc(p,p1,p2,p3);
}

export trC



conjecture [safety] (leader(X) & leader(Y)) -> X=Y

#helpers
conjecture [oneDir] ancestor(X,Y) -> ~ancestor(Y,X)
conjecture [leadNoFollow] leader(X) -> ~ancestor(X,Y)
conjecture [ancBtw] (btw(X,Y,Z) & ancestor(Y,Z)) -> ancestor(X,Y)
conjecture [trans] (ancestor(X,Y) & ancestor(Y,Z)) -> ancestor(X,Z)

conjecture [parentAncestor] parent(X,Y) -> ancestor(X,Y)
conjecture [parentEdge] (ancestor(X,Y) & edge(X,Y)) -> parent(X,Y)
conjecture [leadLeads] (leader(X) & edge(Y,X)) -> parent(Y,X)
conjecture [ancBtw2] (ancestor(X,Z) & btw(X,Y,Z)) -> (ancestor(X,Y) & ancestor(Y,Z))
```

Appendix C

Axiomatization Proofs

C.1 Finite Models and Ring Axioms

Let *Ring* be the conjunction of the ring axioms given in Figure 1.1.

Theorem C.1.1. *Let $\mathcal{I} \models Ring$ and $\mathcal{I}$ have a finite domain D. Define the graph $G = (D, E)$, where $(u, v) \in E$ iff $\mathcal{I} \models next(u, v)$. Then G is a ring graph.*

Targeting this theorem, we first work through a lemma asserting that if $\mathcal{I} \models Ring$, then $btw(x, y, z)$ is only ever satisfied by distinct elements, namely that $\forall a, b, c : btw(a, b, c) \rightarrow a \neq b \wedge a \neq c \wedge b \neq c$

Proof. Let $a, b, c \in D$ and $I \vDash btw(a, b, c)$. Suppose for contradiction that two elements are the same. Case $a = b$: Apply axiom one twice for $btw(c, a, b)$. Case $a = c$: Apply axiom one once for $btw(b, c, a)$. Case $b = c$: Conclude $btw(a, b, c)$.

In any of the above cases, we write $btw(x, y, y)$ for the two duplicate values. Then we apply axiom three to get $\neg btw(x, y, y)$, a contradiction. Thus no duplicates may exist, and all three are distinct. □

And now, the main proof:

Proof. We first show that every node has out-degree at least one. Then we show that every node has out-degree at most one. Finally, we show that the graph is connected.

Let D, I be a finite domain and interpretation and $I \models Ring$. As a temporary goal, we will show that $\forall a.\exists b.next(a,b)$. Let $a \in D$. If $|D| = 1$, then $next(a,a)$ is vacuously true since there are no x to violate the condition. Similarly, $|D| = 2$ vacuously has $next(a,b)$ and $next(b,a)$ for the two elements.

Let $a \in D$ and $2 < |D| < \infty$. Since there are at least three elements, choose distinct b_0, b_1. Since they are distinct, the fourth axiom gives $btw(a,b_0,b_1) \vee btw(a,b_1,b_0)$. Renaming b_0, b_1 without loss of generality, we assume $btw(a,b_1,b_0)$.

Let b_i satisfy $btw(a,b_{i+1},b_i)$ for distinct a, b_i, b_{i+1}. Suppose there exists x such that $x \neq a \wedge x \neq b_{i+1} \wedge \neg btw(a,b_{i+1},x)$. Then a, b_{i+1}, x are all distinct, so we observe that $btw(a,b_{i+1},x) \vee btw(a,x,b_{i+1})$. Since one has already been ruled out, we conclude $btw(a,x,b_{i+1})$. Set $b_{i+2} := x$. Then $btw(a,b_{i+2},b_{i+1})$ is satisfied, and must thus have distinct values.

Iterate the process above so long as there exists such an x. Does such a process terminate? If so, we have guaranteed existence of a b such that $next(a,b)$. If not, we obtain a sequence b_i such that $btw(a,b_{i+1},b_i)$ for all i in the sequence. Thus there must be some $b_j = b_k$ for $j < k$ since we have an infinite sequence over a finite domain D.

$$btw(a,b_{j+1},b_j)$$
$$btw(a,b_{j+2},b_{j+1})$$
$$btw(a,b_{j+2},b_j)$$

Iterating this argument, we eventually find out that

$$btw(a,b_k,b_j)$$

But from our lemma, we then know that $b_k \neq b_j$, contrary to our earlier conclusion. Ergo, the process must terminate.

We have thus established the claim $\forall a : \exists b : next(a,b)$. We next seek to establish uniqueness. Namely, we aim to show $\forall a,b,c : next(a,b) \wedge next(a,c) \rightarrow b = c$.

Let $a, b, c \in D$ and $next(a,b) \wedge next(a,c)$. Suppose for contradiction that $b \neq c$. Then by axiom 4, we conlcude $btw(a,b,c) \vee btw(a,c,b)$.

Suppose $btw(a,b,c)$. We know $next(a,c)$. Thus $\forall x : x \neq a \wedge x \neq c \rightarrow btw(a,c,x)$. Instantiating $x = b$, we satisfy the LHS of the implication, thus $btw(a,c,b)$ But axiom three then says $\neg btw(a,b,c)$ against our initial assumption.

Suppose $btw(a, c, b)$. A similar argument on $next(a, b)$ gives a contradiction. Thus either case leads to a contradiction and we conclude that $b = c$.

Thus given a, there exists a unique successor. We now claim that the graph is connected, so that for every $u, v \in D$, there exists a sequence u_i such that $u_0 = u, u_n = v$, and $I \vDash next(u_i, u_{i+1})$ for all $i \in \{0, \ldots, n-1\}$.

If $|D| = 1$, we have $next(a, a)$ vacuously true, thus have a cycle. Further, $|D| = 2$ is trivial to solve, ($btw(x, y, z)$ always false since duplicates not allowed, and $next(a, b), next(b, a)$ is a cycle) , so we assume $|D| > 2$.

Let $u, v \in D$ and $|D| > 2$. Since $|D| > 2$, choose third distinct element x. Then $btw(u, x, v) \vee btw(u, v, x)$. If $next(u, v)$, then we have a trivial path. Suppose $\neg next(u, v)$. Then consider $next(u, w)$. We have $\forall x.x \neq u \wedge x \neq w \rightarrow btw(u, w, x)$. Instantiate with $x = v$. Then we get $btw(u, w, v)$.

We now claim that if $btw(a, b, c) \wedge next(a, b)$, then there exists an a, c path. Suppose that $btw(a, b, c) \wedge next(a, b)$. Then set $b_0 = b$. Note that the following property holds for $n = 0$.

Property: if $btw(a, b_n, c)$, for all $i \in [0, n]$, the b_i are distinct values in D, then there exists an a, b_n-path, *and* for all $i \in [0, n-1]$, $btw(a, b_i, b_{i+1})$. Consider the successor of b_n, say $next(b_n, b_{n+1})$. If $b_n = c$ or $b_{n+1} = c$, we have our a, c path. Else, we observe that $next(b_n, b_{n+1})$ can be instantiated with $x = c$ to get $btw(b_n, b_{n+1}, c)$. Note that $btw(a, b_{n-1}, b_n)$ gives us $btw(a, b_{n+1}, c)$, our first condition. Our initial assumption by axiom 1 also gives $btw(b_n, c, a)$. Thus we conclude that (by axiom 2) $btw(b_n, b_{n+1}, a)$. Reapplying axiom 1, we get $btw(a, b_n, b_{n+1})$. If $b_{n+1} = b_j$ for some $j \in [0, n-1]$, then we get a contradiction since we have already derived $btw(a, b_j, b_{j+1}), btw(a, b_{j+1}, b_{j+2}), \ldots, btw(a, b_{n-1}, b_n)$, which by applying axiom 2 many times over, gives $btw(a, b_j, b_n)$. Then we also have $btw(a, b_{n+1}, b_n)$, which contradicts axiom 3. Hence the new b_i is distinct from all the others, our second condition. By construction of b_{n+1}, we easily have an a, b_{n+1} path, and we derived $btw(a, b_n, b_{n+1})$ along the way. Thus our induction holds.

We generate a sequence of b_i with the above properties. If we never have $b_i = c$, then we have an infinite sequence of distinct values, but we have a finite domain. Ergo, this cannot happen, and there must be some $b_i = c$. This completes the proof.

Suppose $G = (V, E)$ is the induced graph where $V = D$ and $E = \{(a, b) | next(a, b)\}$. Suppose G is connected and $\forall v \in V : d(v) = 1$. Then G is a cycle.

□