# REWRITING-BASED SYMBOLIC METHODS
# FOR DISTRIBUTED SYSTEM VERIFICATION

BY

STEPHEN SKEIRIK

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:

Professor José Meseguer, Chair
Professor Gul Agha
Professor Grigore Roşu
Professor Peter Ölveczky, University of Oslo

# ABSTRACT

As computer system complexity increases, new methods and logics are needed to scale up to the complexity of practical systems without sacrificing logical precision and ease of specification. To that end, the goal of this research project is to develop rewriting-based symbolic analysis methods that (1) can analyze systems which need an unbounded amount of time and/or space (2) may be highly distributed (3) use modular specification techniques so that work is never wasted (4) are generic across a possibly infinite number of domain theories. Towards this goal, we present our research on theory-generic satisfiability and rewrite-theory-generic specification and analysis methods, discuss prototype implementations, and consider future directions.

# ACKNOWLEDGMENTS

A thesis is not merely the distillation of years of research effort. In its pages, we find a story: the metamorphosis of the wide-eyed student, our protagonist, into a full-fledged researcher. But like all adventurers, our protagonist's success depends on a broad cast of supporting characters. Though they may only appear for a chapter, their impact is unmistakable, and as such, they must be honored for their contributions.

Firstly, I would like to thank my thesis advisor, José Meseguer, for guiding me, believing in me, and insisting on excellence. When I first met you, you were my boss—later, you became my mentor—now, I consider you my friend. I would also like to thank my thesis committee: Gul Agha, Grigore Roșu, and Peter Ölveczky for taking time from your busy schedules to support this work and for your excellent suggestions for improvement. I am indebted to a great group of research collaborators (in alphabetical order): Rakesh Bobba, Indranil Gupta, Raúl Gutiérrez, Si Liu, Muntasir Raihan Rahman, Camilo Rocha, Ralf Sasse, and Andrei Ștefănescu. My thanks goes to all of the other members of the Formal Methods and Declarative Languages Laboratory that supported me in one way or another: Musab Al-Turki, Kyungmin Bae, Andrew Cholewa, and Fan Yang. Special thanks to my friends and officemates Liyi Li and Atul Sandur for mid-afternoon coffee runs and late-night discussions. I am also deeply grateful for the grants that kept food on my table, specifically, NSF grants CNS 13-19109 and CNS 14-09416, AFOSR contract FA8750-11-2-0084, and NRL contract N00173-17-1-G002.

Secondly, I would like to thank my entire church family at the Vineyard Church of Central Illinois. I am especially grateful to Stephen Pety and Wayne Chang for befriending me soon after arriving in Urbana-Champaign and to all of the original members of the Wave (Wayne Chang, Jared Eakins, and Drew Randle) for our many years together. Many thanks to all of the dear friends from both the Pety small group and as well as the Barth/Houmes/Randle group; you were a constant joy and source of strength.

Thirdly, I would like to take a moment to thank those that supported me well before my postgraduate research career began. To Masood Parang from the UTK College of Engineering, please accept my sincerest thanks for finding ways to fund for me through all four years of college. To Drs. Micah Beck and Christopher Brumgard from the UTK Dept. of Computer Science, thank you so much for letting me work with both of you on my very first research experiences as both a high school and undergraduate student. To Kristin Baksa at FHS, a huge thank you for jump-starting my scientific career by encouraging me to participate in the unique high school research programs that you helped to foster which directly led to meeting Micah, Christopher, and Masood. The rest, as they say, is history.

Fourthly, I would like to thank my dear friends and family. To Alex Paradies and Nate Henry, thank you for many rich years of friendship and for many more to come. To my brothers: Samuel, Luke, Ian, David, and Elijah, I adore each of your unique personalities and giftings; family has been so fun with each of you. To my dearest sister, Hadassah, I love you and will pray for you always. To my parents: Robert and Carol, you are the most amazing parents that a son could ever ask for. Words cannot express my gratitude to both of you for continually investing in me since day one.

Finally, all thanks and honor be to the Father God, the Christ Jesus, and the Holy Spirit, who created all things and who sustain all things. Let my heart ever burn for you.

## TABLE OF CONTENTS

# CHAPTER 1 INTRODUCTION

In order to allow formal methods to scale up to specify and analyze real-world distributed systems, appropriately expressive logics and associated tools are needed. To begin with, current approaches to verification may be inadequate. For example, engineers at Amazon Web Services found that:

> "standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems." [1]

Furthermore, according to [2], current practitioners using formal methods still face several challenges: (i) specification time is increased (ii) using formal method-based tooling (e.g. SMT solvers, theorem provers) may require too much time/effort (iii) formal method-based tooling may be unable to express desired properties.

To attack challenge (i), (a) *expressive* and *intuitive* specification methods are needed with support for good *modularity* properties, so that different pieces of a specification may be designed and verified and then composed. Challenge (ii) can be addressed by (b) tightly integrating formal-methods into the *entire* design/verification process; and (c) designing logics/tools with a tunable *abstraction dial* so that only the minimum amount of detail necessary is used to complete a specification/verification task. Finally, challenge (iii) can be alleviated by (d) providing composable, *theory-generic* methods so that practitioners are not limited to a fixed bag of theoretical tricks, so to speak, during specification and analysis.

We believe that the combination of rewriting logic and Maude as an executable specification logic and interpreter is a good candidate to address the three challenges above: (a) a large number of concurrent logics/process calculi and semantic approaches may be mapped into rewriting logic (and thus Maude) with a minimal representation distance [3],[4], (b) a wide array of tools are available directly within Maude via the Maude Formal Environment [5]; and (c) through its use of both equations and rewrite rules as well as strong support for parameterized modules, Maude has a built-in abstraction dial allowing for drastic state space reductions [6]. In particular, we believe *symbolic* rewriting-based methods are particularly promising approaches for specifying and analyzing real-world distributed systems. Furthermore, (d) we will show how symbolic rewriting-based approaches to verification provide a specifier with great flexibility by supporting reasoning about *any* theory subject to a few syntactic and easily checkable requirements. Aside from the symbolic nature of our work, another recurring theme is using *semantics-preserving* signature and rewrite theory *transformations* to shift our ground and effectively attack a problem from a new vantage point.

In this thesis we will examine three separate symbolic rewriting-based methods we have developed over the past few years, namely: (1) order-sorted pattern operations, (2) variant satisfiability, and (3) constructor-based reachability logic.

**Pattern Operations.** As a first example of the symbolic "nature" of rewriting-based methods, consider term patterns. Term patterns are used everywhere in functional and logic programming: to define predicates and functions, to perform automated deduction tasks like rewriting, matching, unification, resolution, and Knuth-Bendix completion, and also as a *symbolic notation* to describe *languages* as sets of term instances, and *language operations* by corresponding symbolic operations on the term patterns defining them. Such pattern operations, including pattern intersection, union, and difference, were first systematically studied by Lassez and Marriott in [7] and further studied in, e.g., [8, 9, 10, 11] have many applications to, e.g., machine learning, negation in logic programming, inductive theorem proving, etc... These original papers focused primarily on the *untyped* case. However, for greater expressiveness, many declarative languages—such as OBJ [12], CafeOBJ [13], and Maude [6]—support rich type disciplines, and we would like our pattern operations to provide support for these languages. A question that we will ask and answer is: can these untyped algorithms be lifted to the order-sorted case? In fact, we will see that the problem, as originally formulated, is intractable, but that through a suitable *signature transformation* $\Sigma \mapsto \Sigma^{\#}$, the problem has a quite natural solution. In fact, this transformation is just a new method to reduce order-sorted signatures into equivalent *many-sorted* signatures; using this transfer principle, we will: (a) reduce validity of a first-order formula in an initial *order-sorted* algebra to the validity of an associated formula in a corresponding *many-sorted* initial algebra; since the first-order theory of a many-sorted initial algebra is known to be decidable [14, 15, 16], we obtain a new proof of decidability that is much simpler than previous attempts [17, 18]; and (b) develop a new *order-sorted* algorithm for pattern operations based on the signature $\Sigma \cup \Sigma^{\#}$; prove its correctness; show how it can be reduced to *many-sorted* pattern operations

**Metalevel Algorithms for Variant Satisfiability.** Moving beyond symbolic operations at the syntactic level, we can consider more *semantic* symbolic methods. Recently, one quite popular approach is satisfiability modulo theories (SMT). SMT solving is at the heart of some of the most effective theorem proving and infinite-state model checking formal verification methods that can scale up to impressive verification tasks. A current limitation, however, is its *lack of extensibility*: current SMT solvers support a (typically small) library of decidable theories. Although these theories can be combined by the Nelson-Oppen (NO) [19, 20] or Shostak [21] methods under some conditions, only the theories in the SMT solver library and their combinations are available to the user: any other theories extending the tool must be implemented by the tool builders.

In practice, of course, the problem a user has to solve may not be expressible by the theories available in an SMT solver's library. Therefore, the goal of making SMT solvers *user-extensible*, so that a *user* can easily *define* new decidable theories and use them in the verification process is highly desirable. Recall that *E-unifiability* is a well-known *subproblem* of SMT solving, namely, satisfiability in the free $(\Sigma, E)$-algebra $T_{\Sigma/E}(X)$ on countably many variables $X$, but restricted to *positive* (i.e., negation-free) quantifier-free (QF) formulas. Until recently, unification tools also suffered from a lack of extensibility: such tools usually support a small library of theories $(\Sigma, E)$, combinable by methods similar to the NO method ([22] explicitly relates the NO method and combination algorithms for unification). Again, the *user* could not extend such *decidable* unifiability/unification algorithms by defining new theories and using a *theory-generic* algorithm. However, true user-extensibility has now been achieved for $E$-unification in theories $(\Sigma, E)$ satisfying the *finite variant property* (FVP) [23] thanks to *variant unification* based on *folding variant narrowing* [24]. In fact, variant unification for user-definable FVP theories is already supported by Maude 2.7.1.

This suggests an obvious question: could variant unification be generalized to *variant satisfiability*, so that, under suitable conditions on and FVP theory $(\Sigma, E)$, satisfiability of QF formulas in the initial algebra $T_{\Sigma/E}$ becomes *decidable* by a *theory-generic* satisfiability algorithm? This would then make satisfiability *user-extensible* as desired. This question has been positively answered in [25] by giving general conditions under which satisfiability of QF formulas in the initial algebra $T_{\Sigma/E}$ of an FVP theory $(\Sigma, E)$ is decidable. Suppose that: (i) the convergent rewrite theory $\mathcal{R} = (\Sigma, B, R)$ is a so-called FVP decomposition of $(\Sigma, E)$ (which is what it means for $(\Sigma, E)$ to be FVP), (ii) $B$ has a finitary $B$-unification algorithm, and (ii) $\mathcal{R}$ has an *OS-compact* constructor decomposition $\mathcal{R}_\Omega$ (definition in Section 4.2). Then satisfiability of QF formulas in $T_{\Sigma/E}$ is decidable by a *theory-generic* algorithm called variant satisfiability.

However, the results in [25] do not really provide an *algorithm* in the full sense of the word, but rather a theoretical *skeleton* on which such an algorithm can be fleshed out. Specifically, they *assume* that the constructor decomposition $\mathcal{R}_\Omega$ is *OS-compact*, but do not provide a way to *automate* both the checking of OS-compactness and the implementation of the various *auxiliary functions* needed for variant satisfiability based on OS-compactness. They also use the notions of *constructor variant* and *constructor unifier* (see Section 4.2), but give only their theoretical definitions instead of algorithms to compute them. Thus, we will define new algorithms to check for OS-compactness as well as to computer constructor variants and constructor unifiers. Here we see again how we can use signature transformations in order to reduce the complex problem of computing constructor variants/unifiers into a simpler one (in this case, unification).

**Constructor-Based Reachability Logic.** The final symbolic method we consider in this thesis will be constructor-based *reachability logic*. The main applications of reachability logic to date have been as a *language-generic* logic of programs [26, 27]. In these applications, a $\mathbb{K}$ specification of a language's operational semantics by means of rewrite rules is assumed as the language's "golden semantic standard," and then a correct-by-construction reachability logic for a language so defined is automatically obtained [27]. This method has been shown effective in proving a wide range of programs in real programming languages specified within the $\mathbb{K}$ Framework.

Although the foundations of reachability logic are very general [26, 27], they do not provide a straightforward answer to the following non-trivial questions: (1) Could a reachability logic be developed to verify not just conventional programs, but also *distributed system designs and algorithms* formalized as *rewrite theories* in rewriting logic [28]? And (2) if so, what would be the most natural way to conceive such a *rewrite-theory-generic* logic?

Although a first step towards a reachability logic for rewrite theories was taken in [29], a few important questions have been left unanswered. Firstly, how can we prove *invariants* of a distributed system? Since invariants are the most basic safety properties, support for proving invariants is a *sine qua non* requirement. We will show that the naive approach is unworkable, but by a simple *theory transformation*, the problem has a quite natural solution. Another important open question is how to best take advantage of the wealth of equational reasoning techniques such as matching, unification, and narrowing modulo an equational theory $(\Sigma, E)$, e.g., [30, 24], and of some recent results on decidable satisfiability of quantifier-free formulas in initial algebras, e.g., [31, 25] to further *automate* reachability logic deduction. A third important issue is *simplicity*. Reachability logic as originally defined has eight inference rules [26, 27]. Could a reachability logic for rewrite theories be simpler? We will tackle head on these three questions to provide a general reachability logic suitable for reasoning about properties of *both* distributed systems and programs based on their rewriting logic semantics.

The plan of the thesis is as follows. We first review some preliminary notions; this section is labeled so that familiar readers may skip ahead if desired. The body chapters cover:

1. order-sorted pattern term operations; boolean algebras over term patterns; term patterns as a method to reduce order-sorted problems to many-sorted problems

2. meta-level algorithms to support a *theory-generic*, *parameteric*, and *composable* satisfiability method based on the notions of variant, variant unification, and variant satisfiability

3. a new, simplified, *rewrite-theory-generic* semantics and proof system for constructor-based reachability

logic using symbolic methods developed in earlier chapters that has been mechanized and tested on simple examples;

4. our capstone project, verification of security properties of the Illinois Browser Operating System (IBOS) as reachability properties.

We conclude with a discussion of our current research and possible topics for future work.

# CHAPTER 2 PRELIMINARIES

We present some preliminaries on order-sorted algebra, rewriting logic, variants, first-order equational formulas, and order-sorted signature morphisms. The material is adapted from [28] and [32], generalizing [33]. The presentation is self-contained: we only assume familiarity with many-sorted signatures and algebras, e.g., [34].

## 2.1 ORDER-SORTED ALGEBRA

**Definition 2.1** *An* order-sorted signature *is a triple* $\Sigma = (S, \leqslant, \Sigma)$ *with* $(S, \leqslant)$ *a poset and* $(S, \Sigma)$ *a many-sorted signature.* $\hat{S} = S/{\equiv_{\leqslant}}$*, called the set of* connected components *of* $(S, \leqslant)$*, is the quotient of* $S$ *under the equivalence relation* $\equiv_{\leqslant} = (\leqslant \cup \geqslant)^+$*. The order* $\leqslant$ *and equivalence* $\equiv_{\leqslant}$ *are extended to sequences of the same length in the usual way, e.g.,* $s'_1 \ldots s'_n \leqslant s_1 \ldots s_n$ *iff* $s'_i \leqslant s_i$, $1 \leqslant i \leqslant n$*.* $\Sigma$ *is called* sensible *(resp.* monotonic*) if for any two operators* $f : w \to s, f : w' \to s' \in \Sigma$*, with* $w$ *and* $w'$ *of same length, we have* $w \equiv_{\leqslant} w' \;\Rightarrow\; s \equiv_{\leqslant} s'$*. (resp.* $w \geqslant w' \;\Rightarrow\; s \geqslant s'$*). Note that a* many-sorted signature $\Sigma$ *is the special case in which the poset* $(S, \leqslant)$ *is discrete, i.e.,* $s \leqslant s'$ *iff* $s = s'$*. For connected components* $[s_1], \ldots, [s_n], [s] \in \hat{S}$

$$f_{[s]}^{[s_1]\ldots[s_n]} = \{f : s'_1 \ldots s'_n \to s' \in \Sigma \mid s'_i \in [s_i] \;\; 1 \leqslant i \leqslant n, \; s' \in [s]\} \tag{2.1}$$

*is the family of "subsort polymorphic" operators* $f$ *for those components. For* $f : s_1 \cdots s_n \to s$*, let args*$(f) = \{s_1, \cdots, s_n\}$ *and ran*$(f) = s$*.* □

If each connected component $[s] \in \widehat{S}$ *contains a top element* $\top_{[s]} \in [s]$ such that for each $s' \in [s]$, $\top_{[s]} \geqslant s'$, we say that $\Sigma$ is *topped*. Similarly, if $\Sigma$ is topped and for each operator $f : s_1 \ldots s_n \to s \in \Sigma$ we have an operator $f : [s_1] \ldots [s_n] \to [s] \in \Sigma$, we say that $\Sigma$ is *kind-complete*. Whenever necessary, we may assume that a signature is *topped* and *kind-complete*; when we do so, we will add a remark to that effect. Note this involves no real loss of generality since a signature can always be topped/kind-completed in a way that is consistent with its original meaning.

**Definition 2.2** *For* $\Sigma = (S, \leqslant, \Sigma)$ *an OS signature,* $A$ *is an* order-sorted $\Sigma$-algebra *iff:*

- *$A$ is a many-sorted $(S, \Sigma)$-algebra $A$,*
- *whenever $s \leqslant s'$, then we have $A_s \subseteq A_{s'}$, and*
- *whenever $f : w \to s, f : w' \to s' \in f_{[s]}^{[s_1]\ldots[s_n]}$ and $\overline{a} \in A^w \cap A^{w'}$, then we have $A_{f:w\to s}(\overline{a}) = A_{f:w'\to s'}(\overline{a})$, where $A^{s_1\ldots s_n} = A_{s_1} \times \ldots \times A_{s_n}$.*

*An* order-sorted $\Sigma$-homomorphism $h : A \to B$ *is a many-sorted* $(S, \Sigma)$*-homomorphism such that whenever* $[s] = [s']$ *and* $a \in A_s \cap A_{s'}$*, then we have* $h_s(a) = h_{s'}(a)$*.* $h$ *is* injective*, resp.* surjective*, resp.* bijective*, iff for each* $s \in S$ $h_s$ *is injective, resp. surjective, resp. bijective. We call* $h$ *an* isomorphism *if there is another order-sorted* $\Sigma$*-homomorphism* $g : B \to A$ *such that for each* $s \in S$*,* $h_s \circ g_s = 1_{B_s}$*, and* $g_s \circ h_s = 1_{A_s}$*, with* $1_{A_s}, 1_{B_s}$ *the identity functions on* $A_s, B_s$*. If* $\Sigma$ *is topped, one can show* $f$ *is an isomorphism iff* $f$ *is bijective. Order-sorted* $\Sigma$*-algebras and homomorphisms define a category* $\mathbf{OSAlg}_\Sigma$*.*□

**Theorem 2.1** *[32] The category* $\mathbf{OSAlg}_\Sigma$ *has an initial algebra. Furthermore, if* $\Sigma$ *is sensible, then the term algebra* $T_\Sigma$ *with:*

- *if $a : \epsilon \to s$ then $a \in T_{\Sigma,s}$, ($\epsilon$ denotes the empty string),*
- *if $t \in T_{\Sigma,s}$ and $s \leqslant s'$ then $t \in T_{\Sigma,s'}$,*
- *if $f : s_1 \dots s_n \to s$ and $t_i \in T_{\Sigma,s_i}$ $1 \leqslant i \leqslant n$, then $f(t_1,\dots,t_n) \in T_{\Sigma,s}$,*

*is initial, i.e., there is a unique $\Sigma$-homomorphism to each $\Sigma$-algebra.* □

For $[s] \in \widehat{S}$, $T_{\Sigma,[s]}$ denotes the set $T_{\Sigma,[s]} = \bigcup_{s' \in [s]} T_{\Sigma,s'}$. $T_\Sigma$ will (ambiguously) denote: (i) the term algebra; (ii) its underlying $S$-sorted set; and (iii) the set $T_\Sigma = \bigcup_{s \in S} T_{\Sigma,s}$. An OS signature $\Sigma$ is said to *have non-empty sorts* iff for each $s \in S$, $T_{\Sigma,s} \neq \emptyset$. An important requirement on a sensible and monotonic signature is *regularity* (or just *preregularity*[1] [33, 6]) [33]. Regularity requires for each operator $f \in \Sigma$ and sort string $u \in S^*$ that, if the set $\{ws \in S^* \mid f : w \to s \in \Sigma \wedge w \geqslant u\}$ is non-empty, then it has a smallest element. This ensures that each $\Sigma$-term $t \in T_\Sigma(X)$ has a *least sort*, denoted $ls_\Sigma(t)$, with $t \in T_\Sigma(X)_{ls_\Sigma(t)}$ and makes order-sorted automated deduction tasks like term rewriting or unification much easier: the matching of a term $t$ to a variable $x\!:\!s$ will succeed iff $ls_\Sigma(t) \leqslant s$. Without regularity, or preregularity, a costly determination of all possible sorts of $t$ is needed.

An $S$-sorted set $X = \{X_s\}_{s \in S}$ of *variables*, satisfies $s \neq s' \Rightarrow X_s \cap X_{s'} = \emptyset$, and the variables in $X$ are always assumed disjoint from all constants in $\Sigma$. The $\Sigma$-*term algebra* with variables in $X$, $T_\Sigma(X)$, is the *initial algebra* for the signature $\Sigma(X)$ obtained by adding to $\Sigma$ the variables in $X$ *as extra constants.* Since a $\Sigma(X)$-algebra is just a pair $(A, \alpha)$, with $A$ a $\Sigma$-algebra, and $\alpha$ an *interpretation of the constants* in $X$, i.e., an $S$-sorted function $\alpha \in [X \to A]$, the $\Sigma(X)$-initiality of $T_\Sigma(X)$ can be expressed as the following corollary of Theorem 2.1:

**Theorem 2.2** *(Freeness Theorem). If $\Sigma$ is sensible, for each $A \in \mathbf{OSAlg}_\Sigma$ and $\alpha \in [X \to A]$, there exists a unique $\Sigma$-homomorphism, $\_\alpha : T_\Sigma(X) \longrightarrow A$ extending $\alpha$, i.e., such that for each $s \in S$ and $x \in X_s$ we have $x\alpha_s = \alpha_s(x)$.*□

In particular, when $A = T_\Sigma(Y)$, an interpretation of the constants in $X$, i.e., an $S$-sorted function $\sigma \in [X \to T_\Sigma(Y)]$ is called a *substitution*, and its unique homomorphic extension $\_\sigma : T_\Sigma(X) \to T_\Sigma(Y)$ is also called a substitution. Define $dom(\sigma) = \{x \in X \mid x \neq x\sigma\}$, and $ran(\sigma) = \bigcup_{x \in dom(\sigma)} vars(x\sigma)$. Given variables $Z$, the substitution $\sigma|_Z$ agrees with $\sigma$ on $Z$ and is the identity elsewhere. A *variable specialization* is a substitution $\rho$ that just renames a few variables and may lower their sort. More precisely, $dom(\rho)$ is a finite set of variables $\{x_1, \dots, x_n\}$, with respective sorts $s_1, \dots, s_n$, and $\rho$ injectively maps the $x_1, \dots, x_n$ to variables $x'_1, \dots, x'_n$ with respective sorts $s'_1, \dots, s'_n$ such that $s'_i \leqslant s_i$, $1 \leqslant i \leqslant n$.

### 2.1.1 First-Order Equational Formulas

The first-order language of *equational* $\Sigma$-*formulas* is defined in the usual way: its atoms are $\Sigma$-*equations* $t = t'$. The set $Form(\Sigma)$ of *equational* $\Sigma$-*formulas* is then inductively built from atoms by: conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and universal ($\forall x{:}s$) and existential ($\exists x{:}s$) quantification with sorted variables $x{:}s \in X_s$ for some $s \in S$. The literal $\neg(t = t')$ is denoted $t \neq t'$. Given a $\Sigma$-algebra $A$, a formula $\varphi \in Form(\Sigma)$, and an assignment $\alpha \in [Y \to A]$, with $Y = fvars(\varphi)$ the free variables of $\varphi$, the *satisfaction relation* $A, \alpha \models \varphi$ is defined inductively as usual.

[1] An order-sorted signature $\Sigma$ is *preregular* [33] iff for each $\Sigma$-term $t \in T_\Sigma(X)$ the set $\{s \in S \mid t \in T_\Sigma(X)_s\}$ has a least element, denoted $ls_\Sigma(t)$, in the poset order $(S, \leqslant)$.

An OS *equational theory* is a pair $T = (\Sigma, E)$, with $E$ a set of (possibly conditional) $\Sigma$-equations. $\mathbf{OSAlg}_{(\Sigma,E)}$ denotes the full subcategory of $\mathbf{OSAlg}_{\Sigma}$ with objects those $A \in \mathbf{OSAlg}_{\Sigma}$ such that $A \models E$, called the $(\Sigma, E)$-*algebras*. $\mathbf{OSAlg}_{(\Sigma,E)}$ has an *initial algebra* $T_{\Sigma/E}$ [32]. Given $T = (\Sigma, E)$ and $\varphi \in \mathit{Form}(\Sigma)$, we call $\varphi$ $T$-*valid*, written $E \models \varphi$, iff $A \models \varphi$ for each $A \in \mathbf{OSAlg}_{(\Sigma,E)}$. We call $\varphi$ $T$-*satisfiable* iff there exists $A \in \mathbf{OSAlg}_{(\Sigma,E)}$ with $\varphi$ satisfiable in A. Note that $\varphi$ is $T$-*valid* iff $\neg\varphi$ is $T$-*unsatisfiable*. The inference system in [32] is *sound and complete* for OS equational deduction, i.e., for any OS equational theory $(\Sigma, E)$, and $\Sigma$-equation $u = v$ we have an equivalence $E \vdash u = v \quad \Leftrightarrow \quad E \models u = v$. Deducibility $E \vdash u = v$ is abbreviated as $u =_E v$, called $E$-*equality*.

We recall the definitions of $E$-unifier and $E$-unification algorithm below:

**Definition 2.3** *(Unifier, Unification Algorithm). Given an OS equational theory $(\Sigma, E)$ and a system of $\Sigma$-equations, i.e., a conjunction $\phi \equiv u_1 = v_1 \wedge \ldots \wedge u_n = v_n$ of $\Sigma$-equations, an $E$-*unifier *of $\phi$ is a substitution $\sigma$ such that $u_i\sigma =_E v_i\sigma$, $1 \leqslant i \leqslant n$.*

*An $E$-*unification algorithm *for $(\Sigma, E)$ is an algorithm generating for each system of $\Sigma$-equations $\phi$ and finite set of variables $W \supseteq \mathit{vars}(\phi)$ a *complete set *of $E$-unifiers $\mathit{Unif}_E^W(\phi)$ where each $\tau \in \mathit{Unif}_E^W(\phi)$ is assumed idempotent and with $\mathit{dom}(\tau) = \mathit{vars}(\phi)$, and is "away from $W$" in the sense that $\mathit{ran}(\tau) \cap W = \varnothing$. The set $\mathit{Unif}_E^W(\phi)$ is called "complete" in the precise sense that for any $E$-unifier $\sigma$ of $\phi$ there is a $\tau \in \mathit{Unif}_E(\phi)$ and a substitution $\rho$ such that $\sigma|_W =_E (\tau\rho)|_W$, where, by definition, $\alpha =_E \beta$ for substitutions $\alpha, \beta$ means $(\forall x \in X)\ \alpha(x) =_E \beta(x)$. Such an algorithm is called *finitary *if it always terminates with a *finite set *$\mathit{Unif}_E^W(\phi)$ for any $\phi$. We will use the notation $\mathit{Unif}_E^{\mathit{exp}_1,\ldots,\mathit{exp}_n}(\phi)$ as an abbreviation for $\mathit{Unif}_E^W(\phi)$, where $W = \mathit{vars}(\phi) \cup \bigcup_{1 \leqslant i \leqslant n} \mathit{vars}(\mathit{exp}_i)$.*

Note the lack of predicate symbols above is only *apparent*: full order-sorted first-order logic can be *reduced* to order-sorted algebra and equational formulas (see [35, 36, 25]).

### 2.1.2 Signature Morphisms

**Definition 2.4** *A *signature morphism *$H : \Sigma \to \Sigma'$ (called a *view *in Maude [6]) is a monotonic function $H : (S, \leqslant) \to (S', \leqslant')$ of the underlying posets of sorts, together with a mapping $H$ sending each $f : s_1 \ldots s_n \to s$ in $\Sigma$ to a term $H(f) \in T_{\Sigma'}(\{x_1{:}H(s_1), \ldots, x_n{:}H(s_n)\})_{H(s)}$. $H$ defines a well-typed translation of the syntax of $\Sigma$ into that of $\Sigma'$. It inductively maps each $\Sigma$-term $t$ to a $\Sigma'$-term $H(t)$ by mapping $x{:}s$ to $x{:}H(s)$, and $H(f(t_1, \ldots, t_n))$ to $H(f)\{x_1{:}H(s_1) \mapsto H(t_1), \ldots, x_n{:}H(s_n) \mapsto H(t_n)\}$, where $\{x_1{:}H(s_1) \mapsto H(t_1), \ldots, x_n{:} H(s_n) \mapsto H(t_n)\}$ denotes the obvious substitution. $H$ extends naturally to a translation of equational formulas $H : \mathit{Form}(\Sigma) \to \mathit{Form}(\Sigma')$ by mapping atoms according to $H$, respecting Boolean connectives, and mapping each quantifier $\forall x{:}s$ (resp. $\exists x{:}s$) to $\forall x{:}H(s)$ (resp. $\exists x{:}H(s)$).*

*A *signature inclusion*, denoted $\Sigma \hookrightarrow \Sigma'$, is a signature morphism that is a poset inclusion $(S, \leqslant) \hookrightarrow (S', \leqslant')$ on sorts and maps each $f : s_1 \ldots s_n \to s$ to itself: more precisely, to the term $f(x_1{:}s_1, \ldots, x_n{:}s_n)$. The most important subsignature inclusion we consider is the *constructor subsignature *$\Omega \subseteq \Sigma$.* ▫

A signature morphism $H : \Sigma \to \Sigma'$ induces a functor in the *opposite* direction $\_\,|_H\colon \mathbf{OSAlg}_{\Sigma'} \to \mathbf{OSAlg}_{\Sigma}$, where for each $B \in \mathbf{OSAlg}_{\Sigma'}$, the algebra $B\,|_H \in \mathbf{OSAlg}_{\Sigma}$, called its $H$-*reduct*, is defined using $H$ as follows: (i) for each $s \in S$, $(B\,|_H)_s = B_{H(s)}$; and (ii) for each $f : s_1 \ldots s_n \to s$ in $\Sigma$, $(B\,|_H)_f$ is the function $\lambda(x_1 \in B_{H(s_1)}, \ldots, x_n \in B_{H(s_n)}).\ H(f) : B_{H(s_1)} \times \ldots \times B_{H(s_n)} \to B_{H(s)}$ defined by the term $H(f)$ in the $\Sigma'$-algebra $B$.

In Goguen and Burstall's sense, the key point about order-sorted signature morphisms is that they make order-sorted logic an *institution* [37], so that *truth is preserved along translations*, i.e. for any $B \in \mathbf{OSAlg}_{\Sigma'}$ and any $\Sigma$-sentence $\varphi$ we have the equivalence:

$$B \models H(\varphi) \;\Leftrightarrow\; B\,|_H \models \varphi \tag{2.2}$$

This equivalence can be checked in several ways. For example, one can use the embedding of order-sorted logic in membership equational logic, itself embedded in many-sorted first-order logic, as detailed in [32]. This reduces the issue to the same well-known equivalence for many-sorted first-order logic.

## 2.2 REWRITING LOGIC

We now recall some basic concepts about *rewriting logic*. Note that in this thesis, we use rewrite theories at two levels: (a) rewrite theories provide a means to mechanize equational deduction for an OS equational theory $(\Sigma, E)$ by orienting the equations $E$ as rewrite rules—we focus on theories at this level in chapters 3 and 4; (b) rewrite theories provide a means to axiomatize a *distributed system* as a rewrite theory $\mathcal{R}$, so that concurrent computation is modeled as concurrent rewriting with the rules of $\mathcal{R}$ *modulo* the equations of $\mathcal{R}$—we focus on theories at this level in chapters 5 and 6. The survey in [28] gives a fuller account of rewriting logic. In this thesis we use the Maude rewriting engine extensively to take our rewrite theories and obtain concrete, executable algorithms—not mere theoretical results. We will see how Maude represents rewrite theories in Section 2.3 of this chapter.

We use the notation for term positions, subterms, and term replacement from [38]: (i) positions in a term viewed as a tree are marked by strings $p \in \mathbb{N}^*$ specifying a path from the root, (ii) $t|_p$ denotes the subterm of term $t$ at position $p$, and (iii) $t[u]_p$ denotes the result of *replacing* subterm $t|_p$ at position $p$ by $u$.

### 2.2.1 Rewriting as Equational Deduction

**Definition 2.5** *(Unconditional Rewrite Theory, Rewrite Relation). An unconditional* rewrite theory *is a triple $\mathcal{R} = (\Sigma, B, R)$ with $(\Sigma, B)$ an order-sorted equational theory and $R$ a set of unconditional* $\Sigma$-rewrite rules*, i.e., sequents $l \to r$, with $l, r \in T_\Sigma(X)_{[s]}$ for some $[s] \in \widehat{S}$ such that each equation $u = v \in B$ is* regular*, i.e., $vars(u) = vars(v)$, and* linear*, i.e., there are no repeated variables in $u$, and no repeated variables in $v$. The* one-step *$R, B$-rewrite relation $t \to_{R,B} t'$, holds between $t, t' \in T_{\widehat{\Sigma}}(X)_{[s]}$, $[s] \in \widehat{S}$, iff there is a rewrite rule $l \to r \in R$, a substitution $\sigma \in [X \to T_\Sigma(X)]$, and a term position $p$ in $t$ such that $t|_p =_B l\sigma$, $t' = t[r\sigma]_p$. Note that $t \in T_\Sigma(X)_{[s]}$ and $t \to_{R,B} t'$ does not necessarily imply $t' \in T_\Sigma(X)_{[s]}$ but only $t' \in T_{\widehat{\Sigma}}(X)_{[s]}$, where $\widehat{\Sigma}$ is the kind-completion of $\Sigma$. This is because, unless further conditions are imposed on $B$ and $R$, in general we do not have $ls(l\sigma) \geqslant ls(r\sigma)$, so that $t' = t[r\sigma]_p$ need not be a $\Sigma$-term, but in general will only be a $\widehat{\Sigma}$-term.*

*$\mathcal{R}$ is called: (i)* terminating *iff the relation $\to_{R,B}$ is well-founded; (ii)* strictly $B$-coherent *[39] iff whenever $u \to_{R,B} v$ and $u =_B u'$ there is a $v'$ such that $u' \to_{R,B} v'$ and $v =_B v'$; (iii)* confluent *iff $u \to^*_{R,B} v_1$ and $u \to^*_{R,B} v_2$ imply that there are $w_1, w_2$ such that $v_1 \to^*_{R,B} w_1$, $v_2 \to^*_{R,B} w_2$, and $w_1 =_B w_2$ (where $\to^*_{R,B}$ denotes the reflexive-transitive closure of $\to_{R,B}$). Note that if (i)–(iii) hold, then for each $\widehat{\Sigma}$-term $t$ there is a $\widehat{\Sigma}$-term $u$ such that $t \to^*_{R,B} u$ and $(\nexists v)\ u \to_{R,B} v$. We then write $u = t!_{R,B}$ and $t \to!_{R,B} t!_{R,B}$, and call $t!_{R,B}$ the $R, B$-*normal form *of $t$, which, by confluence, is unique up to $B$-equality.*

Below we gather some useful notions regarding equational axioms.

**Definition 2.6** *(A/C/U/ACCU Axioms). Let $f$ be some binary operator and $e$ some constant. $U$ refers to right- and/or left-identity axioms of the form $f(x,e) = x$, or $f(e,x) = x$ for given choices of $f$ and $e$. $A$ refers to an associativity axiom of the form $f(f(x,y),z) = f(x,f(y,z))$ for given choices of $f$. $C$ refers to a commutativity axiom of the form $f(x,y) = f(y,x)$ for given choices of $f$. Also, $AC = A \cup C$, $ACU = AC \cup U$, and ACCU refers to any subset of ACU where $f$ is associative only if $f$ is commutative, i.e. $f(f(x,y),z) = f(x,f(y,z))$ implies $f(x,y) = f(y,x)$.*

Recall that the notion of preregular signature ensures a least sort $ls(t)$ for each term $t$. How should this notion be generalized when reasoning modulo axioms $B$? Intuitively we wish to have a least sort not only for a term $t$, but for a $B$-equivalence class $[t]_B$. $B$-preregularity provides such a generalization. For $B$ any combination of $A$ and/or $C$ and/or $U$ axioms, Maude automatically checks $B$-preregularity in the sense defined below.

**Definition 2.7** *(B-preregular Signature). Given regular and linear $\Sigma$-equational axioms $B$, a preregular OS signature $\Sigma$ is called $B$-*preregular *iff $B$ can be decomposed as a disjoint union $B = B_0 \uplus B_1$ such that, orienting equations $(u = v) \in B_1$ as rewrite rules $R(B_1)$ of the form $u \to v$, the following properties are satisfied:*

1. *($B_0$ sort-preserving). For each $u = v \in B_0$ and variable specialization $\rho$, $ls(u\rho) = ls(v\rho)$.*
2. *($R(B_1)$ is sort-decreasing). For each $u \to v$ in $R(B_1)$ and variable specialization $\rho$, $ls(u\rho) \geqslant ls(v\rho)$.*
3. *The restricted rewrite theory $(\Sigma, B_0, R(B_1))$ is such that $\to_{R(B_1),B_0}$ is terminating, strictly $B_0$-coherent, and confluent.*
4. *(Subsort polymorphism). If any typing of an operator $f$ in a subsort-polymorphic family $f^{[s_1]\ldots[s_n]}_{[s]}$ satisfies any axioms in $B$, then any other typing in $f^{[s_1]\ldots[s_n]}_{[s]}$ satisfies the* exact same *axioms.*[2]

We can relate $A$, $C$ and $U$ axioms and $B$-preregularity as follows. To be well-behaved, $A$ and $C$ axioms should be sort-preserving. Instead, $U$ axioms may not be so, since in an identity rule, say, $f(x,e) \to x$, variable $x$ may have a different sort than $f(x,e)$; such rules should be sort-decreasing. Maude automatically checks $B$-preregularity in this precise sense.

We remark briefly that the notion of $B$-preregular signature does indeed provide the desired notion of least sort $ls([t]_B)$ for $B$-equivalence classes. This is so because, by properties (1)–(4) and the Church-Rosser Theorem (see below), we have $t =_B t'$ iff $t!_{R(B_1),B_0} =_{B_0} t'!_{R(B_1),B_0}$, so that, by (1), we can unambiguously define $ls([t]_B) = ls(t!_{R(B_1),B_0})$.

We are now ready to define the notion of *convergent unconditional rewrite theory* and of *decomposition* of an equational theory into a convergent rewrite theory.

**Definition 2.8** *(Convergent Unconditional Rewrite Theory, Decomposition). An unconditional rewrite theory $\mathcal{R} = (\Sigma, B, R)$ is called* convergent *iff it satisfies the following properties:*

1. *For each $l \to r \in R$, $l \notin X$ and $vars(r) \subseteq vars(l)$.*
2. *$\Sigma$ is $B$-*preregular*, with $B = B_0 \uplus B_1$.*

[2] For a kind-complete signature, this just means that the axioms $B$ can always be defined with all variables in such axioms ranging over top sorts. For simplicity in what follows we will assume that (as done automatically in Maude) the axioms $B$ have been extended to top sorts in the kind-completion $\widehat{\Sigma}$.

3. *($R$ is $B$-sort-decreasing). By definition this means that for each $(l \to r) \in R$ and substitution $\alpha$ we have, $ls((l\alpha)!_{R(B_1),B_0}) \geqslant ls((r\alpha)!_{R(B_1),B_0})$.*

4. *$\to_{R,B}$ is terminating, $B$-coherent, and confluent.*

*For convergent rewrite theories we slightly modify the definition of the $R,B$-rewrite relation $t \to_{R,B} t'$ (but keep the same notation) to ensure it holds between $\Sigma$-terms, as opposed to the more general rewrite relation between $\widehat{\Sigma}$-terms in Def. 2.5, as follows. We say that $t \to_{R,B} t'$ holds between $t, t' \in T_{\Sigma}(X)_{[s]}$, $[s] \in \widehat{S}$, iff there is a rewrite rule $l \to r \in R$, a substitution $\sigma \in [X{\to}T_{\Sigma}(X)]$, and a term position $p$ in $t$ such that $t|_p =_B l\sigma$, and $t' = t[(r\sigma)!_{R(B_1),B_0}]_p$. Note that, by assumptions (2)–(3) above, $t[(r\sigma)!_{R(B_1),B_0}]_p$ is always a well-formed $\Sigma$-term.*

*Orienting equations as rewrite rules provides a way to mechanize equational deduction. A key notion is that of a* decomposition*. Given $E$, a set of $\Sigma$-equations, we let $R(E) = \{u \to v \text{ if } \phi \mid u = v \text{ if } \phi \in E\}$, that is, the set of rewrite rules produced by orienting equations from left to right. A* decomposition *of an OS equational theory $(\Sigma, E)$ is a convergent rewrite theory $\mathcal{R} = (\Sigma, B, R)$ where $E = E_0 \uplus B$ and $R = R(E_0)$.*

Note that if $\Sigma$ is $B$-preregular with $B = B_0 \uplus B_1$, then $(\Sigma, B_0, R(B_1))$ is a decomposition in the above sense, since in that case one can show that $R(B_1)$ is $B_0$-sort decreasing iff it is sort-decreasing in the sense of Definition 2.7. More generally, whenever $\mathcal{R} = (\Sigma, B, R)$ with $\Sigma$ $B$-preregular, $B = B_0 \uplus B_1$, and $B_1 = \varnothing$, $R$ is $B$-sort decreasing iff it is sort decreasing in the standard sense. A practical question is how to check in general the $B$-sort decreasingness condition (3) above.

The key property of a decomposition is:

**Theorem 2.3** *(Church-Rosser Theorem) [40, 39] Let $\mathcal{R} = (\Sigma, B, R)$ be a decomposition of $(\Sigma, E)$. Then we have an equivalence: $E \vdash u = v \;\;\Leftrightarrow\;\; u!_{R,B} =_B v!_{R,B}$.*□

If $\mathcal{R} = (\Sigma, B, R)$ is a decomposition of $(\Sigma, E)$, and $X$ an $S$-sorted set of variables, the *canonical term algebra* $C_{\mathcal{R}}(X)$ has $C_{\mathcal{R}}(X)_s = \{[t!_{R,B}]_B \mid t \in T_{\Sigma}(X)_s\}$, and interprets each $f : s_1 \ldots s_n \to s$ as the function $C_{\mathcal{R}}(X)_f : ([u_1]_B, \ldots, [u_n]_B) \mapsto [f(u_1, \ldots, u_n)!_{R,B}]_B$. By the Church-Rosser Theorem we then have an isomorphism $h : T_{\Sigma/E}(X) \cong C_{\mathcal{R}}(X)$, where $h : [t]_E \mapsto [t!_{R,B}]_B$. In particular, when $X$ is the empty family of variables, the canonical term algebra $C_{\mathcal{R}}$ is an initial algebra, and is the most intuitive possible model for $T_{\Sigma/E}$ as an algebra of *values* computed by $R,B$-simplification.

Quite often, the signature $\Sigma$ on which $T_{\Sigma/E}$ is defined has a natural decomposition as a disjoint union $\Sigma = \Omega \uplus \Delta$, where the elements of $C_{\mathcal{R}}$, that is, the *values* computed by $R,B$-simplification, are $\Omega$-terms, whereas the function symbols $f \in \Delta$ are viewed as *defined functions* which are *evaluated away* by $R,B$-simplification. $\Omega$ (with same poset of sorts as $\Sigma$) is then called a *constructor subsignature* of $\Sigma$. Call a decomposition $\mathcal{R} = (\Sigma, B, R)$ of $(\Sigma, E)$ *sufficiently complete* with respect to the *constructor subsignature* $\Omega$ iff for each $t \in T_{\Sigma}$ we have: (i) $t!_{R,B} \in T_{\Omega}$, and (ii) if $u \in T_{\Omega}$ and $u =_B v$, then $v \in T_{\Omega}$. This ensures that for each $[u]_B \in C_{\mathcal{R}}$ we have $[u]_B \subseteq T_{\Omega}$. Of course, we want $\Omega$ *as small as possible* with these properties. Sufficient completeness is closely related to the notion of a *protecting* theory inclusion.

**Definition 2.9** *(Protecting, Constructor Decomposition). An equational theory $(\Sigma, E)$* protects *another theory $(\Omega, E_{\Omega})$ iff $(\Omega, E_{\Omega}) \subseteq (\Sigma, E)$ and the unique $\Omega$-homomorphism $h : T_{\Omega/E_{\Omega}} \to T_{\Sigma/E}|_{\Omega}$ is an isomorphism $h : T_{\Omega/E_{\Omega}} \cong T_{\Sigma/E}|_{\Omega}$.*

*A decomposition* $\mathcal{R} = (\Sigma, B, R)$ protects *decomposition* $\mathcal{R}_0 = (\Sigma_0, B_0, R_0)$ *iff* $\mathcal{R}_0 \subseteq \mathcal{R}$*, i.e.,* $\Sigma_0 \subseteq \Sigma$*,* $B_0 \subseteq B$*, and* $R_0 \subseteq R$*, and for all* $t, t' \in T_{\Sigma_0}(X)$ *we have: (i)* $t =_{B_0} t' \Leftrightarrow t =_B t'$*, (ii)* $t = t!_{R_0,B_0} \Leftrightarrow t = t!_{R,B}$*, and (iii)* $C_{\mathcal{R}_0} = C_{\mathcal{R}}|_{\Sigma_0}$*.*

$\mathcal{R}_\Omega = (\Omega, B_\Omega, R_\Omega)$ *is a* constructor decomposition *of* $\mathcal{R} = (\Sigma, B, R)$ *iff* $\mathcal{R}$ *protects* $\mathcal{R}_\Omega$ *and* $\Sigma$ *and* $\Omega$ *have the same poset of sorts, so that by definition of decomposition,* $\mathcal{R}$ *is sufficiently complete with respect to* $\Omega$*. Finally,* $\Omega$ *is called a subsignature of* free constructors modulo $B_\Omega$ *iff* $R_\Omega = \varnothing$*, so that* $C_{\mathcal{R}_0} = T_{\Omega/B_\Omega}$*.*

The case where all constructor terms are in $R, B$-normal form is captured by $\Omega$ being a subsignature of free constructors modulo $B_\Omega$. Note also that conditions (i) and (ii) are, so called, "no confusion" conditions, and for protecting extensions (iii) is a "no junk" condition, that is, $\mathcal{R}$ does not add new data to $C_{\mathcal{R}_0}$, whereas for conservative extensions (iii) is relaxed to the "no confusion" condition $C_{\mathcal{R}_0} \subseteq C_{\mathcal{R}}|_{\Sigma_0}$, which is already implicit in (i) and (ii). Therefore, protecting extensions are a stronger kind of conservative extensions.

**Example 2.1** *(Integers with Addition). Consider the theory of the integers with addition* $\mathcal{Z}_+$ *with signature* $\Sigma$ *and constructor subsignature* $\Omega$*. Both signatures have sorts Nat, NzNat, NzNeg, and Int, and subsorts NzNat < Nat and Nat NzNeg < Int, where NzNat (resp. NzNeg) denotes the non-zero naturals (resp. negatives). The constructor subsignature* $\Omega$ *has constants 0 of sort Nat and* 1 *of sort NzNat, and operators* $\_+\_ : Nat\ Nat \to Nat$*,* $\_+\_ : NzNat\ NzNat \to NzNat$*, and* $- : NzNat \to NzNeg$ *shown in blue. The signature* $\Sigma$ *contains all the operators defined in* $\Omega$ *and adds one defined function symbol:* $\_+\_ : Int\ Int \to Int$ *shown in red. Let* $B$ *be the set of ACU axioms for* $(+)$ *with identity* 0 *and the equations* $E_0$ *defining* $(+)$ *be the following (with variables* $i{:}Int$*,* $n{:}NzNat$*, and* $m{:}NzNat$*)*

$$i + n + -(n) = i \tag{2.3}$$

$$i + -(n) + -(m) = i + -(n + m) \tag{2.4}$$

$$i + n + -(n + m) = i + -(m) \tag{2.5}$$

*Then* $(\Sigma, B, R)$ *is a decomposition of the theory* $(\Sigma, B \cup E_0)$ *with* $R = \overrightarrow{E_0}$*. Furthermore* $(\Sigma, B, R)$ *protects the constructor decomposition* $(\Omega, B, \varnothing)$*, i.e.* $\Sigma$ *is sufficiently complete with respect to* $\Omega$ *modulo* $B$*.*

Variants

We now recall the key notion of a *variant* which answers, in a sense, two questions: (i) how can we best describe symbolically the elements of $C_{\mathcal{R}}(X)$ that are *reduced substitution instances* of a *pattern term* $t$? and (ii) given an original pattern $t$, how many other patterns do we need to describe the reduced instances of $t$ in $C_{\mathcal{R}}(X)$?

**Definition 2.10** *(Variants). Given a decomposition* $\mathcal{R} = (\Sigma, B, R)$ *of an OS equational theory* $(\Sigma, E)$ *and a* $\Sigma$*-term* $t$*, a* variant[3] *[23, 24] of* $t$ *is a pair* $(u, \theta)$ *such that: (i)* $u =_B (t\theta)!_{R,B}$*, (ii)* $dom(\theta) \subseteq vars(t)$*, and (iii)* $\theta = \theta!_{R,B}$*, that is,* $\theta(x) = \theta(x)!_{R,B}$ *for all variables* $x$*.* $(u, \theta)$ *is called a* ground variant *iff* $u \in T_\Sigma$*.*

**Definition 2.11** *(Variant Preorder). Given variants* $(u, \theta)$ *and* $(v, \gamma)$ *of* $t$*,* $(u, \theta)$ *is called* more general than $(v, \gamma)$*, denoted* $(u, \theta) \sqsupseteq_B (v, \gamma)$*, iff there exists a substitution* $\rho$ *such that: (i)* $(\theta\rho)|_{vars(t)} =_B \gamma$*, and (ii)* $u\rho =_B v$*.*

[3] For a discussion of similar but not exactly equivalent versions of the variant notion see [41]. Here we follow the simpler formulation in [24], rather than the one in [23], because it is technically essential for some results to hold [41].

**Definition 2.12** *(All Variants, Complete Set of Variants). Let $[\![t]\!]^*_{R,B}$ denote the set of all variants of $t$ and $[\![t]\!]_{R,B} = \{(u_i,\theta_i) \mid i \in I\}$ denote a* complete set of variants *of $t$, that is, a set of variants such that for any variant $(v,\gamma)$ of $t$ there is an $i \in I$, such that $(u_i,\theta_i) \sqsupseteq_B (v,\gamma)$.*

**Definition 2.13** *(Finite Variant Property). A decomposition $\mathcal{R} = (\Sigma, B, R)$ of $(\Sigma, E)$ has the* finite variant property *[23] (FVP) iff for each $\Sigma$-term $t$ there is a* finite *complete set of variants $[\![t]\!]_{R,B} = \{(u_1,\theta_1),\ldots,(u_n,\theta_n)\}$. Since if $B$ has a finitary $B$-unification algorithm the relation $(u,\alpha) \sqsupseteq_B (v,\beta)$ is decidable by $B$-matching, in such case we can always assume that if $\mathcal{R} = (\Sigma, B, R)$ is FVP, $[\![t]\!]_{R,B}$ can be chosen to be not only complete, but also a* minimal set of most general *variants, in the sense that for $1 \leqslant i < j \leqslant n$, $(u_i,\theta_i) \not\sqsupseteq_B (u_j,\theta_j) \ \wedge\ (u_j,\theta_j) \not\sqsupseteq_B (u_i,\theta_i)$.*

*Also, given any finite set of variables $W \supseteq vars(t)$ we can always choose $[\![t]\!]_{R,B}$ to be of the form $[\![t]\!]^W_{R,B}$, where each $(u_i,\theta_i) \in [\![t]\!]^W_{R,B}$ has $\theta_i$ idempotent with $dom(\theta_i) = vars(t)$, and "away from $W$," in the sense that $ran(\theta_i) \cap W = \varnothing$. As for unifiers, $[\![t]\!]^{exp_1,\ldots,exp_n}_{R,B}$ abbreviates $[\![t]\!]^W_{R,B}$, where $W = vars(\phi) \cup \bigcup_{1 \leqslant i \leqslant n} vars(exp_i)$.*

*Narrowing* is a form of *symbolic reduction* where the term to be reduced $B$-*unifies* at some position $p$ with the lefthand side of a rewrite rule instead of just $B$-matching such lefthand side. *Folding variant narrowing* is a narrowing strategy that provides an effective method to generate $[\![t]\!]_{R,B}$ whenever $B$ has a finitary unification algorithm [24]. Furthermore, since $[\![t]\!]_{R,B}$ is finite for each $t$ whenever $\mathcal{R}$ is FVP, the strategy *terminates* iff $\mathcal{R}$ is FVP.

FVP is a *semi-decidable* property [41], which can be easily verified (when it holds) by checking, using folding variant narrowing, that for each function symbol $f$ the term $f(x_1,\ldots,x_n)$, with the sorts of the $x_1,\ldots,x_n$ those of $f$, has a finite complete set of variants.

**Example 2.2** *(Integers with Addition, Variants). Recall the theory $\mathcal{Z}_+$ specified in Example 2.1 and its decomposition $(\Sigma, B, R)$. By using folding variant narrowing one can automatically check that $(\Sigma, B, R)$ is FVP. Thus, for any term $t$, we can always compute a finite and complete set of variants $[\![t]\!]^W_{R,B}$ with $W \supseteq vars(t)$. Let $i$ and $i'$ be variables of sort Int and $n$, $n'$, $m$, and $p$ be variables of sort NzNat. Then for the term $i + n$ where $W \supseteq \{i, n\}$, the Maude folding variant narrowing algorithm generates the following complete set of variants $[\![i+n]\!]^W_{R,B}$:*

$$\begin{aligned}
&(i' + n', && \{i \mapsto i', && n \mapsto n'\}) && (2.6)\\
&(i', && \{i \mapsto i' + -(n'), && n \mapsto n'\}) && (2.7)\\
&(i' + n', && \{i \mapsto i' + -(m), && n \mapsto n' + m\}) && (2.8)\\
&(i' + -(m), && \{i \mapsto i' + -(m + n'), && n \mapsto n'\}) && (2.9)\\
&(i' + n' + -(m), && \{i \mapsto i' + -(m + p), && n \mapsto n' + p\}). && (2.10)
\end{aligned}$$

*Note that, according to Definition 2.10, given any variant $(t,\theta)$ in the list above, we have $t =_B (i+n)\theta!_{R,B}$. As an example, we consider the first and then the second variant. Let $\theta_1 = \{i \mapsto i', n \mapsto n'\}$ and $\theta_2 = \{i \mapsto i' + -(n'), n \mapsto n'\}$. Then we have $(i+n)\theta_1 = i' + n'$ and $(i+n)\theta_2 = i' + -(n') + n'$. But clearly, $i' + n'$ is irreducible by rules $R$, while $i' + -(n') + n'$ reduces to $i'$ by rule $i + n + -(n) \to i$.*

### 2.2.2 Rewriting as Concurrent Computation

When mechanizing equational deduction, we are interested in an canonical term algebra $C_{\mathcal{R}}$ induced by a rewrite theory $(\Sigma, B, R)$ which is the decomposition of some equational theory $(\Sigma, E)$ where the states are $B$-equivalence classes of terms (where $B$ is restricted to ensure decidability). Instead, when using rewriting to model general concurrent computation, we typically desire an *canoncial reachability model* $\mathcal{C}_{\mathcal{R}}$, i.e., a kind of transition system whose states are $E$-equivalence classes of terms defined by the decomposition of a theory $(\Sigma, E)$, and whose transitions are defined by rewrite rules $R$ over such equivalence classes. Since equations $E$ are oriented as rules operationally, we require a second kind of *coherence* property between rules and equations that we briefly touch on below.

**Definition 2.14** *(Rewrite Theory). A* rewrite theory *is a 3-tuple* $\mathcal{R} = (\Sigma, E \cup B, R)$ *with* $(\Sigma, E \cup B)$ *an OS equational theory with $E$ possibly conditional and $R$ a set of possibly conditional* $\Sigma$-rewrite rules*, i.e., sequents $l \to r$ if $\phi$, with $l, r \in T_{\Sigma}(X)_{[s]}$ for some $[s] \in \widehat{S}$, and $\phi$ a quantifier-free $\Sigma$-formula.*[4] *We further assume that:*

1. *Each equation $u = v \in B$ is* regular*, i.e., $vars(u) = vars(v)$, and* linear*, i.e., there are no repeated variables in $u$, and no repeated variables in $v$. Furthermore, $\Sigma$ is $B$-*preregular*.*

2. *The equations $E$, when oriented as rewrite rules $\vec{E} = \{u \to v \text{ if } \psi \mid u = v \text{ if } \psi \in E\}$, are* convergent *modulo $B$*[5]*, that is, sort-decreasing, strictly coherent, confluent, and operationally terminating as rewrite rules modulo $B$ [44].*

3. *The rules $R$ are* ground coherent *with the equations $E$ modulo $B$ [45].*

We refer to [28, 44, 45] for more details, but give here an intuitive high-level explanation of what the above conditions mean in practice. Conditions (1)–(2) ensures that the initial algebra $T_{\Sigma/E \cup B}$ is isomorphic to the *canonical term algebra* $C_{\Sigma/E,B}$, whose elements are $B$-equivalence classes of $\vec{E}, B$-irreducible ground $\Sigma$-terms. In the context of (1)–(2), condition (3) ensures that "computing $\vec{E}, B$-canonical forms before performing $R, B$-rewriting" is a *complete* strategy. That is, if $t \to_{R,B} t'$ and $u = t!_{E,B}$, i.e., $t \to^{*}_{\vec{E},B} u$ with $u$ in $\vec{E}, B$-canonical form (abbreviated in what follows to $u = t!$), then there exists a $u'$ such that $u \to_{R,B} u'$ and $t'! =_B u'!$.

Conditions (1)–(3) allow a simple and intuitive description of the *initial reachability model* $\mathcal{T}_{\mathcal{R}}$ [46] of $\mathcal{R}$ as the *canonical reachability model* $\mathcal{C}_{\mathcal{R}}$ whose states are the elements of the *canonical term algebra* $C_{\Sigma/E,B}$, and where the one-step transition relation $[u] \to_{\mathcal{R}} [v]$ holds iff $u \to_{R,B} u'$ and $[u'!] = [v]$. Furthermore, if $u \to_{R,B} u'$ has been performed with a rewrite rule $l \to r$ *if* $\phi \in R$ and a ground substitution $\sigma \in [Y \to T_{\Sigma}]$, then, assuming $B$-equality is decidable, checking whether condition $E \cup B \models \phi\sigma$ holds is *decidable* by reducing the terms in $\phi\sigma$ to $\vec{E}, B$-canonical form.

[4] Usually, $\phi$ is assumed to be a *conjunction* of $\Sigma$-equations. We give here this more general definition for three reasons: (i) often, using *equationally-defined equality predicates* [42], a quantifier-free formula can be transformed into a conjunction of equalities; (ii) the more general notion is particularly useful for symbolic methods; and (iii) the semantics for this more general notion has been studied in detail in [43].

[5] This notion of convergence is slightly different than the previous notion in Definition 2.8; this is because of the extra complexity of *conditional* equations that must be accounted for.

```
fmod PRES-NAT is
  sort Bool .
  op true : -> [ctor] .
  op false : -> [ctor] .

  sort NzNat Nat .
  subsort NzNat < Nat .
  op 0 : -> Nat [ctor] .
  op 1 : -> NzNat [ctor] .
  op _+_ : Nat Nat -> Nat [ctor assoc comm id: 0] .
  op _+_ : NzNat Nat -> NzNat [ctor assoc comm id: 0] .

  var J K : Nat . var P : NzNat .

  op _<=_ : Nat Nat -> Bool .
  op _<_  : Nat Nat -> Bool .

  eq J     <= J + K = true  [variant] .
  eq J + P <= J     = false [variant] .
  eq J     <  J + P = true  [variant] .
  eq J + K <  J     = false [variant] .
endfm

mod COUNTER-EXAMPLE is
  protecting FVP-NAT .

  var N : Nat .

  sort Counter .
  op {_} : Nat -> Counter [ctor] .

  rl {N + 1} => {N + 1 + 1} .
  rl {N + 1} => {N} .
endm
```

Figure 2.1: Theory specification for a Counter.

## 2.3 MAUDE SYNTAX

Since this thesis is concerned with finding concrete rewriting-based logics and algorithms for doing distributed system analysis, we require a means to mechanize rewrite theories. Thankfully, the Maude rewriting engine [6] provides such a means. Thus, before continuing, we gloss a few keywords in the Maude syntax and describe their associated concepts. We use the module in Figure 2.1 as an example.

In Maude, the primary definitional unit is a *module*, which is surrounded by the keyword pair `fmod`/`endfm` or `mod`/`endm`. The former specifies a *functional* module which is defined only by (ground) confluent and (ground) operationally-terminating equations, while the latter specifies a *system* module, which may contain both equations of the above form as well as potentially non-deterministic, non-terminating rewrite rules. Generally, functional modules specify one or more data structures which are *imported* by potentially several different system modules, each specifying a different system of interest.

As an example, consider a module specifying a counter (`COUNTER-EXAMPLE`) that may nondeterministically increment or decrement. The underlying functional module `PRES-NAT` specifies the theory of Presburger natural numbers. In the above example, the keyword `protecting` specifies that the system module `COUNTER-EXAMPLE` imports the functional module `PRES-NAT` in a semantics-preserving way that is precisely described in Definition 2.9.

In Maude, a module's syntax is specified by sort, subsort, and operator declarations using the keywords `sort`, `subsort`, and `op` respectively while its semantics is specified by equation and rule declarations using keywords `eq` and `rl` respectively.[6] The variable keyword `var` is used to declare variables that will be used in later rules and equations. Constructor operators, the particular symbols which form the constructor subtheory of a given theory, are specified using the `ctor` attribute, which appears in optional square brackets (`[]`) that occur after any operator declaration.

In the example above, we have four sort declarations (`Bool`, `NzNat`, `Nat`, and `Counter`) and one subsort declaration (`NzNat` < `Nat`). There are 9 operator declarations: 7 constructor symbols and 2 defined symbols. The equations in lines 18-21 ensure that the two defined symbols (`<` and `<=`), when applied to ground arguments, always evaluate to the `Bool` constructors `true` and `false`. The `variant` attribute that appears tagged on the right-hand side of these equations means that they also satisfy the *finite variant property*, so that unification problems over equalities containing these operators are *decidable*. Furthermore, validity and satisfiability of QF formulas in the initial algebra for this theory is decidable by *variant satisfiability* [36, 47].

Finally, the two rewrite rules on lines 32-33 define the state changes of the counter. Recall that this system is non-terminating, since the the increment rule on line 32 can loop. The single `Counter` operator (`{_}`) is used to mark those numbers which correspond to the current state of the counter. This is used to control the application of these two rules; even though the sorts `Nat` and `Counter` are in bijective correspondence, it is useful to separate the two, since we certainly do *not* want all numbers to non-deterministically vary. Furthermore, this extra operator ensures that our theory is topmost.

For anyone familiar with basic functional programming, the specification in Figure 2.1 hopefully is not difficult to read. There are, however, two unusual features that need to be mentioned. The first feature is Maude's support for mixfix syntax; any underbars appearing in an operator declaration (`op`) represent an argument position. For example, in the operator declaration (`op _+_`), the first underbar corresponds to the first argument while the second underbar represents the second argument, so that typed operator (`op _+_ : Nat Nat -> Nat`) applied to the arguments `0` and `1` would be written as `0 + 1` and yield a result of type `Nat`. The second feature is that binary function symbols in Maude may be declared as associative, commutative, and/or having a unit using the `assoc`, `comm`, and `id:` attributes respectively. These designations are not just descriptive; any equations in which, for example, an associative-commutative operator appears, is actually applied *modulo* associativity and commutativity. These features combined allow Maude to very naturally specify equations over lists, sets, multisets, etc... at both the syntactic and semantic level.

[6] Note that our concept of a sort/subsort is often called type/subtype in the programming language literature.

# CHAPTER 3 PATTERN OPERATIONS[1]

## 3.1 INTRODUCTION

Term patterns are used everywhere in functional and logic programming: to define predicates and functions, to perform automated deduction tasks like rewriting, matching, unification, resolution, and Knuth-Bendix completion, and also as a *symbolic notation* to describe *languages* as sets of term instances, and *language operations* by corresponding symbolic operations on the term patterns defining them. Such pattern operations, first systematically studied by Lassez and Marriott in [7] and further studied in, e.g., [8, 9, 10, 11] have many applications to, e.g., machine learning, negation in logic programming, sufficient completeness of function definitions, inductive theorem proving, and automated model building.

For greater expressiveness many declarative languages support rich type disciplines. This holds true for both higher-order functional languages and rule-based languages. For example, OBJ [12], CafeOBJ [13], and Maude [6] all support types, subtypes, subtype polymorphism, and —through their parameterized types— polymorphic and dependent types. Obviously, all the above-mentioned applications of pattern operations are also needed for these languages. What is not at all obvious —and to the best of our knowledge does not seem to have been investigated so far— is whether the algorithms defining the Boolean algebra of pattern operations for the *untyped* case in, e.g., [7, 8, 9, 10, 11] extend in a straightforward way to the more expressive patterns now available in these richer type disciplines. The example described in Figure 3.1 clearly shows that they do not.
The graph on the left describes an *order-sorted signature* [33] with two types, $A$ and $B$, and a subtype inclusion $A < B$ depicted by the vertical bar. $f$ is *subtype polymorphic*, with two typings: $f : A \to A$, and $f : B \to B$. We have constants $a, b$ of respective types $A$, $B$. A *pattern* $t$, i.e., a term possibly with variables, denotes the set (language) $\llbracket t \rrbracket = \{t\sigma \mid \sigma \;\; \textit{ground}\}$ of all its *ground instances*. The symbolic pattern difference $t - t'$ denotes the language $\llbracket t - t' \rrbracket = \llbracket t \rrbracket - \llbracket t' \rrbracket$. In the untyped case, it is well-known [7] that when $t$ and $t'$ are *linear patterns* (have no repeated variables), the symbolic difference $t - t'$ always denotes a language expressible as $\llbracket u_1 \rrbracket \cup \ldots \cup \llbracket u_k \rrbracket$, for $\{u_1, \ldots, u_k\}$ a finite set of patterns. If this were to hold in the order-sorted case, it should hold, in particular, for $\llbracket x{:}B - y{:}A \rrbracket$, with $x{:}B, y{:}A$ variables of sorts $A, B$. Adopting the convention $f^0(x) = x$, we have, $\llbracket y{:}A \rrbracket = \{f^n(a) \mid n \geqslant 0\}$, and $\llbracket x{:}B \rrbracket = \llbracket y{:}A \rrbracket \cup \{f^n(b) \mid n \geqslant 0\}$. Therefore, $\llbracket x\,{:}\,B - y\,{:}\,A \rrbracket = \{f^n(b) \mid n \geqslant 0\}$. But *there is no finite set of patterns* $\{u_1, \ldots, u_k\}$ such that $\llbracket u_1 \rrbracket \cup \ldots \cup \llbracket u_k \rrbracket = \{f^n(b) \mid n \geqslant 0\}$. Indeed, the only possible choice for a $u_i$ is $u_i = b$. All other choices: $u_i = a$, $u_i = x'{:}B$, $u_i = y'{:}A$, $u_i = f^{n+1}(x'{:}B)$, or $u_i = f^{n+1}(y'{:}A)$, $n \geqslant 0$, are impossible.

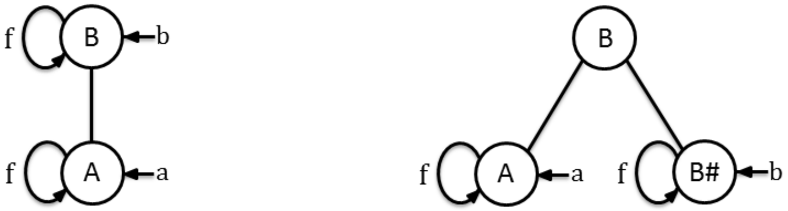


Figure 3.1: Failure of Pattern Operations in Order-Sorted Signatures

Is all lost? Not if we make our signature more expressive: the graph on the right of Figure 3.1 describes a

[1] Chapter content was originally available at https://doi.org/10.1007/s00165-017-0415-5. Reprinted with permission.

signature where we add a new subtype $B^{\#} < B$, lower the typing of $b$ to $B^{\#}$, and add the typing $f : B^{\#} \to B^{\#}$. Now $[\![z{:}B^{\#}]\!] = \{f^n(b) \mid n \geqslant 0\}$, and we can *symbolically compute* the difference $x{:}B - y{:}A = z{:}B^{\#}$. This example shows that the problem is insoluble *as formulated*, but it can be solved by a *signature transformation* extending the original signature $\Sigma$. In Section 3.2 we formally define such a transformation $\Sigma \mapsto \Sigma^{\#}$ that enriches a finite order-sorted signature $\Sigma$ with additional sorts like the sort $B^{\#}$ above. This is a key step for obtaining a Boolean algebra of *order-sorted* patterns in Section 3.4.

But the $\Sigma \mapsto \Sigma^{\#}$ transformation has other far-reaching consequences. Since it is well-known that pattern operations are intimately connected with first-order logic formulas and with negation elimination in such formulas [8, 9, 10, 11], we should first of all ask what light can the $\Sigma \mapsto \Sigma^{\#}$ transformation shed on the *validity of formulas in initial order-sorted algebras*. As we show in Section 3.3, it sheds a lot of light: it makes the validity of a first-order formula in an initial *order-sorted* algebra equivalent to the validity of an associated formula in an associated *many-sorted* initial algebra. Since the first-order theory of a many-sorted initial algebra is well-known to be decidable [14, 15, 16], this proves the decidability of the first-order theory of an initial order-sorted algebra. This result goes back to [17, 18], but the proof obtained through the $\Sigma \mapsto \Sigma^{\#}$ transformation is considerably simpler. Furthermore, it provides a new, general *transfer principle* to *reduce* certain order-sorted algebra problems to many-sorted algebra ones.

We put this transfer principle to work for order-sorted pattern operations in Section 3.4, where we show that they can be *reduced* to operations on *many-sorted* $\Sigma^{\#}$-patterns. Furthermore, we have developed and shown the correctness of an intrinsically *order-sorted* algorithm for pattern operations based on the signature $\Sigma \cup \Sigma^{\#}$ that enjoys important advantages because it allows much simpler, shorter, and "user friendlier" expression of both the pattern problems to be solved and the answers yielded by the algorithm, and because it has considerably better performance than the many-sorted one. As reported in Section 3.6, we have implemented this algebra of order-sorted pattern operations in Maude using reflection, and have performed an experimental evaluation of the, indeed substantial, advantages of the algorithm performing pattern operations at the order-sorted level of $\Sigma \cup \Sigma^{\#}$ over the one performing such operations at the many-sorted level of $\Sigma^{\#}$.

Two additional important advantages of the order-sorted algorithm for pattern operation over the signature $\Sigma \cup \Sigma^{\#}$ are studied in detail in Sections 3.4.3 and 3.5. In Section 3.4.3 we show that the subalgebra of *linear patterns* defines a very useful class of *regular tree languages*, which we call *linear pattern languages*, and provides a *computable Boolean algebra* for operations on such languages, which is isomorphic to a Boolean subalgebra of the Boolean algebra $\mathcal{P}(T_\Sigma)$ of tree languages $L$ that are subsets $L \subseteq T_\Sigma$. What this means in practice is that there is no need to rely on tree automata operations to combine linear pattern languages: much simpler operations, including not just the Boolean ones, but also the algebraic operations on languages $(L_1, \ldots, L_n) \mapsto f(L_1, \ldots, L_n)$ associated to each $f \in \Sigma$, are available in an effective way as linear pattern operations.

A second advantage of the computable algebra of linear patterns, which we discuss in Section 3.5, is that it provides a *reduction* of the problem of whether a formula $\varphi$ with *membership constraints* in the expressive language of [17, 18] is satisfied in an order-sorted initial algebra $T_\Sigma$ to that of whether a simpler purely equational formula $\pi(\varphi)$ is satisfied in the initial order-sorted algebra $T_{\Sigma\cup\Sigma^{\#}}$, eliminating again the need for performing tree automata operations involved in ascertaining the validity of the original formula $\varphi$.

As already pointed out, pattern operations have been known to have many applications at the unsorted level. The goal of this chapter is to extend such applications, and enable new ones, for more expressive programming and formal specification languages using subtypes and subtype polymorphism. Sections 3.4.3

and 3.5 already discuss, as mentioned above, some new applications. To give a flavor for the wide range of applications possible, we discuss three kinds of such applications in Section 3.7: (i) verification of sufficient completeness for order-sorted equational programs; (ii) elimination of the `otherwise` feature —used in advanced languages such as Haskell [48], ASF+SDF [49], and Maude [6] to specify the result of a function when no other pattern definition applies to the subterm being evaluated—; and (iii) verification of invariants of concurrent systems specified as rewrite theories [50].

Section 3.8 discusses related work and presents some conclusions.

## 3.2 THE $\Sigma \mapsto \Sigma^{\#}$ SIGNATURE TRANSFORMATION

We define a signature transformation $\Sigma \mapsto \Sigma^{\#}$ that will give us the key to study validity of equational formulas in initial order-sorted algebras in Section 3.3 and pattern operations in Section 3.4. $\Sigma$ is a regular order-sorted finite signature with poset of sorts $(S, \leqslant)$. As first remarked by H. Comon-Lundh in [17], an order-sorted signature $\Sigma$ is just a $\Sigma^u$-tree automaton, with $\Sigma^u$ the unsorted version of $\Sigma$, set of states $S$, and transitions rules: (i) $f(s_1, \ldots, s_n) \to s$ for each $f : s_1 \ldots s_n \to s$ in $\Sigma$, and (ii) $\epsilon$-rules $s \to s'$ for each $s < s'$ in $(S, \leqslant)$. $T_{\Sigma,s}$ is the language accepted by the accepting state $s$. This means that the problem of whether $T_{\Sigma,s} = \varnothing$, or whether any Boolean combination of sets $T_{\Sigma,s_1}, \ldots, T_{\Sigma,s_n}$ is empty, are problems decidable by an emptiness check on a regular tree language.

To construct $\Sigma^{\#}$ we must first define its set $S^{\#}$ of sorts. Call $s \in S$ *atomic* iff $s$ is a minimal element in the poset $(S, \leqslant)$. The key idea is to add to $S$ new atomic sorts $s^{\#}$ characterizing all terms whose least sort is exactly $s$, where $s$ is non-atomic. But we want $s^{\#}$ to be non-empty. Let $\downarrow s = \{s' \in S \mid s' < s\}$, and $glbs(s)$ the maximal elements of $\downarrow s$. Call $s \in S$ *redundant* iff $T_{\Sigma,s} - \bigcup_{s' \in glbs(s)} T_{\Sigma,s'} = \varnothing$. We only add $s^{\#}$ to $S^{\#}$ if $s$ is non-atomic and irredundant. Since non-emptiness is decidable, we can effectively construct $S^{\#}$ as the set containing all atomic sorts in $S$ and all new sorts $s^{\#}$ with $s \in S$ non-atomic and irredundant.

We want a *many-sorted* signature $\Sigma^{\#}$ on sorts $S^{\#}$ such that: (i) for $s$ an atomic sort in $\Sigma$, we have $T_{\Sigma^{\#},s} = T_{\Sigma,s}$, (ii) for each $s^{\#} \in S^{\#}$ we have $T_{\Sigma^{\#},s^{\#}} = T_{\Sigma,s} - \bigcup_{s' \in glbs(s)} T_{\Sigma,s'}$; and (iii) if $s, s' \in S^{\#}$ and $s \neq s'$, then $T_{\Sigma^{\#},s} \cap T_{\Sigma^{\#},s'} = \varnothing$. Thus, we will be able to represent each sort $s \in S$ as a *disjoint union* of sorts in $S^{\#}$. That is, define the function $atoms : S \to \mathcal{P}(S^{\#})$ inductively as follows: $atoms(s) =$ **if** $s$ is atomic **then** $\{s\}$ **else if** $s$ is irredundant **then** $\{s^{\#}\} \cup atoms(s_1) \cup \ldots atoms(s_n)$ **else** $atoms(s_1) \cup \ldots atoms(s_n)$ **fi fi**, where $glbs(s) = \{s_1, \ldots, s_n\}$. It then follows from (i)–(iii) above that for any $s \in S$ we will have:

$$T_{\Sigma,s} = \biguplus_{s' \in atoms(s)} T_{\Sigma^{\#},s'} \tag{3.1}$$

This is what we want. We still have to define $\Sigma^{\#}$. For this, it is useful to decompose $\Sigma$ as a "telescope" $\Sigma_0 \subset \Sigma_1 \subset \ldots \Sigma_{k-1} \subset \Sigma$. We assume that each constant $a : \epsilon \to s$ in $\Sigma$ has a single declaration of the specified sort $s$. To simplify the $\Sigma^{\#}$ construction we also assume, without real loss of generality, that $\Sigma$ can have "subsort overloading" but does not have any "ad-hoc overloading;" that is, if $(f : s_1 \ldots s_m \to s), (f : s'_1 \ldots s'_m \to s') \in \Sigma$ then $[s_i] = [s'_i]$ $1 \leqslant i \leqslant m$, and $[s] = [s']$. Recall the notation $f_{[s]}^{[s_1]\ldots[s_m]}$ for the set of all subsort-overloaded operators $f$ for these components. Given $(f : s_1 \ldots s_m \to s) \in f_{[s]}^{[s_1]\ldots[s_m]}$ define:

$$(f : s_1 \ldots s_m \to s)\!\downarrow = \{(f : s'_1 \ldots s'_m \to s') \in f_{[s]}^{[s_1]\ldots[s_m]} \mid s'_1 \ldots s'_m s' < s_1 \ldots s_m s\}. \tag{3.2}$$

as its set of *strictly smaller typings*. Define: $\Sigma_0 = \{(f : s_1 \dots s_m \to s) \in \Sigma \mid (f : s_1 \dots s_m \to s)\downarrow = \emptyset\}$, and, inductively, $\Sigma_{n+1} = \{(f : s_1 \dots s_m \to s) \in \Sigma \mid (f : s_1 \dots s_m \to s)\downarrow \subseteq \Sigma_n\}$. Because of the finiteness of $\Sigma$, we get a fixpoint $\Sigma_k = \Sigma_{k+1} = \Sigma$, giving us the above-mentioned telescope. Note that regularity of $\Sigma_n$, $n \geqslant 0$, follows easily by construction from the regularity of $\Sigma$. Furthermore, for any $t \in T_{\Sigma_n}(X)$ we have $ls_{\Sigma_n}(t) = ls_\Sigma(t)$. For example, for $\Sigma$ a signature with sorts *Nat* and *NzNat* (non-zero naturals) with 0, $s$ (successor), and with + subsort overloaded for sorts *Nat* and *NzNat*, its telescope reaches the fixpoint for $\Sigma_1 = \Sigma$, as shown in Figure 3.2.

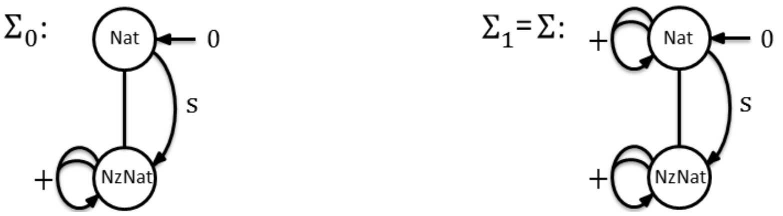


Figure 3.2: Telescope for $\Sigma$

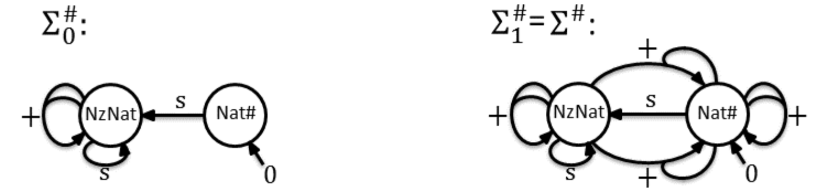


Figure 3.3: Telescope for $\Sigma^{\#}$

We will define a telescope $\Sigma_0^{\#} \subseteq \Sigma_1^{\#} \subseteq \dots \Sigma_{k-1}^{\#} \subseteq \Sigma^{\#}$ that closely mirrors that of $\Sigma$. First of all, note that the map $atoms : S \to \mathcal{P}(S^{\#})$ naturally extends to a map on strings, $atoms : S^* \to \mathcal{P}((S^{\#})^*)$ by defining: $atoms(\epsilon) = \{\epsilon\}$, and $atoms(sw) = \{s'w' \mid s' \in atoms(s) \wedge w' \in atoms(w)\}$. Note also that the mapping $(f : s_1 \dots s_m \to s) \mapsto s_1 \dots s_m$ defines a function $arity : \Sigma \to S^*$. Define $\Sigma_0^{\#} = \{(f : w \to s^{\bullet}) \mid (f : s_1 \dots s_m \to s) \in \Sigma_0,\ \ w \in atoms(s_1 \dots s_m)\}$, where $s^{\bullet} =$ **if** $s$ atomic **then** s **else** $s^{\#}$ **fi**. Then define $\Sigma_{n+1}^{\#}$ inductively as follows: $\Sigma_{n+1}^{\#} = \Sigma_n^{\#} \cup \{(f : w \to s^{\#}) \mid (f : s_1 \dots s_m \to s) \in \Sigma_{n+1} - \Sigma_n,\ \ s\ irredundant,\ \ w \in atoms(s_1 \dots s_m) - \{arity(f : w' \to s') \mid (f : w' \to s') \in \Sigma_n^{\#}\}\}$. If $\Sigma_k = \Sigma$, we define $\Sigma_k^{\#} = \Sigma^{\#}$ and obtain a telescope $\Sigma_0^{\#} \subseteq \Sigma_1^{\#} \subseteq \dots \Sigma_{k-1}^{\#} \subseteq \Sigma^{\#}$ as claimed.

For example, for the above-mentioned signature $\Sigma$ with 0, $s$, and + subsort overloaded for sorts *Nat* and *NzNat*, the telescope for its associated $\Sigma^{\#}$ reaches its fixpoint for $\Sigma_1^{\#} = \Sigma^{\#}$ as shown in Figure 3.3.[2]

The main properties of the $\Sigma \mapsto \Sigma^{\#}$ transformation are as follows:

**Theorem 3.1** *($\Sigma^{\#}$ Correctness). Let $\Sigma$ be a regular order-sorted signature with non-empty sorts. Then:*

1. *$\Sigma^{\#}$ is sensible*

2. *for $s, s' \in S^{\#}$, $s \nequiv s' \Rightarrow T_{\Sigma^{\#},s} \cap T_{\Sigma^{\#},s'} = \emptyset$*

[2] Note that in $\Sigma$ the least sort for the terms $0 + s(0)$ and $s(0) + 0$ is *Nat*, even though we intuitively "know" that they could safely be typed with sort *NzNat*. This is because the only typing we have for + with sort *NzNat* is $\_+\_ :$ *NzNat NzNat* → *NzNat*, which requires *both* arguments to have sort *NzNat*. In $\Sigma^{\#}$ the terms $0 + s(0)$ and $s(0) + 0$ have both sort $Nat^{\#}$ using respective typings $\_+\_ : Nat^{\#}\ NzNat \to Nat^{\#}$ and $\_+\_ : NzNat\ Nat^{\#} \to Nat^{\#}$. Of course, if we had declared in $\Sigma$ the additional typings $\_+\_ : Nat\ NzNat \to NzNat$ and $\_+\_ : NzNat\ Nat \to NzNat$, we would get the more intuitive typings $\_+\_ : Nat^{\#}\ NzNat \to NzNat$ and $\_+\_ : NzNat\ Nat^{\#} \to NzNat$ in $\Sigma^{\#}$, allowing us to type both $0 + s(0)$ and $s(0) + 0$ with sort *NzNat*.

3. *for each $s \in S$, $T_{\Sigma,s} = \biguplus_{s' \in atoms(s)} T_{\Sigma^{\#},s'}$*

4. *$t \in T_{\Sigma} \ \wedge \ ls_{\Sigma}(t) = s \ \Leftrightarrow \ t \in T_{\Sigma^{\#}} \ \wedge \ ls_{\Sigma^{\#}}(t) = s^{\bullet}$.*

**Proof 3.1** *It is easy to prove by structural induction that any term in a sensible many-sorted signature has a unique sort, so that (2) follows from (1). For the same reason, (1) makes $\Sigma^{\#}$ trivially regular. Furthermore, from the definition of the function atoms and (4), we can easily obtain (3). So, we only need to prove (1) and (4). Since $\Sigma^{\#}$ is sensible iff each $\Sigma^{\#}_{n}$ is sensible, we can prove (1) by proving that $\Sigma^{\#}_{n}$ is sensible for each $n$ by induction on $n$.*
**Base case***: $n = 0$. We can prove $\Sigma^{\#}_{0}$ sensible by contradiction. Suppose that we have two typings $f : w^{\bullet} \to s^{\bullet}$ and $f : w^{\bullet} \to s'^{\bullet}$ in $\Sigma^{\#}_{0}$ with $s \neq s'$. By the definition of $\Sigma_0$, this can only happen if $f$ is not a constant and we have two different subsort-overloaded typings $f : u \to s, f : u' \to s' \in \Sigma_0$ and $w^{\bullet} \in atoms(u) \cap atoms(u')$. By the definition of the function atoms this can only happen if we have $w \leqslant u, u'$. By regularity this requires the existence of $f : w' \longrightarrow s''$ in $\Sigma$ with $w \leqslant w' \leqslant u, u'$ and $w's'' \leqslant ws, ws'$. By the definition of $\Sigma_0$ this can only happen if either $w's'' = ws$ and $ws < ws'$, or $w's'' = ws'$ and $ws' < ws$; but in either case we cannot have $f : u \to s, f : u' \to s' \in \Sigma_0$.*
**Induction Step***. We assume that $\Sigma^{\#}_{n}$ is sensible and prove $\Sigma^{\#}_{n+1}$ sensible. Note that, by the definition of $\Sigma^{\#}_{n+1}$, $f : w^{\bullet} \to s^{\#} \in \Sigma^{\#}_{n+1} - \Sigma^{\#}_{n}$ iff there is an $f : u \to s \in \Sigma_{n+1} - \Sigma_{n}$ with $s$ irredundant and $w^{\bullet} \in atoms(u) - \{arity(f : w'' \to s'') \mid (f : w'' \to s'') \in \Sigma^{\#}_{n}\}$. Therefore, since $\Sigma^{\#}_{n}$ is sensible, a failure of $\Sigma^{\#}_{n+1}$ being sensible can only happen with two different typings $f : w^{\bullet} \to s^{\#}, f : w^{\bullet} \to s'^{\#} \in \Sigma^{\#}_{n+1} - \Sigma^{\#}_{n}$. This means that we have two different subsort-overloaded $f : u \to s, f : u' \to s' \in \Sigma_{n+1} - \Sigma_{n}$ with $w^{\bullet} \in atoms(u) \cap atoms(u')$. By the definition of the function atoms this can only happen if we have $w \leqslant u, u'$. But then regularity requires the existence of $f : w' \to s'''$ with $w \leqslant w' \leqslant u, u'$ and $w's''' \leqslant us, u's'$, which forces $w^{\bullet} \in atoms(w')$. Since $w^{\bullet} \notin \{arity(f : w'' \to s'') \mid (f : w'' \to s'') \in \Sigma^{\#}_{n}\}\}$ and $w's''' \leqslant us, u's'$, we must have $f : w' \to s''' \in \Sigma_{n+1} - \Sigma_{n}$. But this then forces either $w's'' = us$ and $us < u's'$, or $w's'' = u's'$ and $u's' < us$, both contradicting $f : u \to s, f : u' \to s' \in \Sigma_{n+1} - \Sigma_{n}$.*

*The proof of (4) essentially reduces to proving the following lemma:*

***Lemma 3.1*** *Let $u \leqslant w$ be words of equal length in $S^{*}$ such that $f : w \to s' \in \Sigma$, and let $f : v \to s$ be the smallest possible typing in $\Sigma$ with $u \leqslant v \leqslant w$ and $vs \leqslant ws'$. Then, $f : u^{\bullet} \to s^{\bullet} \in \Sigma^{\#}$.*

***Proof 3.2*** *If $f : v \to s \in \Sigma_0$, this follows from the definition of $\Sigma^{\#}_{0}$. Suppose $f : v \to s \in \Sigma_{n+1} - \Sigma_{n}$. Since $\Sigma$ has non-empty sorts, there must then be terms of smallest sort $s$, so that $s$ is irredundant. Therefore, the only way in which we can fail to have $f : u^{\bullet} \to s^{\bullet} \in \Sigma^{\#}_{n+1}$ is by having some $f : u^{\bullet} \to s''^{\bullet} \in \Sigma^{\#}_{n}$. But this can only happen if there is an $f : v' \to s'' \in \Sigma_{n}$ with $u \leqslant v'$, which by regularity forces $v's'' \geqslant vs$, and, since $v's'' \neq vs$ makes $v's'' > vs$, which forces $f : v \to s \in \Sigma_{n-1}$, contradicting $f : v \to s \in \Sigma_{n+1} - \Sigma_{n}$.*

*Using (1), which ensures that $ls_{\Sigma^{\#}}(t)$ is a well-defined function, we can now prove (4) by structural induction. To see the ($\Rightarrow$) implication, note that the result is trivial if $t$ is a constant, so let $f(t_1, \ldots, t_n) \in T_{\Sigma} \ \wedge \ ls_{\Sigma}(f(t_1, \ldots, t_n)) = s$, with $ls_{\Sigma}(t_i) = s_i$, $1 \leqslant i \leqslant n$. By the induction hypothesis we then have $t_i \in T_{\Sigma^{\#}} \ \wedge \ ls_{\Sigma^{\#}}(t_i) = s^{\bullet}_{i}$, $1 \leqslant i \leqslant n$. Furthermore, $ls_{\Sigma}(f(t_1, \ldots, t_n)) = s$ and regularity imply that we have an $f : v \to s \in \Sigma$ with $u = s_1 \ldots s_n \leqslant v$ and $vs$ smallest possible with this property. But then Lemma 3.1 ensures the existence of $f : u^{\bullet} \to s^{\bullet} \in \Sigma^{\#}$, proving $f(t_1, \ldots, t_n) \in T_{\Sigma^{\#}} \ \wedge \ ls_{\Sigma^{\#}}(f(t_1, \ldots, t_n)) = s$, as desired. For the ($\Leftarrow$) implication, constants are again trivial. Also let $f(t_1, \ldots, t_n) \in T_{\Sigma^{\#}} \ \wedge \ ls_{\Sigma^{\#}}(f(t_1, \ldots, t_n)) = s^{\bullet}$, with $ls_{\Sigma^{\#}}(t_i) = s^{\bullet}_{i}$, $1 \leqslant i \leqslant n$. The induction hypothesis then gives us $t_i \in T_{\Sigma} \wedge ls_{\Sigma}(t_i) = s_i$, $1 \leqslant i \leqslant n$. Since* *for $u = s_1 \ldots s_n$ we have $f : u^\bullet \to s^\bullet \in \Sigma^\#$, by the construction of $\Sigma^\#$ we must have some $f : w \to s' \in \Sigma$ with $u \leqslant w$, and if $f : v \to s''$ is the smallest possible typing in $\Sigma$ with $u \leqslant v \leqslant w$ and $vs'' \leqslant ws'$, then Lemma 3.1 ensures the existence of $f : u^\bullet \to s''^\bullet \in \Sigma^\#$, and $\Sigma^\#$ sensible forces $s = s''$. Therefore, $f(t_1, \ldots, t_n) \in T_\Sigma \wedge ls_\Sigma(f(t_1, \ldots, t_n)) = s$, as desired.*

### 3.2.1 Variations on the $\Sigma^\#$ Theme

Several signatures closely related to $\Sigma$ and $\Sigma^\#$ are also very useful. The most obvious is their union $\Sigma \cup \Sigma^\#$, with set of operators the set-theoretic union $\Sigma \cup \Sigma^\#$ and poset of sorts $(S \cup S^\#, (\leqslant \cup <^\#)^*)$, with $\leqslant$ the order in $(S, \leqslant)$, and $<^\# = \{(s^\#, s) \mid s$ *nonatomic and irredundant*$\}$. $\Sigma \cup \Sigma^\#$ is even more intuitive than $\Sigma^\#$, because it *refines* $\Sigma$ into a richer semantics-preserving signature by just adding to it the new atoms $s^\#$, so that now the least sort of any ground term $t$ will always be an atomic sort. This means that we have *sharpened* the typing of any such $t$ as much as possible, which is the reason for the $\Sigma^\#$ notation. For example, for $\Sigma$ the signature on the left side of Figure 3.1, with subsort inclusion $A < B$, constants $a$ of sort $A$, $b$ of sort $B$, and subsort-overloaded unary operator $f$, $\Sigma \cup \Sigma^\#$ is the signature depicted on the right side of Figure 3.1.

Note that we have subsignature inclusions $J : \Sigma \hookrightarrow \Sigma \cup \Sigma^\#$ and $J' : \Sigma^\# \hookrightarrow \Sigma \cup \Sigma^\#$. Furthermore, $\Sigma \cup \Sigma^\#$ enjoys very good properties, which make it an initial-semantics-preserving enrichment of both $\Sigma$ and $\Sigma^\#$:

**Lemma 3.2** *$\Sigma \cup \Sigma^\#$ is regular, $T_{\Sigma \cup \Sigma^\#} |_J = T_\Sigma$, and $T_{\Sigma \cup \Sigma^\#} |_{J'} = T_{\Sigma^\#}$.*

**Proof 3.3** *Let $u \in (S \cup S^\#)^*$ be such that there is an $f : w \to s'$ in $\Sigma \cup \Sigma^\#$ and $u \leqslant w$. If $u = u'^\bullet$, then the construction of $\Sigma^\#$, Lemma 3.1, and the order $(\leqslant \cup <^\#)^*$ ensure that there is a smallest possible typing of the form $f : u'^{bullet} \to s^\bullet$. Otherwise, $u \notin (S^\#)^*$, say, $u = s_1 \ldots s_{i_1}^\bullet \ldots s_{i_k}^\bullet \ldots s_n$, with $0 \leqslant k < n$. But then the only $f : w \to s'$ in $\Sigma \cup \Sigma^\#$ with $u \leqslant w$ and those $f : w \to s'$ in $\Sigma$ with $s_1 \ldots s_{i_1} \ldots s_{i_k} \ldots s_n \leqslant w$, for which there is one with smallest possible typing by the regularity of $\Sigma$. The identity $T_{\Sigma \cup \Sigma^\#} |_{J'} = T_{\Sigma^\#}$ follows easily from the fact that the atomic sorts of $\Sigma \cup \Sigma^\#$ are precisely the sorts in $S^\#$, and the only operators relating those sorts are exactly those in $\Sigma^\#$. Note also that it follows easily from Theorem 3.1 that, as sets of terms, we have $T_{\Sigma \cup \Sigma^\#} = T_\Sigma = T_{\Sigma^\#}$, and that for any term in that set we have $ls_{\Sigma \cup \Sigma^\#}(t) = ls_{\Sigma^\#}(t)$. Therefore, to prove $T_{\Sigma \cup \Sigma^\#} |_J = T_\Sigma$ we just have to show that for each $s \in S$ we have $T_{\Sigma \cup \Sigma^\#, s} = T_{\Sigma, s}$. But since in the order $(\leqslant \cup <^\#)^*$ the atoms below any $s \in S$ are precisely the set of sorts $atoms(s)$, the equality $ls_{\Sigma \cup \Sigma^\#}(t) = ls_{\Sigma^\#}(t)$ and (3) in Theorem 3.1 give us $T_{\Sigma \cup \Sigma^\#, s} = T_{\Sigma, s}$, as desired.*

Two other useful signatures are $\Sigma^\#_\top$ and $\Sigma^\#_c$. $\Sigma^\#_\top$ is an order-sorted signature with operations those in $\Sigma^\#$ and with sorts $S_\top \cup S^\#$, where $S_\top = \{\top_{[s]} \mid [s] \in \hat{S}\}$ is the set of top sorts of each connected component in $(S, \leqslant)$. Its order is defined as the identity relation on $S^\# \cup S_\top$, plus the subsort inclusions $s' \leqslant \top_{[s]}$ for each $s' \in atoms(\top_{[s]})$. We have a subsignature inclusion $K : \Sigma^\#_\top \hookrightarrow \Sigma \cup \Sigma^\#$. Reasoning as in Lemma 3.2 it is easy to show that $T_{\Sigma \cup \Sigma^\#} |_K = T_{\Sigma^\#_\top}$.

$\Sigma^\#_c$ is a *many-sorted version* of $\Sigma^\#_\top$. Its set of sorts is $S_\top \cup S^\#$, but now $s \leqslant s'$ iff $s = s'$. The operations of $\Sigma^\#_c$ are those of $\Sigma^\#$ plus the coercion operators $\{c : s' \to \top_{[s]} \mid [s] \in \hat{S},\ \ s' \in atoms(\top_{[s]}) - \{\top_{[s]}\}\}$, which mimic the subsort inclusions $s' < \top_{[s]}$ in $\Sigma^\#_\top$. We then have a signature morphism $H : \Sigma^\#_c \to \Sigma^\#_\top$ that is the identity on sorts and on the operators in $\Sigma^\#$ and maps each coercion $c : s' \to \top_{[s]}$ to the term $x_1{:}s'$.

For example, for the above-mentioned signature $\Sigma$ with $0$, $s$, and $+$ subsort overloaded for sorts *Nat* and *NzNat*, the signatures $\Sigma^\#_\top$ and $\Sigma^\#_c$ are shown in Figure 3.4.

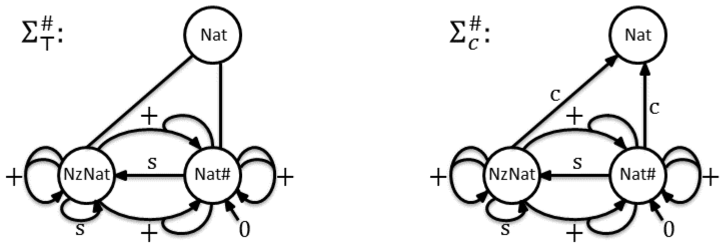


Figure 3.4: Signatures $\Sigma_{\top}^{\#}$ and $\Sigma_{c}^{\#}$

The following diagram summarizes this section:

$$\Sigma \xhookrightarrow{J} \Sigma \cup \Sigma^{\#} \xhookleftarrow{K} \Sigma_{\top}^{\#} \xleftarrow{H} \Sigma_{c}^{\#} \tag{3.3}$$

## 3.3 EQUATIONAL FORMULAS IN INITIAL ORDER-SORTED ALGEBRAS

The main goal of this section is to *reduce* the validity of equational first-order formulas in an initial *order-sorted* algebra to the validity of semantically equivalent formulas in an initial *many-sorted* algebra. The main idea of this reduction is to exploit diagram 3.3 at the end of Section 3.2.1, which begins with an order-sorted signature $\Sigma$ and ends with a many-sorted signature $\Sigma_{c}^{\#}$. Like Alice in Wonderland's Cheshire cat's smile, all order-sorted features vanish in the passage from $\Sigma$ to $\Sigma_{c}^{\#}$. This reduction seems useful for at least three reasons:

1. Its provides a new, very simple proof of the *decidability* of first-order formulas in initial order-sorted algebras. A non-trivial proof of such a decidability result goes back to [17, 18], but it requires quite complex formulas and formula transformations involving sort membership constraints based on quite general sort expressions, whose semantics is defined using tree automata.

2. The reduction-based proof given here provides a useful new *transfer principle*, by which problems with a perhaps unclear solution at the order-sorted level can be reduced to problems having a clear solution at the many-sorted level. For example, as further explained in Section 3.4, the puzzling anomaly about pattern operations in initial order-sorted algebras discussed in the Introduction has a systematic solution thanks to this transfer principle.

3. Above reasons (1)–(2) embrace each other in a useful way in Section 3.5, where we show how one can faithfully reduce the problem of whether a formula $\varphi$ involving sort membership constraints in the richer language of [17, 18] is valid in an initial order-sorted algebra $T_{\Sigma}$ to the same problem for a purely equational formula $\pi(\varphi)$ in the order-sorted algebra $T_{\Sigma\cup\Sigma^{\#}}$. Thanks to this reduction, all the needed tree automata operations are eliminated through the translation $\varphi \mapsto \pi(\varphi)$, which uses simple pattern operations.

The main idea of the reduction is to assign to each first-order sentence $\varphi$ in the language of a finite and regular order-sorted signature $\Sigma$ a corresponding sentence $\varphi_{c}^{\#}$ in the language of the many-sorted signature $\Sigma_{c}^{\#}$, and then prove that we have an equivalence $T_{\Sigma} \models \varphi \;\Leftrightarrow\; T_{\Sigma_{c}^{\#}} \models \varphi_{c}^{\#}$. To obtain such an equivalence we make our way from $T_\Sigma$ and $\varphi$ to $T_{\Sigma_c^\#}$ and $\varphi_c^\#$ by moving from left to right along the diagram 3.3. Since some of the steps in this sequence of signature morphisms are easy consequences of the equivalence (†) in Section 2.1.2, we can quickly get such easy equivalences out of the way. Indeed, since $J$ is a subsignature inclusion, it is the identity on formulas, and since by Lemma 3.2 we have the equality $T_{\Sigma\cup\Sigma^\#}\mid_J = T_\Sigma$, (†) applied to $J$ gives us the equivalence $T_\Sigma \models \varphi \;\Leftrightarrow\; T_{\Sigma\cup\Sigma^\#} \models \varphi$. On the leftmost side, (†) gives us the equivalence $T_{\Sigma_\top^\#} \models H(\varphi_c^\#) \;\Leftrightarrow\; T_{\Sigma_\top^\#}\mid_H \models \varphi_c^\#$. The interesting twist, however, is that the unique $\Sigma_c^\#$-homomorphism $h : T_{\Sigma_c^\#} \to T_{\Sigma_\top^\#}\mid_H$ from the initial $\Sigma_c^\#$-algebra $T_{\Sigma_c^\#}$ is obviously the identity on the sorts $S^\#$ and maps each term $c(t) \in T_{\Sigma_c^\#,\top_{[s]}}$ to the term $t \in T_{\Sigma_\top^\#,\top_{[s]}}$. That is, $h$ is *bijective*, and therefore a $\Sigma_c^\#$-*isomorphism* $h : T_{\Sigma_c^\#} \cong T_{\Sigma_\top^\#}\mid_H$, which gives us the equivalence $T_{\Sigma_\top^\#}\mid_H \models \varphi_c^\# \;\Leftrightarrow\; T_{\Sigma_c^\#} \models \varphi_c^\#$. Therefore, stringing these last two equivalences together, we get the equivalence $T_{\Sigma_\top^\#} \models H(\varphi_c^\#) \;\Leftrightarrow\; T_{\Sigma_c^\#} \models \varphi_c^\#$. We will then be done proving our desired equivalence $T_\Sigma \models \varphi \;\Leftrightarrow\; T_{\Sigma_c^\#} \models \varphi_c^\#$ if we can define a mapping $\varphi \mapsto \varphi^\#$ such that $H(\varphi_c^\#) = \varphi^\#$ and we show an equivalence $T_{\Sigma\cup\Sigma^\#} \models \varphi \;\Leftrightarrow\; T_{\Sigma_\top^\#} \models \varphi^\#$.

What makes the mapping $\varphi \mapsto \varphi^\#$ not entirely obvious is that $\Sigma_\top^\#$ has considerably fewer sorts than the plentiful $\Sigma\cup\Sigma^\#$. In particular, we have to find a way to express equations and quantifiers involving variables with sorts of $\Sigma\cup\Sigma^\#$ not present in $\Sigma_\top^\#$ in the poorer language of $\Sigma_\top^\#$. The key idea for this is to observe that every ground $\Sigma\cup\Sigma^\#$-term has an atomic least sort in $S^\#$, and that, by Theorem 3.1–(3) and Lemma 3.2, we have the equality $T_{\Sigma\cup\Sigma^\#,s} = \biguplus_{s'\in atoms(s)} T_{\Sigma^\#,s'}$. Therefore, abbreviating $t \in T_{\Sigma\cup\Sigma^\#,s}$ to $t : s$, we have, $t : s \;\Leftrightarrow\; \bigvee_{s'\in atoms(s)} t : s'$, which is a property expressible in the language of $\Sigma_\top^\#$. Here is now the detailed mapping $\varphi \mapsto \varphi^\#$ using these ideas. Without loss of generality we may assume $\varphi$ in prenex form, that is, $\varphi = Q\varphi_0$, with $Q$ a sequence of quantifiers and $\varphi_0$ quantifier-free. The mapping $\varphi \mapsto \varphi^\#$ decomposes into a mapping $\varphi_0 \mapsto \varphi_0^\#$ for the quantifier-free part and a mapping for the quantifiers.

We first need some notation. $\overline{x{:}s}$ abbreviates a sequence of variables $x_1{:}s_1,\ldots,x_n{:}s_n$. We can always decompose the free variables of $\varphi_0$ as $\mathit{fvars}(\varphi_0) = \overline{x{:}s},\overline{y{:}p}$, with $\overline{x{:}s}$ variables having non-atomic sorts, and $\overline{y{:}p}$ variables having atomic sorts. Also, if $\overline{x{:}s} = x_1 : s_1,\ldots,x_n : s_n$, then $\overline{x{:}s}_\top$ denotes the variables $\overline{x{:}s}_\top = x_1 : \top_{[s_1]},\ldots,x_n : \top_{[s_n]}$. In the same spirit, $\overline{x{:}s} = \bar{t}$ abbreviates the *conjunction of equations* $x_1 : s_1 = t_1 \wedge \ldots \wedge x_n{:}s_n = t_n$, and $\{\overline{x{:}s} = \bar{t}\}$ abbreviates the *substitution* $\{x_1{:}s_1 \mapsto t_1,\ldots,x_n{:}s_n \mapsto t_n\}$. Given variables $\overline{x{:}s}$ with sorts in $S$, let $\mathit{Spec}(\overline{x{:}s}, S^\#)$, called the set of $S^\#$-*specializations* of $\overline{x{:}s}$, be the set $\mathit{Spec}(\overline{x{:}s}, S^\#) = \{\overline{x{:}s} = \overline{z{:}q} \mid |\overline{x{:}s}| = |\overline{z{:}q}| \wedge q_i \in atoms(s_i), 1 \leqslant i \leqslant |x : s|\}$, where $|\overline{x{:}s}|$ denotes the length of the sequence of variables $\overline{x{:}s}$. To avoid variable capture we will always assume that the variables $\overline{z{:}q}$ are *fresh* variables, different for each $(\overline{x{:}s} = \overline{z{:}q}) \in \mathit{Spec}(\overline{x{:}s}, S^\#)$ and not appearing anywhere else. Viewed as a substitution $\{\overline{x{:}s} = \overline{z{:}q}\}$, each specialization $\overline{x{:}s} = \overline{z{:}q}$ is just a variable mapping lowering the sort $s_i$ of each $x_i$ to a sort $q_i \in atoms(s_i)$ for $z_i$. We can now define the mapping $\varphi_0 \mapsto \varphi_0^\#$ —where $\mathit{fvars}(\varphi_0) = \overline{x{:}s},\overline{y{:}p}$, with $\overline{x{:}s}$ variables having non-atomic sorts, and $\overline{y{:}p}$ variables having atomic sorts— as follows:

$$[\varphi_0^\# = \bigvee_{(\overline{x{:}s}=\overline{z{:}q})\in \mathit{Spec}(\overline{x{:}s},S^\#)} (\exists\overline{z{:}q})\ (\overline{x{:}s}_\top = \overline{z{:}q} \;\wedge\; (\varphi_0\{\overline{x{:}s} = \overline{z{:}q}\})). \tag{3.4}$$

Note that $\mathit{fvars}(\varphi_0^\#) = \overline{x{:}s}_\top,\overline{y{:}p}$. For example, for the above-mentioned signature $\Sigma$ with 0, $s$, and $+$ subsort overloaded for sorts $\mathit{Nat}$ and $\mathit{NzNat}$, if $\varphi$ is $x + y = y + x$, with $x, y : \mathit{Nat}$, then, assuming $x_1, y_1 : \mathit{NzNat}$, and $x_2, y_2 : \mathit{Nat}^\#$, $\varphi^\#$ is:

$$\begin{aligned}&(\exists x_1, y_1)\; x = x_1 \wedge y = y_1 \wedge x_1 + y_1 = y_1 + x_1 \vee\\ &(\exists x_2, y_2)\; x = x_2 \wedge y = y_2 \wedge x_2 + y_2 = y_2 + x_2 \vee\\ &(\exists x_1, y_2)\; x = x_1 \wedge y = y_2 \wedge x_1 + y_2 = y_2 + x_1 \vee\\ &(\exists x_2, y_1)\; x = x_2 \wedge y = y_1 \wedge x_2 + y_1 = y_1 + x_2.\end{aligned} \tag{3.5}$$

The semantic equivalence between $\varphi_0$ and $\varphi_0^{\#}$ can then be expressed as follows:

**Lemma 3.3** *For $\varphi_0$ as above, $\alpha \in [\overline{x{:}s}_{\top}, \overline{y{:}p} \to T_{\Sigma^{\#}_{\top}}]$ satisfies $T_{\Sigma^{\#}_{\top}}, \alpha \models \varphi_0^{\#}$ iff there exists $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_{\Sigma}]$ such that $\alpha = \beta \circ \{\overline{x{:}s}_{\top} = \overline{x{:}s}\}$ and $T_{\Sigma}, \beta \models \varphi_0$.*

**Proof 3.4** *To see the $(\Rightarrow)$ implication, note that $T_{\Sigma^{\#}_{\top}}, \alpha \models \varphi_0^{\#}$ iff there is an $(\overline{x : s} = \overline{z : q}) \in Spec(\overline{x : s}, S^{\#})$ such that $T_{\Sigma^{\#}_{\top}}, \alpha \models (\exists \overline{z{:}q})\; (\overline{x{:}s}_{\top} = \overline{z{:}q} \;\wedge\; (\varphi_0\{\overline{x{:}s} = \overline{z{:}q}\}))$. For each $1 \leqslant i \leqslant |\overline{x : s}|$ this forces $\alpha(x_i{:}\top_{[s_i]}) \in T_{\Sigma^{\#}, q_i}$, and, since $q_i \in atoms(s_i)$, by Theorem 3.1–(3) we must have $\alpha(x_i{:}\top_{[s_i]}) \in T_{\Sigma, s_i}$. This means that there is a $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_{\Sigma}]$ with $\beta\,|_{\overline{y:p}} = \alpha\,|_{\overline{y:p}}$, and $\beta(x_i{:}s_i) = \alpha(x_i{:}\top_{[s_i]})$, $1 \leqslant i \leqslant |\overline{x : s}|$, such that $\alpha = \beta \circ \{\overline{x{:}s}_{\top} = \overline{x{:}s}\}$. To see that $T_{\Sigma}, \beta \models \varphi_0$ it is enough to reason by structural induction on the Boolean structure of $\varphi_0$ and show that for each equation $u = v$ appearing in $\varphi_0$, assuming $k = |\overline{z{:}q}|$, we have the equivalence: $T_{\Sigma^{\#}_{\top}}, \alpha\,|_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\} \models (u = v)\{\overline{x{:}s} = \overline{z{:}q}\} \;\Leftrightarrow\; T_{\Sigma}, \beta \models u = v$. But this is trivial, since by the definition of $\beta$ we have $u\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\}) = u\beta$, and $v\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\}) = v\beta$.*

*The $(\Leftarrow)$ implication can also be reduced to the case $\varphi_0 = u = v$, with $fvars(u = v) = \overline{x{:}s}, \overline{y{:}p}$. For any $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_{\Sigma}]$ such that $T_{\Sigma}, \beta \models u = v$, let $q_i = ls_{\Sigma}(\beta(x_i{:}s_i))^{\bullet}$, and let $\alpha = \beta \circ \{\overline{x{:}s}_{\top} = \overline{x{:}s}\}$. Then we have $u\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\}) = u\beta = v\beta = v\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\})$, and therefore $T_{\Sigma^{\#}_{\top}}, \alpha \models (\exists \overline{z{:}q})\; (\overline{x{:}s}_{\top} = \overline{z{:}q} \;\wedge\; (\varphi_0\{\overline{x{:}s} = \overline{z{:}q}\}))$, which proves $T_{\Sigma^{\#}_{\top}}, \alpha \models \varphi_0^{\#}$, as desired.*

Since $\varphi = Q\varphi_0$, to define $\varphi^{\#}$ we still need to deal with the quantifiers $Q$. This is done inductively for each individual quantifier as follows. If $s \in S \cap (S^{\#} \cup S_{\top})$, then $((\forall x{:}s)\; \varphi)^{\#} = (\forall x{:}s)\; \varphi^{\#}$, and $((\exists x{:}s)\; \varphi)^{\#} = (\exists x{:}s)\; \varphi^{\#}$. Otherwise, let $atoms(s) = \{q_1, \ldots, q_k\}$, then, $((\forall x{:}s)\; \varphi)^{\#} = (\forall x{:}\top_{[s]})\; (((\exists \overline{z{:}q})\; \bigvee_{i=1}^{k} x{:}\top_{[s]} = z_i{:}q_i) \Rightarrow \varphi^{\#})$, and $((\exists x{:}s)\; \varphi)^{\#} = (\exists x{:}\top_{[s]}, \overline{z{:}q})\; (\bigvee_{i=1}^{k} x{:}\top_{[s]} = z_i{:}q_i) \;\wedge\; \varphi^{\#})$.

The key syntactic invariant maintained by this translation is of course that if $fvars(\varphi) = \overline{x{:}s}, \overline{y{:}p}$, then $fvars(\varphi^{\#}) = \overline{x{:}s}_{\top}, \overline{y{:}p}$. And the key semantic invariant is that for each $\alpha \in [\overline{x{:}s}_{\top}, \overline{y{:}p} \to T_{\Sigma^{\#}_{\top}}]$ we have $T_{\Sigma^{\#}_{\top}}, \alpha \models \varphi^{\#}$ iff there exists $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_{\Sigma}]$ such that $\alpha = \beta \circ \{\overline{x{:}s}_{\top} = \overline{x{:}s}\}$ and $T_{\Sigma}, \beta \models \varphi$. For quantifier-free formulas this has already been proved in Lemma 3.3. That this remains true after each quantification step is easy to check and left to the reader: indeed, the above treatment of quantifiers is analogous to how in set theory we restrict quantifiers ranging over all sets to quantifiers ranging over a given set $A$ by defining $(\forall x \in A)\; \varphi = (\forall x)\; (x \in A \Rightarrow \varphi)$, and $(\exists x \in A)\; \varphi = (\exists x)\; (x \in A \;\wedge\; \varphi)$. Our treatment is not just analogous, but in fact a special case: we have just captured $x \in T_{\Sigma, s}$ by the formula $(\exists \overline{z{:}q})\; \bigvee_{i=1}^{k} x = z_i{:}q_i$. Therefore, for any sentence $\varphi$ (i.e., $fvars(\varphi) = \varnothing$) we get $T_{\Sigma} \models \varphi \;\Leftrightarrow\; T_{\Sigma^{\#}_{\top}} \models \varphi^{\#}$.

To close all the proof steps along the Cheshire cat's sequence 3.3 we need to define the formula $\varphi_c^{\#}$ such that $H(\varphi_c^{\#}) = \varphi^{\#}$. We can get $\varphi_c^{\#}$ from $\varphi^{\#}$ as follows. Since $\Sigma^{\#}_{\top}$ and $\Sigma^{\#}_{c}$ have the same sorts, the variables and quantifiers in $\varphi^{\#}$ and $\varphi_c^{\#}$ stay the same. We just replace each equation $u = v$ appearing somewhere in $\varphi^{\#}$ by the equation $c(u) = c(v)$ at the same position in $\varphi_c^{\#}$, unless: (i) $\top_{[s]}$ is atomic (then $u = v$ is left unchanged), or (ii) $\top_{[s]}$ is non-atomic and either $u$ or $v$ are variables of sort $\top_{[s]}$, which are then left unchanged. This gives us the desired semantic equivalence $T_{\Sigma} \models \varphi \;\Leftrightarrow\; T_{\Sigma^{\#}_{c}} \models \varphi_c^{\#}$.

Since both the technical report version [15] of Maher's paper [14], and the disunification paper by Comon and Lescanne [16] prove that the first-order theory of a *many-sorted* initial algebra $T_\Omega$ is *decidable* —i.e., that there is an algorithm to decide for any formula $\phi$ whether $T_\Omega \models \phi$ holds or not— we then get as a corollary of the above equivalence the following theorem,[3] already known since [17, 18], but now obtained in a different way and with a considerably simpler proof:

**Theorem 3.2** *Let $\Sigma$ be a finite and regular order-sorted signature. For any first-order formula $\varphi \in Form(\Sigma)$ the validity problem $T_\Sigma \models \varphi$ is decidable.* □

## 3.4 PATTERN OPERATIONS IN INITIAL ORDER-SORTED ALGEBRAS

Given an order-sorted signature $\Sigma$, by a $\Sigma$-*pattern* we just mean a term $t \in T_\Sigma(X)$, where we assume $X_s$ countably infinite for each sort $s \in S$. We call $t$ a pattern to emphasize that $t$ is a symbolic description of a *language*, namely the set $[\![t]\!] = \{t\sigma \mid \sigma \in [X \to T_\Sigma]\}$ of its *ground instances*. Similarly, a finite set of patterns $\{t_1, \ldots, t_n\}$ is a symbolic description of the language $[\![t_1, \ldots, t_n]\!] = [\![t_1]\!] \cup \ldots \cup [\![t_n]\!]$. A language need not be a set of strings. Since strings are just a special case of trees, it can be a *tree language*, that is, a subset $L \subseteq T_\Sigma$ for some $\Sigma$. Therefore, $\mathcal{P}(T_\Sigma)$ is the set of all $\Sigma$-tree languages, and we have a function

$$[\![\_]\!] : \mathcal{P}_{fin}(T_\Sigma(X)) \longrightarrow \mathcal{P}(T_\Sigma) : \{t_1, \ldots, t_n\} \mapsto [\![t_1, \ldots, t_n]\!] \tag{3.6}$$

sending each finite set of patterns to its associated language. Call a language $L \subseteq T_\Sigma$ a *pattern language* iff $L = [\![t_1, \ldots, t_n]\!]$ for some finite set of patterns $\{t_1, \ldots, t_n\}$. The most obvious question is that of *representability*: which languages $L \subseteq T_\Sigma$ are pattern languages, i.e., can be symbolically *represented* by some $\{t_1, \ldots, t_n\}$? Pattern languages are closed under finite unions by construction. Are they closed under finite intersections? Obviously *yes*, since, by distributivity we can reduce the problem to the intersection of two patterns $[\![u]\!] \cap [\![v]\!]$, and we have $[\![u]\!] \cap [\![v]\!] = [\![u\sigma_1]\!] \cup \ldots \cup [\![u\sigma_n]\!]$, where $\{\sigma_1, \ldots, \sigma_n\} = DUnif_\Sigma(u, v)$, the set of *most general disjoint order-sorted unifiers* of $u$ and $v$ in $\Sigma$ [51]; that is, before unifying $u$ and $v$, we rename $v$ if necessary to make its variables disjoint from those of $u$. Are $T_\Sigma$ and $\varnothing$ pattern languages? Yes: $\varnothing = [\![\varnothing]\!]$, and $T_\Sigma = [\![x_1 \!:\! \top_{[s_1]}, \ldots, x_k \!:\! \top_{[s_k]}]\!]$, where $\hat{S} = \{[s_1], \ldots, [s_k]\}$. So, the only missing Boolean operation is *complement*. But since complement and difference are expressible in terms of each other: $\overline{A} = \top - A$, and $A - B = A \cap \overline{B}$, we can rephrase the question thus: are pattern languages closed under differences? In general they are *not*. For example, for $\Sigma$ unsorted and having a constant $a$ and a binary $f$, the language $[\![f(x, y)]\!] - [\![f(z, z)]\!]$ is *not* a pattern language (see Prop. 4.5 in [7]). However, in the unsorted case (see Corollary in pg. 314, [7]) $[\![t_1, \ldots, t_n]\!] - [\![t'_1, \ldots, t'_m]\!]$ *is* a pattern language when the $t_i$ and the $t'_j$ are *linear* terms, and more general cases than just sets of linear patterns also yield differences that are pattern languages [7, 8, 9, 10, 11].

### 3.4.1 Many- and Order-Sorted Pattern Difference Algorithms

Since all other Boolean operations are already taken care of, all we need is a way of symbolically defining the *difference* $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ of two finite sets of *order-sorted* patterns whenever this represents a pattern language. As illustrated by the example in Figure 3.1, if we insist on remaining in the given signature

[3] Theorem 3.2 holds for $\Sigma$ finite and regular because any such $\Sigma$ can be transformed into a semantically equivalent signature with no ad-hoc overloading (by symbol renaming) and with each connected component having a top sort (added when missing).

$\Sigma$ this cannot be done, even for sets of linear patterns. However, we can use the $\Sigma \mapsto \Sigma^{\#}$ transformation and the *transfer principle* from order-sorted problems to many-sorted ones discussed in Section 3.3 to obtain a solution based on the following two simple observations:

1. As *sets* (not as algebras) we have $T_\Sigma = T_{\Sigma^{\#}}$.
2. For any *order-sorted* pattern $t \in T_\Sigma(X)$ we have the *language equality* $[\![t]\!] = \bigcup_{(\overline{x:s}=\overline{z:q}) \in Spec(\overline{x:s},S^{\#})} [\![t\{\overline{x:s} = \overline{z:q}\}]\!]$, where $\overline{x:s} = \mathit{fvars}(t)$.

where both (1) and (2) are simple corollaries of Theorem 3.1. This then yields a straightforward way of representing a difference of finite sets of *order-sorted* $\Sigma$-patterns $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ as a difference of finite sets of *many-sorted* $\Sigma^{\#}$-patterns: we just replace each $t_i$ (resp $t'_j$) by the finite set of many-sorted $\Sigma^{\#}$-patterns $\{t_i\{\overline{x:s} = \overline{z:q}\} \mid (\overline{x:s} = \overline{z:q}) \in Spec(\overline{x:s}, S^{\#})\}$, where $\overline{x:s} = \mathit{fvars}(t_i)$. For the example in Figure 3.1, this method transforms the order-sorted symbolic difference $\{x{:}B\} - \{y{:}A\}$ into the many-sorted symbolic difference: $\{x{:}A, z{:}B^{\#}\} - \{y{:}A\}$.

Since —with the possible exception of the treatment of *finite sorts* (see below), which warrants an extension of the unsorted algorithms— the unsorted algorithms for computing the symbolic difference of two sets of patterns have a straightforward generalization to the many-sorted case, we can just use the above reduction to the many-sorted case and many-sorted versions of the difference algorithms in [7, 9, 10, 11] to solve the problem of computing when possible the symbolic difference of order-sorted patterns $\{t_1, \ldots, t_n\}-\{t'_1, \ldots, t'_m\}$ as a finite set of (many-sorted) patterns.

But is this the best we can do? There can be some practical limitations, both in performance and at the representational level. For order-sorted signatures with rich type structures a set $atoms(s)$ may have a considerable number of sorts in $S^{\#}$, so that the sets $\{t_i\{\overline{x:s} = \overline{z:q}\} \mid (\overline{x:s} = \overline{z:q}) \in Spec(\overline{x:s}, S^{\#})\}$ for each $t_i$ (resp. $t'_i$) can become quite big, affecting performance. It also means that the representation of the *solutions* to symbolic difference problems, besides being possibly quite big, may also be more *verbose* than necessary. For example, in the signature in the right side of Figure 3.1, we can compute the *order-sorted* symbolic difference $\{x:B\} - \{b\} = \{f(y:B), a\}$, which is shorter and more intuitive than the equivalent many-sorted representation $\{x{:}B\} - \{b\} = \{f(z{:}B^{\#}), f(z'{:}A), a\}$.

We present below an attractive alternative, namely, an *order-sorted* algorithm for computing symbolic differences $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ in the extended order-sorted signature $\Sigma \cup \Sigma^{\#}$ that does not require any transformation of the original problem and can significantly overcome the above limitations by yielding simpler and shorter representations and better performance (see Section 3.6.1).

Let us describe this algorithm. First of all, thanks to the Boolean equation $(A \cup B) - C = (A - C) \cup (B - C)$, we can decompose $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ as a union $\{t_1\} - \{t'_1, \ldots, t'_m\} \cup \ldots \cup \{t_n\} - \{t'_1, \ldots, t'_m\}$. Second, thanks to the Boolean equation $A - B = A - (A \cap B)$ we can reduce $\{t\} - \{t_1, \ldots, t_n\}$ to the equivalent symbolic expression $\{t\} - \{t\sigma \mid \sigma \in DUnif_\Sigma(t, t_1) \cup \ldots \cup DUnif_\Sigma(t, t_n)\}$. Thus, all our symbolic difference problems can be reduced to unions of problems of the form $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ with $\sigma_1, \ldots, \sigma_n$ substitutions instantiating $t$.

The algorithm below gives priority to the easier and frequently occurring cases, using the order-sorted extension of the more general algorithm of Lassez and Marriott [7] only when the simpler algorithms cannot be applied. We also exploit the fact that a sort $s$ may be *finite* —i.e., $T_{\Sigma\cup\Sigma^{\#},s}$ is a finite set— plus the decidability of sort finiteness to increase the successful difference cases.

**Order-Sorted Pattern Difference Algorithm**

1. If $t, t\sigma_1, \ldots, t\sigma_n$ are all *linear* terms, we apply the inference rules below.

2. Otherwise, when $\sigma_1, \ldots, \sigma_n$ are all *linear*, i.e., $\sigma_i(x), \sigma_i(y)$ are linear terms not sharing any variables when $x \neq y$, we reduce to case (1) (see Section 3.4.2).

3. Otherwise, if $\sigma_i$ is non-linear and $y{:}s$ occurs more than once either in $\sigma_i(x)$ or in $\sigma_i(x), \sigma_i(z)$, $x \neq z$, with $s$ finite, $T_{\Sigma \cup \Sigma^{\#}, s} = \{u_1, \ldots, u_k\}$, then we replace the problem $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ by the problem $\{t\} - \{t\sigma_1, \ldots, t\sigma_i\{y \mapsto u_1\}, \ldots, t\sigma_i\{y \mapsto u_k\}, \ldots, t\sigma_n\}$ and check again the new problem.

4. Outside cases (1)–(3) above, we invoke the order-sorted version of the algorithm in [7], which is more efficient than those in [9, 10, 11] and gives a full answer to difference problems $\{t\} - \{t_1, \ldots, t_n\}$, whereas those in [9, 10, 11] give a full answer to arbitrary Boolean combinations (see Section 3.4.2).

As shown below, Case (1) always succeeds for linear patterns, Case (2) reduces to Case (1), and Case (3) tries to reduce to Case (1). Only when a reduction to the linear pattern Case (1) is impossible, is Case (4) (the Lassez-Marriott algorithm) invoked. This latter algorithm, either succeeds for the given non-linear patterns, yielding a solution $\{u_1, \ldots, u_m\}$ to the pattern difference problem $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$, or fails in finite time, yielding an error message, which may include a partial answer.

In Case (1), where all terms $t, t\sigma_1, \ldots, t\sigma_n$ are linear, the following rewrite rules are applied:

1. $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\} \to (\{t\} - \{t\sigma_1\}) \cap \ldots \cap (\{t\} - \{t\sigma_n\})$

2. $\{t\} - \varnothing \to \{t\}$

3. $\{f(t_1, \ldots, t_n)\} - \{f(t_1\sigma, \ldots, t_n\sigma)\} \to$
   $\{f(t_1, \ldots, u, \ldots, t_n) \mid u \in (\{t_i\} - \{t_i\sigma\}), 1 \leqslant i \leqslant n\}$,
   where $\mathit{fvars}(u)$ are fresh variables.

4. $\{x{:}s\} - \{y{:}s'\} \to \{z_1{:}q_1, \ldots, z_k{:}q_k\}$,
   where $\{q_1, \ldots, q_k\} = \mathit{atoms}(s) - \mathit{atoms}(s')$

5. $\{x{:}s\} - \{f(t_1, \ldots, t_n)\} \to$
   $\{\overline{z{:}q}\} \cup \bigcup\{\{x_p{:}p\} - \{f(t_1, \ldots, t_n)\{\rho\} \mid \rho \in \mathit{Spec}(Y, S^{\#}),$
   $p = \mathit{ls}_{\Sigma^{\#}}(f(t_1, \ldots, t_n)\{\rho\})\} \mid p \in \mathit{atoms}(s) \cap \mathit{atoms}(f(t_1, \ldots, t_n))\}$
   *if* $s \notin S^{\#}$,
   where $Y = \mathit{fvars}(f(t_1, \ldots, t_n))$, $\overline{z{:}q} = z_1{:}q_1, \ldots, z_k{:}q_k$, $\{q_1, \ldots, q_k\} = \mathit{atoms}(s) - \mathit{atoms}(f(t_1, \ldots, t_n))$, and
   $\mathit{atoms}(f(t_1, \ldots, t_n)) = \{\mathit{ls}_{\Sigma^{\#}}(f(t_1, \ldots, t_n)\{\rho\}) \mid \rho \in \mathit{Spec}(Y, S^{\#})\}$.

6. $\{x{:}s\} - \{f(t_1, \ldots, t_n)\} \to$
   $\{u \mid u \in \mathit{Pat}(s) - \{f(\overline{x{:}s})\}\} \cup \{f(\overline{x{:}s})\} - \{f(t_1, \ldots, t_n)\}$
   *if* $s = \mathit{ls}_{\Sigma^{\#}}(f(t_1, \ldots, t_n)) \in S^{\#}$,
   where $\overline{x{:}s} = x_1{:}s_1, \ldots, x_n{:}s_n$, $s_i = \mathit{ls}_{\Sigma^{\#}}(t_i)$, and
   $\mathit{Pat}(s) = \{g(x_1{:}s_1, \ldots, x_n{:}s_n) \mid g : s_1 \ldots s_n \to s \in \Sigma^{\#}\}$.

### 3.4.2 Correctness of the Order-Sorted Pattern Difference Algorithm

The correctness of the order-sorted pattern difference algorithm can be stated as follows.

**Theorem 3.3** *(Correctness). The above order-sorted pattern difference algorithm terminates either with a finite set of patterns, or with an error message indicating an undefined or partial result; and without error for linear patterns. Furthermore, if the pattern difference operation $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ is defined for terms $t, t\sigma_1, \ldots, t\sigma_n$ and returns the set of patterns $\{u_1, \ldots, u_m\}$ as its result, then*

$$[\![t]\!] - [\![t\sigma_1, \ldots, t\sigma_n]\!] = [\![u_1, \ldots, u_m]\!]. \tag{3.7}$$

**Proof 3.5** *We reason by Cases (1)–(4) in the algorithm. For Case (1) (linear patterns), we can show the correctness of rules (1)–(6) by showing that each rule exp $\to$ exp′ is* language-preserving, *i.e., that $[\![exp]\!] = [\![exp']\!]$. For rule (1) this follows from the Boolean equation $A - (B \cup C) = (A - B) \cap (A - C)$. For rule (2) this is trivial. For rule (3) this follows from the following set equalities:*
$[\![f(t_1, \ldots, t_n)]\!] - [\![f(t_1\sigma, \ldots, t_n\sigma)]\!] =$
$\{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma]\} - \{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \ \wedge \ t_1\gamma \in [\![t_1\sigma]\!] \ \wedge \ \ldots \ \wedge \ t_n\gamma \in [\![t_n\sigma]\!]\} =$
$\{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \ \wedge \ \neg(t_1\gamma \in [\![t_1\sigma]\!] \ \wedge \ \ldots \ \wedge \ t_n\gamma \in [\![t_n\sigma]\!])\} =$
$\{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \ \wedge \ (t_1\gamma \notin [\![t_1\sigma]\!] \ \vee \ \ldots \ \vee \ t_n\gamma \notin [\![t_n\sigma]\!])\} =$
$\bigcup_{1 \leqslant i \leqslant n}\{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \ \wedge \ t_i\gamma \in [\![t_i - t_i\sigma]\!]\} =$
$\bigcup_{u_1 \in (\{t_1\} - \{t_1\sigma\})}[\![f(u_1, \ldots, t_n)]\!] \cup \ldots \cup \bigcup_{u_n \in (\{t_n\} - \{t_n\sigma\})}[\![f(t_1, \ldots, u_n)]\!]$.
*For rule (4) it follows easily from (2)–(3) in Theorem 3.1. For rule (5) this follows again from (2)–(3) in Theorem 3.1, the fact that if $\biguplus_p A_p$ is a disjoint union and if $B_p \subseteq A_p$, then $\biguplus_p A_p - \biguplus_p B_p = \biguplus_p (A_p - B_p)$, and also $[\![f(t_1, \ldots, t_n)]\!] = \biguplus_{\rho \in Spec(Y, S^\#)}[\![f(t_1, \ldots, t_n)\{\rho\}]\!]$. In this case, we know that $A_p = [\![x_p\!:\!p]\!]$ and furthermore know that $B_p = [\![\{f(t_1, \ldots, t_n)\{\rho\} \mid \rho \in Spec(Y, S^\#),\ p = ls_{\Sigma^\#}(f(t_1, \ldots, t_n)\{\rho\})\}]\!]$. The correctness of rule (6) follows for each $s \in S^\#$ from the language equality $[\![x{:}s]\!] = \biguplus_{v \in Pat(s)}[\![v]\!]$.*

*It is easy to prove that rules (1)–(6) terminate on any input of the form $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ with all terms linear, resulting in a combination of unions and intersections of finite sets of patterns which, by systematic application of distributivity of $\cap$ over $\cup$ and computation of symbolic $\cap$ operations, results in a finite set of $\Sigma \cup \Sigma^\#$-patterns denoting the same language as $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$.*

*Case (2), i.e., a problem $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ where some terms are non-linear but the $\sigma_i$ are linear can be reduced to an intersection of cases of the form $\{t\} - \{t\sigma\}$ where $\sigma = \{\overline{x{:}s} = \overline{v}\}$ is linear. Then we can apply the single rewrite rule $\{t\} - \{t\sigma\} \to \{t\{x_i{:}s_i \mapsto w\} \mid x_i{:}s_i \in \overline{x{:}s},\ w \in \{x_i{:}s_i\} - \{v_i\}\}$, which, since $\sigma$ is linear, reduces the problem to computing the differences $\{x_i{:}s_i\} - \{v_i\}$ between linear terms. The proof of correctness of this rule is entirely analogous to that for rule (3) of the linear case above and is left to the reader.*

*The correctness of the transformation in Case (3) reduces to the observation that if $\sigma(x)$ is non-linear but the sort $s$ of one the variables $y$ occurring more than once in $\sigma_i(x)$ is finite with $T_{\Sigma \cup \Sigma^\#, s} = \{u_1, \ldots, u_k\}$, then $[\![t\sigma]\!] = [\![t\sigma\{y \mapsto u_1\}]\!] \cup \ldots \cup [\![t\sigma\{y \mapsto u_k\}]\!]$.*

*In Case (4), the correctness of the Lassez-Marriott algorithm is proved in detail in [7] for the unsorted case and has a straightforward extension to the order-sorted case when the signature is $\Sigma \cup \Sigma^\#$. For references on the complexity analysis of this algorithm and a discussion of why it is considerably more efficient than similar algorithms in [9, 10, 11], we refer the reader to the detailed discussion by Pichler [10], where a more general —but computationally more costly— algorithm is given which can successfully compute pattern solutions whenever they exist for disjunctions of difference problems of the form $\{t_1\} - \{t_1\sigma_1^1, \ldots, t_1\sigma_{n_1}^1\} \cup$*

$\ldots \cup \{t_k\} - \{t_k\sigma_k^1, \ldots, t_k\sigma_{n_k}^k\}$. *This is a key part of more general, but also more costly, algorithms that, given any Boolean expression Bexp involving term patterns, can* decide *whether* $[\![Bexp]\!]$ *is a pattern language and in the affirmative case can compute its pattern representation [9, 10, 11]. Instead, our algorithm —which generalizes that of Lassez and Marriott to order-sorted patterns— can only decide if* $[\![t]\!] - [\![t_1, \ldots, t_n]\!]$ *is an order-sorted pattern language and in the affirmative case can construct its explicit representation (see Thm. 4.1 in [7]). Since in practice many application problems can be expressed in the form* $\{t\} - \{t_1, \ldots, t_n\}$*, we consciously trade off the extra generality of the algorithms in [9, 10, 11] for the considerably greater efficiency of the Lassez-Marriott one [7].*

A last practical issue is reducing the size of solutions of Boolean operations on finite sets of order-sorted patterns. That is, the resulting solution $\{t_1, \ldots, t_n\}$ may be bigger than necessary. This problem can be addressed by the application of two size-reducing and language-preserving rewrite rules, namely:

1. $\{t_1, \ldots, t_n\} \rightarrow \{t_2, \ldots, t_n\}$ *if* $t_2 \succcurlyeq t_1$, where we use associativity commutativity of set union to make the order of $t_1, t_2$ in the set immaterial, and where, by definition,[4] $t \succcurlyeq t'$ iff $t' = t\sigma$ for some substitution $\sigma$; that is, we can remove $t_1$ if it is a substitution *instance* of $t_2$, so that $[\![t_1]\!] \subseteq [\![t_2]\!]$.

2. $\{t_1, \ldots, t_n\} \rightarrow \{u, t_3, \ldots, t_n\}$ *if* $\{u\} = lgg_{\Sigma \cup \Sigma^\#}(t_1, t_2) \wedge \{u\} - \{t_1, t_2\} = \varnothing$, where we require that the order-sorted *least general generalization* of $t_1, t_2$, denoted $lgg_{\Sigma \cup \Sigma^\#}(t_1, t_2)$ [52], which could be a set of terms, is actually a singleton set, and is furthermore "tight," i.e., $[\![u]\!] \subseteq [\![t_1, t_2]\!]$ (which, since $\{u\} = lgg_{\Sigma \cup \Sigma^\#}(t_1, t_2)$, implies $[\![u]\!] = [\![t_1, t_2]\!]$).

For example, the result of the difference $\{x{:}B\} - \{b\} = \{f(z{:}B^\#), f(z'{:}A), a\}$ for the signature $\Sigma \cup \Sigma^\#$ in the right side of Figure 3.1 is reduced to $\{f(y{:}B), a\}$ by rule (2). There is of course a tradeoff between performance and succinctness of the computed answer. Furthermore, interest in optimizing the computed answer may depend on the application. More experimentation is needed to find a good balance. For example, an optimization based on rule (1) should be more efficient than one based on both rules (1)–(2).

### 3.4.3 Linear Pattern Languages as Regular Tree Languages

What advantages do we gain from this algorithm? Quite substantial ones to reason effectively about languages. Let $LT_\Sigma(X) \subseteq T_\Sigma(X)$ denote the set of *linear* terms in $T_\Sigma(X)$. Note that if $u \in LT_\Sigma(X)$ then $[\![u]\!]$ is a *regular* tree language. This follows from order-sorted signatures being tree automata, plus the regular expression fact that if $L_1, \ldots, L_n$ are regular languages, then $f(L_1, \ldots, L_n)$ is a regular language (more on this below). Also, $\mathcal{P}_{fin}(LT_{\Sigma \cup \Sigma^\#}(X))$ is closed under symbolic: (i) unions; (ii) intersections, because disjoint unifiers of linear terms are linear; and (iii) differences, since rules (1)–(6) preserve linearity of terms. Furthermore, given $\{t_1, \ldots, t_n\}, \{t'_1, \ldots, t'_m\} \in \mathcal{P}_{fin}(LT_{\Sigma \cup \Sigma^\#}(X))$ we can use pattern differences to *decide* whether $[\![\{t_1, \ldots, t_n\}]\!] = [\![\{t'_1, \ldots, t'_m\}]\!]$. Indeed, $[\![\{t_1, \ldots, t_n\}]\!] = [\![\{t'_1, \ldots, t'_m\}]\!] \Leftrightarrow \{t_1, \ldots, t_n\} \equiv \{t'_1, \ldots, t'_m\}$, where the relation $\equiv$ is defined by the equivalence: $\{t_1, \ldots, t_n\} \equiv \{t'_1, \ldots, t'_m\} \Leftrightarrow \{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\} = \varnothing \wedge \{t'_1, \ldots, t'_m\} - \{t_1, \ldots, t_n\} = \varnothing$. By the homomorphism theorem for Boolean algebras, this means that $[\![\_]\!]$ defines an *injective homomorphism of Boolean algebras*

$$[\![\_]\!] : \mathcal{P}_{fin}(LT_{\Sigma \cup \Sigma^\#}(X))/{\equiv} \rightarrow \mathcal{P}(T_\Sigma). \tag{3.8}$$

[4] We use the order in the same direction as in, e.g., [7], so that $\succcurlyeq$ is the "more or equally general than" relation. Other authors use $\preccurlyeq$ for the *same* relation.

This is as good as it gets, since $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))/{\equiv}$ is a *computable* Boolean algebra, where all operations become effective. This offers an attractive, simpler alternative to tree automata to effectively perform Boolean operations on linear pattern languages in a symbolic way.

The intimate connection between regular tree languages and the computable Boolean algebra of linear pattern languages $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))/\equiv$ can be further clarified: not only is $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))/\equiv$ a computable Boolean algebra structure, it is also a computable (unsorted) $\Sigma^u$-algebra structure, where $\Sigma^u_0 = \{a \mid \exists a : \epsilon \to s \in \Sigma\}$, and, for $n \geqslant 1$, $\Sigma^u_n = \{f \mid \exists f : s_1 \ldots s_n \to s \in \Sigma\}$. What an operation $f \in \Sigma^u_n$ in $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))/\equiv$ achieves in an effective, symbolic way is the regular language operation $f(L_1, \ldots, L_n) = \{f(t_1, \ldots, t_n) \mid t_i \in L_i,\ \ 1 \leqslant i \leqslant n\}$.

Of course, regular language operations are usually defined on $\mathcal{P}(T_{\Sigma^u})$ and they must now be *relativized* to $\mathcal{P}(T_\Sigma) \subseteq \mathcal{P}(T_{\Sigma^u})$ for the obvious reason that, being interested in order-sorted algebras $T_\Sigma$, we *do not care* about languages in $\mathcal{P}(T_{\Sigma^u}) - \mathcal{P}(T_\Sigma)$, since they are *meaningless* from the point of view of $T_\Sigma$. This relativization is easy to achieve. Just note that the function $\_ \cap T_\Sigma : \mathcal{P}(T_{\Sigma^u}) \ni L \mapsto (L \cap T_\Sigma) \in \mathcal{P}(T_\Sigma)$ is a *surjective homomorphism of Boolean algebras.* This is easy to see since, up to isomorphism, $\_ \cap T_\Sigma$ is just the Boolean algebra homomorphism $[j_{T_\Sigma} \to 1_2] : [T_{\Sigma^u} \to 2] \to [T_\Sigma \to 2]$ associated to the inclusion function $j_{T_\Sigma} : T_\Sigma \hookrightarrow T_{\Sigma^u}$. We have already defined for each $f \in \Sigma^u_n$ the operation $f_{\mathcal{P}(T_{\Sigma^u})} : \mathcal{P}(T_{\Sigma^u})^n \to \mathcal{P}(T_{\Sigma^u})$. Its relativization $f_{\mathcal{P}(T_\Sigma)} : \mathcal{P}(T_\Sigma)^n \to \mathcal{P}(T_\Sigma)$ is defined as the function $f_{\mathcal{P}(T_\Sigma)} : \mathcal{P}(T_\Sigma)^n \ni (L_1, \ldots, L_n) \mapsto (f_{\mathcal{P}(T_{\Sigma^u})}(L_1, \ldots, L_n) \cap T_\Sigma) \in \mathcal{P}(T_\Sigma)$. Note the useful fact that the Boolean homomorphism $\_ \cap T_\Sigma : \mathcal{P}(T_{\Sigma^u}) \to \mathcal{P}(T_\Sigma)$ is *also* a $\Sigma^u$-*homomorphism.* This is because for any $f \in \Sigma^u_n$ and $(L_1, \ldots, L_n) \in T^n_{\Sigma^u}$ we always have by monotonicity $(f_{\mathcal{P}(T_{\Sigma^u})}(L_1 \cap T_\Sigma, \ldots, L_n \cap T_\Sigma) \cap T_\Sigma) \subseteq (f_{\mathcal{P}(T_{\Sigma^u})}(L_1, \ldots, L_n) \cap T_\Sigma)$, and the equivalence $f(t_1, \ldots, t_n) \in T_\Sigma \Leftrightarrow f(t_1, \ldots, t_n) \in T_\Sigma \bigwedge_{1 \leqslant i \leqslant n} t_i \in T_\Sigma$ easily gives us the other containment $(f_{\mathcal{P}(T_{\Sigma^u})}(L_1, \ldots, L_n) \cap T_\Sigma) \subseteq (f_{\mathcal{P}(T_{\Sigma^u})}(L_1 \cap T_\Sigma, \ldots, L_n \cap T_\Sigma) \cap T_\Sigma)$.

To substantiate the claim that $f_{\mathcal{P}(T_\Sigma)}$ can be effectively defined in the algebra $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))$, and therefore in $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))/{\equiv}$, in a *symbolic* way we can use a signature transformation $\Sigma \mapsto \Sigma^\Box$, where $\Sigma^\Box = (S^\Box, \leqslant^\Box, \Sigma^\Box)$ extends $\Sigma = (S, \leqslant, \Sigma)$ as follows: $S^\Box = S \uplus \hat{S}$, $\leqslant^\Box$ is the smallest partial order containing $\leqslant$ and the inequalities $\{\top_{[s]} < [s] \mid [s] \in \hat{S}\}$, and $\Sigma^\Box$ adds to $\Sigma$ an operator $f : [s_1] \ldots [s_n] \to [s]$ for each subsort polymorphic family $f^{[s_1]\ldots[s_n]}_{[s]}$ in $\Sigma$. The symbolic description of $f_{\mathcal{P}(T_{\Sigma^\Box})}$ in $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma^\Box\cup\Sigma^{\Box\#}}(X))$ is both easy and easy to check correct: for $(T_1, \ldots, T_n) \in \mathcal{P}_{\mathit{fin}}(LT_{\Sigma^\Box\cup\Sigma^{\Box\#}}(X))^n$ we define:

$$f_{\mathcal{P}_{\mathit{fin}}(LT_{\Sigma^\Box\cup\Sigma^{\Box\#}}(X))}(T_1, \ldots, T_n) = \{f(t_1, \ldots, t_n) \in LT_{\Sigma^\Box\cup\Sigma^{\Box\#}}(X) \mid t_i \in T_i,\ 1 \leqslant i \leqslant n\} \tag{3.9}$$

But since we have $T_\Sigma \subseteq T_{\Sigma^\Box} \subseteq T_{\Sigma^u}$, the homomorphism $\_ \cap T_\Sigma$ factors as the composition $(\_ \cap T_{\Sigma^\Box});(\_ \cap T_\Sigma)$, where the second factor has an easy, effective symbolic description on linear patterns as the function

$$\mathcal{P}_{\mathit{fin}}(LT_{\Sigma^\Box\cup\Sigma^{\Box\#}}(X)) \ni T \mapsto (T \cap \{x_1{:}\top_{[s_1]}, \ldots, x_k{:}\top_{[s_k]}\}) \in \mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X)) \tag{3.10}$$

where $\hat{S} = \{[s_1], \ldots, [s_k]\}$. Thus, our desired, effective symbolic description of $f_{\mathcal{P}(T_\Sigma)}$ in $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))$ is the function:

$$\begin{gathered}\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X))^n \ni (T_1, \ldots, T_n) \mapsto \\ (f_{\mathcal{P}_{\mathit{fin}}(LT_{\Sigma^\Box\cup\Sigma^{\Box\#}}(X))}(T_1, \ldots, T_n) \cap \{x_1{:}\top_{[s_1]}, \ldots, x_k{:}\top_{[s_k]}\}) \in \mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^{\#}}(X)).\end{gathered} \tag{3.11}$$

## 3.5 MEMBERSHIP CONSTRAINTS AS EQUATIONAL FORMULAS

The symbolic Boolean and $\Sigma^u$-operations on linear pattern languages defined in Section 3.4.3 have as a direct consequence a useful *simplification* in the methods for deciding validity in an initial order-sorted algebra $T_\Sigma$ for formulas in a *richer language* than that of equational formulas $\mathit{Form}(\Sigma)$. The richer language in question, let us call it $\mathit{MCForm}(\Sigma)$, is Comon and Delor's language of *equational formulas with membership constraints* [18], for whose formulas they prove that validity in an initial order-sorted algebra $T_\Sigma$ is decidable.

Their language is untyped, but typing is achieved by a rich sublanguage of *sort expressions* used to define *unary membership predicates*. This allows them to deal with other regular tree languages besides those contained in $T_\Sigma$. Since from the perspective of our algebra of interest $T_\Sigma$ we do *not* care about such other languages, we give below a typed version[5] of their language of formulas $\mathit{MCForm}(\Sigma)$ for $\Sigma = (S, \leqslant, \Sigma)$ an order-sorted signature. It is just the extension of the first-order language $\mathit{Form}(\Sigma)$ of equational formulas obtained by adding a countable collection of unary membership predicates of the form $\_ \in \zeta$, where $\zeta$ is a *sort expression* in a set $\mathit{SExp}$ inductively defined by the grammar:

$$\top \mid \bot \mid s \mid \ \neg\zeta \mid \zeta \wedge \zeta \mid \zeta \vee \zeta \mid f(\zeta, \overset{n}{\ldots}, \zeta) \tag{3.12}$$

where $s \in S$ and $f \in \Sigma^u_n$, $n \in \mathbb{N}$. The new atoms are then of the form $t \in \zeta$ with $t \in T_\Sigma(X)$. We then close $\mathit{MCForm}(\Sigma)$ under all Boolean formula operations and under universal and existential quantification by variables with sorts in $S$.

The semantic meanings $[\![\zeta]\!]$ associated to the $\zeta \in \mathit{SExp}$ are precisely the *regular sublanguages* of $T_\Sigma$ generated by the corresponding Boolean operations, the operations (and constants) $f \in \Sigma^u$, and the constants $\top$ and $\bot$, from the regular languages $T_{\Sigma,s}$ associated to each $s \in S$. But note that, for $\Sigma$ finite and regular, all such languages are symbolically describable in an effective way as finite sets of patterns in $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^\#}(X))$. That is, there is an effective mapping

$$\pi : \mathit{SExp} \ni \zeta \mapsto \pi(\zeta) \in \mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^\#}(X)) \tag{3.13}$$

such that $[\![\zeta]\!] = [\![\pi(\zeta)]\!]$ and having the obvious inductive definition using the Boolean operations and those in $\Sigma^u$, with the base case $\pi(s) = \{x{:}s\}$ for each $s \in S$. The satisfaction of formulas $\varphi \in \mathit{MCForm}(\Sigma)$ in $T_\Sigma$ naturally extends that for the sublanguage $\mathit{Form}(\Sigma)$ by defining for each atom $t \in \zeta$ and ground substitution $\alpha$ of the variables of $t$, $T_\Sigma, \alpha \models t \in \zeta$ iff $t\alpha \in [\![\zeta]\!]$.

The simplification made possible by $\mathcal{P}_{\mathit{fin}}(LT_{\Sigma\cup\Sigma^\#}(X))$ to reduce formulas in $\mathit{MCForm}(\Sigma)$ to formulas in the simpler language $\mathit{Form}(\Sigma \cup \Sigma^\#)$ should now be obvious. Consider a membership atom $t \in \zeta$, and its set of patterns $\pi(\zeta)$. If $\pi(\zeta) = \varnothing$, then we have $T_\Sigma \models (t \in \zeta \Leftrightarrow t \neq t)$. Otherwise, $\pi(\zeta) = \{u_1, \ldots, u_k\}$, $k \geqslant 1$, where without loss of generality we may assume that $Y = \mathit{vars}(\pi(\zeta))$ is disjoint from $\mathit{vars}(t)$, and that $i \neq j \Rightarrow \mathit{vars}(u_i) \cap \mathit{vars}(u_j) = \varnothing$. But then we have $T_{\Sigma\cup\Sigma^\#} \models (t \in \zeta \Leftrightarrow (\exists Y)\ t = u_1 \vee \ldots \vee t = u_k)$. Call $\pi(t \in \zeta)$ the formula $t \neq t$, resp. $(\exists Y)\ t = u_1 \vee \ldots \vee t = u_k$, obtained from $t \in \zeta$ depending on whether $\pi(\zeta)$ is empty or not. This defines an obvious mapping

$$\mathit{MCForm}(\Sigma) \ni \varphi \mapsto \pi(\varphi) \in \mathit{Form}(\Sigma \cup \Sigma^\#) \tag{3.14}$$

just by replacing atoms of the form $t \in \zeta$ by their corresponding formulas $\pi(t \in \zeta)$ and such that we have $T_\Sigma \models \varphi \ \Leftrightarrow\ T_{\Sigma\cup\Sigma^\#} \models \pi(\varphi)$.

Note that $T_\Sigma, \alpha \models \neg(t \in \zeta) \ \Leftrightarrow T_\Sigma, \alpha \models t \in \neg\zeta$. This allows us to relate quantifier-free formulas in

[5] The untyped language in [18] then appears in our terms as the special case of the typed language $\mathit{MCForm}(\Sigma^\square)$, where $\Sigma^\square$ has a single connected component, say $[s]$, since then $T_{\Sigma^\square,[s]} = T_{\Sigma^u}$ and there is no need to explicitly type a variable as, say, $x{:}s$ because we can instead assert the membership predicate $x \in s$ for $x$ a variable of the universal sort $[s]$ and therefore *untyped*.

*MCForm*$(\Sigma)$ and *Form*$(\Sigma \cup \Sigma^{\#})$ in a simple way:

**Lemma 3.4** *Let $\varphi$ be a quantifier-free formula in MCForm$(\Sigma)$. We can effectively associate to $\varphi$ a quantifier free formula $\pi(\varphi)$ in Form$(\Sigma \cup \Sigma^{\#})$ such that $\varphi$ is satisfiable in $T_\Sigma$ iff $\pi(\varphi)$ is satisfiable in $T_{\Sigma\cup\Sigma^{\#}}$.*

**Proof 3.6** *Without loss of generality we may assume that $\varphi$ has the DNF form:*

$$\varphi \equiv \bigvee_{i\in I}(\bigwedge G^i \wedge \bigwedge D^i \bigwedge C^i) \tag{3.15}$$

*with $G^i$ a finite set of $\Sigma$-equations, $D^i$ a finite set of $\Sigma$-disequations, and $C^i$ a finite set of membership atoms of the form $t \in \zeta$. Then $\varphi$ is satisfiable in $T_\Sigma$ iff $T_\Sigma \models (\exists X)\ \varphi$, where $X = vars(\varphi)$. But $T_\Sigma \models (\exists X)\ \varphi$ iff $T_{\Sigma\cup\Sigma^{\#}} \models (\exists X)\ \varphi'$, where $\varphi' \equiv \bigvee_{i\in I'}(\bigwedge G^i \wedge \bigwedge D^i \bigwedge \pi(C^i))$, where $I' = \{i \in I \mid (\forall (t \in \zeta) \in C^i)\ \pi(\zeta) \neq \varnothing\}$, and where for each $(t \in \zeta) \in C^i$ with $i \in I'$ all the variables $Y = vars(\pi(\zeta))$ are disjoint from $X$, $j \neq j' \Rightarrow vars(u_j) \cap vars(u'_j) = \varnothing$, and also disjoint from the variables $Y' = vars(\pi(\zeta'))$ for any other $t' \in \zeta'$ occurring in $\varphi$. But since then the formulas $\pi(t_j \in \zeta_j) = (\exists Y_j)\ t_j = u^j_1 \vee \ldots \vee t_j = u^j_{k_j}$, $j \in J$, appearing in $\varphi'$ have disjoint bound variables $Y_j$ also disjoint from $X$, we have the equivalences:*

$$T_\Sigma \models (\exists X)\ \varphi \Leftrightarrow T_{\Sigma\cup\Sigma^{\#}} \models (\exists X)\ \varphi' \Leftrightarrow T_{\Sigma\cup\Sigma^{\#}} \models (\exists (X \uplus \biguplus_{j\in J} Y_j))\ \pi(\varphi) \tag{3.16}$$

*where $\pi(\varphi)$ is the quantifier-free formula obtained from $\varphi'$ by replacing each $\pi(t_j \in \zeta_j) = (\exists Y_j)\ t_j = u^j_1 \vee \ldots \vee t_j = u^j_{k_j}$ by the disjunction $t_j = u^j_1 \vee \ldots \vee t_j = u^j_{k_j}$. Therefore, $\varphi$ is satisfiable in $T_\Sigma$ iff $\pi(\varphi)$ is satisfiable in $T_{\Sigma\cup\Sigma^{\#}}$.*

This reduction of formulas in *MCForm*$(\Sigma)$ to formulas in *Form*$(\Sigma \cup \Sigma^{\#})$ offers several simpler alternatives for deciding the satisfiability of a formula $\varphi \in$ *MCForm*$(\Sigma)$ in an initial order-sorted algebra $T_\Sigma$. On the one hand, we can now use Theorem 3.2 in conjunction with the many-sorted decision procedures by Maher [15] or by Comon and Lescanne [16] to decide whether $T_\Sigma \models \varphi$. On the other hand, for quantifier free formulas $\varphi \in$ *MCForm*$(\Sigma)$, we can use Lemma 3.4 above in conjunction with the order-sorted decision procedure described in Theorem 8 of [36] to remain at the order-sorted level and decide $T_\Sigma \models \varphi$ in a more direct way. We illustrate this last possibility by means of a simple example.

**Example 3.1** *Consider the order-sorted signature $\Sigma_{NAT}$ with sorts Nat, Odd and Even, subsort inclusions Odd, Even $<$ Nat, constant $0 : \epsilon \to$ Even, and operators $s$ : Even $\to$ Odd, $s$ : Odd $\to$ Even, and $s$ : Nat $\to$ Nat. It is easy to check that the sort Nat is* redundant*, so that the only atomic sorts in $\Sigma^{\#}_{NAT}$ are Odd and Even, and we have $\Sigma_{NAT} = \Sigma_{NAT} \cup \Sigma^{\#}_{NAT}$. Consider now the problem of deciding whether $T_{\Sigma_{NAT}} \models \varphi$ for $\varphi$ the following formula in MCForm$(\Sigma_{NAT})$:*

$$\varphi \equiv (\forall x, y)\ (x = 0 \Rightarrow x \neq s(y)) \wedge x \in ((Odd \vee Even) \wedge \neg(Odd \wedge Even)) \tag{3.17}$$

*where the variables $x, y$ have sort Nat. $T_{\Sigma_{NAT}} \models \varphi$ iff the following quantifier-free formula is unsatisfiable in $T_{\Sigma_{NAT}}$:*

$$(x = 0 \wedge x = s(y)) \vee (x \in \neg((Odd \vee Even) \wedge \neg(Odd \wedge Even))) \tag{3.18}$$

*But since the system of equations $x = 0 \wedge x = s(y)$ has no order-sorted unifiers, by Theorem 8 of [36] the first conjunction is unsatisfiable. Consider now the set of patterns $\pi(\neg((Odd \vee Even) \wedge \neg(Odd \wedge Even)))$,* *that is,*

$$\{x{:}Nat\} - ((\{y{:}Odd, z{:}Even\}) \cap (\{x'{:}Nat\} - (\{y'{:}Odd\} \cap \{z'{:}Even\}))) \tag{3.19}$$

*Since the equation $y'\!:\!Odd = z'\!:\!Even$ has no order-sorted unifiers, we have $\{y'\!:\!Odd\} \cap \{z'\!:\!Even\} = \varnothing$. Furthermore, because of the subsort inclusions Odd, Even < Nat, the equations $x'\!:\!Nat = y\!:\!Odd$ and $x'\!:\!Nat = z\!:\!Even$ have respective unifiers $x'\!:\!Nat \mapsto y\!:\!Odd$ and $x'\!:\!Nat \mapsto z\!:\!Even$. So the above Boolean combination of sets of patterns simplifies to*

$$\{x{:}Nat\} - \{y{:}Odd, z{:}Even\} = \varnothing \tag{3.20}$$

*and this also can be automatically computed by our tool.*

*In summary, therefore, $\pi(x \in ((Odd \vee Even) \wedge \neg(Odd \wedge Even))) \equiv x \neq x$, which is also unsatisfiable in $T_{\Sigma_{NAT}}$. Therefore, $T_{\Sigma_{NAT}} \models \varphi$.*

## 3.6 IMPLEMENTATION AND EXPERIMENTS

The algorithms described in this chapter are highly *reflective*, a feature that nearly all of the algorithms in this thesis will share. They are *parametric* on signatures $\Sigma$ and perform meta-level operations on signatures and $\Sigma$-terms, such as order-sorted unification, matching, sort comparisons, and so on, to ultimately compute pattern operations. Fortunately, many of these auxiliary meta-level operations are available, or can be easily programmed, in the Maude language through its reflective features using its `META-LEVEL` module [6]. In `META-LEVEL`, a signature $\Sigma$ is meta-represented as a term $\overline{\Sigma}$ of sort `Module`, and a $\Sigma$-term $t$ is meta-represented as a so-called *meta-term* $\bar{t}$ of sort `Term`.

Since: (i) Maude syntax at the meta-level essentially mirrors the syntax at the object level; and (ii) inference rules such as above rules (1)–(6) can be directly expressed as rewrite rules operating on meta-terms, the *representational distance* between the theoretical description of the algorithm in Section 3.4 and its actual meta-level implementation in Maude is relatively short.

Following the above-mentioned reflective design we have developed in Maude a tool[6] that implements both the $\Sigma \mapsto \Sigma^{\#}$ transformation and the order-sorted pattern operations. The tool has a user interface through which the user can first enter order-sorted specifications and then give commands to perform pattern operations on such specifications.

The tool's implementation of the signature transformation $\Sigma \mapsto \Sigma^{\#}$ described in Section 3.2 essentially coincides with the telescoping procedure described therein. The procedure takes a reflected signature $\overline{\Sigma}$ as an argument and proceeds by non-deterministically selecting an operator $f$ in $\Sigma$ which has not been processed and whose strictly smaller typings have all been processed.

Using the signature transformation procedure the tool also implements the order-sorted pattern operation algorithms described in Section 3.4. The overall algorithm takes as arguments a reflected signature $\overline{\Sigma}$ and a symbolic Boolean expression composed of meta-terms $\bar{t}$ representing $\Sigma$-term patterns using a mixfix syntax where `_U_` represents union, `_&_` represents intersection, and `_-_` represents difference. A set of Boolean equations first reduces each Boolean symbolic pattern expression to a *normal form* (essentially pushing conjunctions/differences down the expression tree). A normal form is then solved using an algorithm that deals with each symbolic difference problem according to the steps described in Section 3.4: the problem is

[6] The tool and examples can be downloaded online at http://maude.cs.illinois.edu/w/index.php?title=Maude_Tools:Order-sorted_Term_Patterns.

first classified according to cases (1)–(4), iterating over the finite-sort transformation of case (3) if needed. Then, either the simpler algorithm for case (1) (essentially rules (1)–(6)), or its case (2) extension (see Section 3.4.2), or the more general order-sorted extension of the Lassez-Marriott algorithm [7] are invoked. Finally, symbolic union and intersection operations are performed to obtain either: (i) a finite set of patterns if the algorithm computed a pattern language, or (ii) a Boolean expression containing some symbolic differences that do not denote pattern languages otherwise.

A user interface has also been constructed in Maude which allows the user to directly enter pattern expressions and theories using the Full Maude [6] syntax, obviating the need to first convert to the slightly more complex meta-term syntax. The user interface is implemented as an extension of Full Maude using Maude's `LOOP-MODE` module [6]. The user interface provides to the user two primary commands: `solve-pattern` and `ms-solve-pattern` to solve pattern intersections and differences in an order-sorted (resp. many-sorted) way. After loading the tool, one can enter any input theories one wishes to reason about and give the desired pattern operation commands. Sections 3.5 and 3.7 contain examples illustrating the use of the tool for solving various pattern operation problems.

Note that, following the usual convention for Full Maude [6], both theory declarations and commands are enclosed in parentheses `()` and that commands are ended with a period before the closing parenthesis `(.)`. Also note that, thanks to Maude's mixfix syntax capabilities, pattern syntax at the tool interface level is almost identical to that used internally by the library. The syntax for variables is the usual `name:sort` notation, so that `X:B` is a variable named `X` which has sort `B`. The `ms-solve-pattern` command (not shown above), first reduces the pattern to all of its many-sorted instances and solves each of them using the many-sorted pattern algorithm.

### 3.6.1 Experimental Comparison of the Many- and Order-Sorted Difference Algorithms

We conducted experiments comparing our order-sorted pattern operations algorithm to its many-sorted reduction. To ground our discussion, we work in a module `COMPLEX-RAT` adapted from [33] that defines the signature of complex numbers. As further explained below, our experiments show that, on average, the many-sorted reduction requires about 1,000 times as many rewrites as the order-sorted algorithm, with the median being 55 times as many rewrites.

The `COMPLEX-RAT` example is a moderate size one, yet a significant illustration of what we take it to be the main lesson we can draw from these experiments: if the sort hierarchies are almost flat, there is no reason to expect any substantial advantages in using the order-sorted algorithm besides the obvious greater simplicity and "user friendliness" gained from expressing both the pattern problems and their solutions in a way as close as possible to the original language of the user. However, as sort hierarchies grow bigger —which is a very common case in large Maude specifications because they heavily use many instances of parameterized data types introducing many subsort inclusions— the very substantial advantages in both performance, simplicity of the questions and answers, and user friendliness already apparent in the `COMPLEX-RAT` example should become even more dramatic.

In general, a complete experimental comparison is impossible, since there are infinitely many signatures, each typically generating an infinite set of term patterns up to renaming. Thus, our goal with these experiments is not any kind of "proof," but rather to provide evidence that in non-toy examples our order-sorted algorithm is both more expressive and more performant.

For greater generality in comparing these two algorithms, we might have randomly generated signatures

```
fmod NAT-SIG is
  sorts Nat NzNat Zero .
  subsorts Zero NzNat < Nat .
  op 0 : -> Zero .
  op s_ : Nat -> NzNat .
  op _+_ : Nat Nat -> Nat .
  op _+_ : NzNat Nat -> NzNat .
  op _+_ : Nat NzNat -> NzNat .
  op _*_ : Nat Nat -> Nat .
  op _*_ : NzNat NzNat -> NzNat .
endfm
```

```
fmod INT-SIG is
  protecting NAT-SIG .
  sorts Int NzInt .
  subsort Nat < Int .
  subsorts NzNat < NzInt < Int .
  op -_ : Int -> Int .
  op -_ : NzInt -> NzInt .
  op _+_ : Int Int -> Int .
  op _*_ : Int Int -> Int .
  op _*_ : NzInt NzInt -> NzInt .
endfm
```

```
fmod RAT-SIG is
  protecting INT-SIG .
  sorts Rat NzRat .
  subsort Int < Rat .
  subsorts NzInt < NzRat < Rat .
  op _/_ : Rat NzRat -> Rat .
  op _/_ : NzRat NzRat -> NzRat .
  op -_ : Rat -> Rat .
  op -_ : NzRat -> NzRat .
  op _+_ : Rat Rat -> Rat .
  op _*_ : Rat Rat -> Rat .
  op _*_ : NzRat NzRat -> NzRat .
endfm
```

```
fmod COMPLEX-RAT is
  protecting RAT-SIG .
  sorts Cpx Imag NzImag NzCpx .
  subsort Rat < Cpx .
  subsort NzRat NzImag < NzCpx .
  subsorts NzCpx < Imag < Cpx .
  subsorts Zero < Imag .
  op _i : Rat -> Imag .
  op _i : NzRat -> NzImag .
  op -_ : Cpx -> Cpx .
  op -_ : NzCpx -> NzCpx .
  op _+_ : Cpx Cpx -> Cpx .
  op _+_ : NzRat NzImag -> NzCpx .
  op _*_ : Cpx Cpx -> Cpx .
  op _*_ : NzCpx NzCpx -> NzCpx .
  op _/_ : Cpx NzCpx -> Cpx .
  op _# : Cpx -> Cpx .
  op |_|^2 : Cpx -> Rat .
  op |_|^2 : NzCpx -> NzRat .
endfm
```

Figure 3.5: COMPLEX-RAT Signature

from the space of all order-sorted signatures and further randomly generated terms in each signature. While this might seem more convincing, the majority of random signatures would never be used for any practical purpose. Thus, in our experiments we have favored practicality of examples over generality and have focused on the above signature `COMPLEX-RAT` which is part of an algebraic specification of the complex rational numbers. This signature —adapted from [33]— or very similar ones have been used in actual programming and verification tasks. Furthermore, it displays a substantial degree of subsorting, which will let us usefully compare the two algorithms (since they are essentially the same in the many-sorted case).

After fixing a signature, there is still the matter of generating terms to be inputs to the difference algorithms. In `COMPLEX-RAT`, we generated a set $T$ of random terms via the following process: Given a desired term depth $n >= 0$, let a counter $i$ be set to 0. While $i < n$, randomly select a non-constant operator for each open node in the term tree (the empty tree is open by default) and increment $i$. For each open node where $i = n$, randomly select a variable with 70% probability or a constant operator with 30% probability. For the experiments in this section, we set $0 \leqslant n \leqslant 2$.

Given a set of terms $T$ randomly generated as above, we computed all pairs $(t_1, t_2) \in T^2$ and further generated the pattern operations $[\![t_1]\!] - [\![t_2]\!]$. Finally, these pattern operations can be input into our order-sorted algorithm as well as their many-sorted transformed versions. The metric used for comparison is the number of rewrites as counted by the Maude rewrite engine used to implement these two algorithms. This metric is useful because it is invariant across different implementations and lets us abstract away from implementation differences.

The many-sorted transformed version of a problem was obtained by applying the sharpening transformation of Section 3.3 to the signature `COMPLEX-RAT` and to the pattern operation to be solved. Since many-sorted signatures are just a special case of order-sorted signatures, we can plug the transformed inputs into our order-sorted algorithm and it will compute results in a strictly many-sorted way. In general, since the transformation of an order-sorted difference problem into a many-sorted one described in Section 3.4 causes each order-sorted term to be expanded into a set of many-sorted ones, we can expect that the order-sorted difference algorithm will perform better, especially in cases with deep subsorting and multiple variables. Another side effect of this transformation is that answer sizes will likely be larger, since each order-sorted term may expand into several many-sorted ones.

In all, we computed roughly a thousand term difference problems using both our order-sorted algorithm and the many-sorted transformed problem and algorithm. On the whole, in the signature `COMPLEX-RAT`, with a thousand difference problems containing terms ranging from size zero to two, the many-sorted algorithm on average needed about 1,000 times as many rewrites as our order-sorted algorithm, with the median being 55 times as many rewrites.

Each dot in Figure 3.6 represents a randomly chosen difference problem among the (roughly) thousand ones we generated. The dot's x-coordinate gives the number of rewrites needed to solve the problem using the order-sorted algorithm, while its y-coordinate gives the corresponding number of rewrites when the problem is reduced to a many-sorted one and the many-sorted algorithm is used. Note the difference in scale: the x-axis continues until 70,000 rewrites while the y-axis goes all the way to 800,000 rewrites.

Actually, the full graph of the data would scale the axes even further, so for formatting purposes some of points were removed and included in the table in Figure 3.7. If included in Figure 3.6, the difference in scale would become so great that the graph would become almost useless as a visual representation of the data. To better understand what is happening with these outlier points, we consider two examples: the outlier with the maximum x-value and the corresponding one with the maximum y-value.

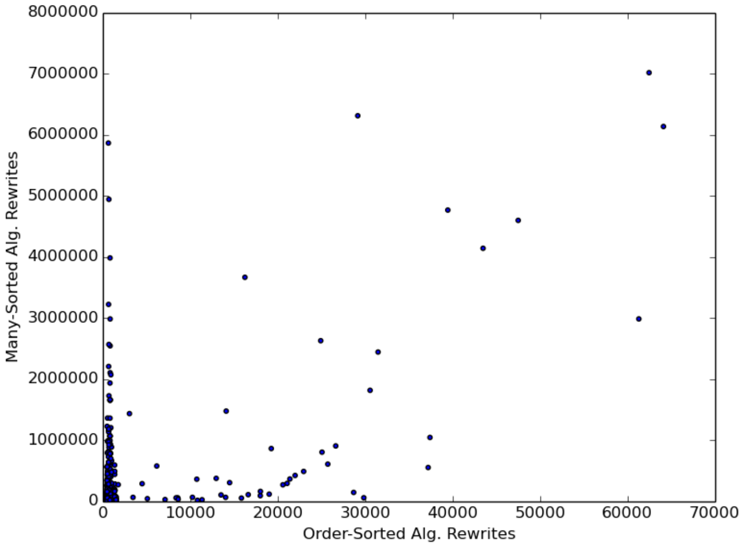


Figure 3.6: Rewrites Used by Order-Sorted vs. Many-Sorted Algorithm

| OS Rewrites | MS Rewrites |
|---|---|
| 23103 | 35811474 |
| 24937 | 49010016 |
| 53823 | 1233545932 |
| 69359 | 12619028 |
| 69765 | 14073907001 |
| 76389 | 21151931 |
| 78891 | 23954195214 |
| 86061 | 29805843801 |
| 91952 | 155444847 |
| 98377 | 5105671052 |
| 125465 | 12175065031 |
| 152156 | 152431926 |
| 192685 | 479356113 |
| 233652 | 713057516 |
| 340468 | 15062845092 |
| 664733 | 2618219 |

Figure 3.7: Experimental Data Outliers

The maximum x-value reported in our data set is 664733, with 2618219 the corresponding y-value and 2618219/664733 ≈ 4 the smallest y/x ratio in Figure 3.7. The operation in question is {-C:Cpx} − {-(R:Rat*0)}. Here, because of the deep sorting and unifiability of the terms, the order-sorted algorithm must consider many possible operators that C could instantiate into that do not unify with R:Rat*0. On the other hand, since there are only two variables, the many-sorted expansion does not generate as many patterns as when the problem contains many variables.

Alternatively, we may consider the maximum y-value: 29805843801, with 86061 its corresponding x-value and 29805843801/86061 ≈ 346334 the largest y/x ratio in Figure 3.7. In this case, the pattern operation is given by:

$$\begin{array}{l}\{\texttt{R1:Rat*R2:Rat}\}\ -\\ \quad\{\texttt{((N1:Nat+N2:Nat)*(N3:Nat+N4:Nat))*((N5:Nat+N6:Nat)+s(0))}\}\end{array}\tag{3.21}$$

As predicted above, this particular operation with eight variables expands into a huge number of cases—1,323 separate cases in fact. Furthermore, each case may require several levels of descent to verify how the terms overlap. In summary, the data collected from our experiments show that for real signatures of interest even small input terms may create a huge difference in performance when choosing between the many-sorted and order-sorted algorithms.

## 3.7 APPLICATIONS AND EXAMPLES

In Sections 3.4.3 and 3.5 we have already discussed applications to, respectively, regular tree language operations, and to reducing the problem of deciding the satisfiability of a formula with memberships constraints in an initial order-sorted algebra to the simpler problem of deciding such satisfiability for a simpler equational formula without such constraints. Here, to give a flavor for the wide range of applications possible, we consider three different ones: (i) sufficient completeness checking for equational order-sorted programs; (ii) elimination of the otherwise feature (used in several languages like Haskell, ASF+SDF, and Maude) for such programs; and (iii) invariant verification.

### 3.7.1 Checking Sufficient Completeness

The pattern operation algorithms presented in this chapter can be used to analyze the sufficient completeness of an equational program in order-sorted equational logic, provided its equations satisfy some conditions, further explained below, that allow an analysis based on term differences. In particular, the conditions in question are always satisfied when the equations are *left-linear*, i.e., their lefthand sides do not have repeated variables.

We illustrate the ideas with a simple example of addition and multiplication in the natural numbers. We first explain the notion of sufficient completeness and how term differences can settle sufficient completeness problems. We assume acquaintance with the following notions: (i) order-sorted rewrite rule $l \to r$, and rewrite relation $\to_R$ associated to a set $R$ of such rules; and (ii) termination of the $\to_R$ relation. For more details about these notions see, e.g., [6].

The sufficient completeness problem goes back to Guttag's thesis [53]. It arose as the question of whether in an equational program $(\Sigma, E)$ defining several recursive functions each such function has been *fully defined* by the equations $E$, where for execution purposes each equation $t = t'$ in $E$ is oriented from left to right as a rewrite rule $t \to t'$. More generally, this problem can be posed for a *terminating* rewrite-rule-based program $(\Sigma, R)$ as follows: we split the order-sorted signature $\Sigma$ as a disjoint union $\Omega \uplus \Delta$, where $\Delta$ are the so-called *defined symbols*, and $\Omega$ the (claimed to be) *constructor symbols* for the rules $R$. Whether or not the operators in $\Omega$ can rightfully be called *constructor symbols* for $R$ is precisely the sufficient completeness problem, so that both notions (constructors and sufficient completeness) stand or fall together. To simplify the exposition we assume that each subsort-polymorphic family of function symbols $f^{[s_1]\ldots[s_n]}_{[s]}$ in $\Sigma$ is fully included in either $\Omega$ or $\Delta$.

**Definition 3.1** *Under the above assumptions on* $(\Sigma, R)$ *and the decomposition* $\Sigma = \Omega \uplus \Delta$*, let* $Irr_R \subset T_\Sigma$ *denote the subset of those ground* $\Sigma$*-terms that are* irreducible *by the rules* $R$*, i.e.,* $t \in Irr_R$ *iff* $(\nexists)\ u$ *s.t.* $t \to_R u$*. Then we call theory* $(\Sigma, R)$ sufficiently complete *with respect to* $\Omega$ *iff* $Irr_R \subseteq T_\Omega$*. Likewise, we then call* $\Omega$ *a* constructor subsignature *for the rules* $R$*.* $\Omega$ *is called a subsignature of* free constructors *for* $R$ *iff* $Irr_R = T_\Omega$*.*

Since $(\Sigma, R)$ is assumed terminating, this means that each ground $\Sigma$-term $t$ *always evaluates to a constructor term*, capturing the idea that the symbols in $\Delta$ have been *fully defined* by the rules $R$. Note that the expression “free constructor subsignature” is well chosen, since in this case no term in $T_\Omega$ can be rewritten, so that $T_\Omega$ is an initial algebra. For example, the operators 0 and $s$ define a free constructor subsignature for the natural number addition rules $n + 0 \to n$ and $n + m \to s(n+m)$, but, while remaining constructors,[7] they are no longer free if we add the equation $s(s(n)) = n$, since then $Irr_R = \{0, s(0)\}$.

Sufficient completeness can be boiled down to the following equivalent property, expressed in the lemma below, whose easy proof is left to the reader:

**Lemma 3.5** *Let* $(\Sigma, R)$ *be a terminating rewrite rule program with* $\Sigma = \Omega \uplus \Delta$ *a decomposition into constructors and defined symbols. Then* $(\Sigma, R)$ *is sufficiently complete with respect to* $\Omega$ *iff for each* $f : s_1 \ldots s_n \to s$ *in* $\Delta$ *and each* $u_i \in T_{\Omega, s_i} \cap Irr_R$*,* $1 \leqslant i \leqslant n$*,* $f(u_1 \ldots u_n) \notin Irr_R$*.*

To cast the sufficient completeness problem as a symbolic term difference problem we use Lemma 3.5 above and define a simple signature transformation $\Sigma \mapsto \Sigma_\Delta$ as follows: the poset of sorts of $\Sigma_\Delta$ extends the poset $(S, \leqslant)$ of $\Sigma$ by adding to each connected component $[s]$ of $\Sigma$ a new sort $d_{[s]}$ with $d_{[s]} > \top_{[s]}$. The operators of $\Sigma_\Delta$ are those of $\Omega$ plus for each $f^{[s_1]\ldots[s_n]}_{[s]} \subseteq \Delta$ an operator $f : d_{[s_1]} \ldots d_{[s_n]} \to d_{[s]}$. Note that for each $s \in S$ we then have $T_{\Sigma_\Delta, s} = T_{\Omega, s}$. What this achieves, is that for any $f : s_1 \ldots s_n \to s$ in $\Delta$ the ground instances of the $\Sigma_\Delta$-term $f(x_1{:}s_1, \ldots, x_n{:}s_n)$ always instantiate the variables $x_1{:}s_1, \ldots, x_n{:}s_n$ by constructor terms. To simplify the exposition let us assume —this assumption can be omitted but a somewhat more complex formulation is then needed— that for each defined symbol $f$ there is an $f : s_1 \ldots s_n \to s$ with

[7] Note that the issue of whether $\Omega$ is a constructor subsignature is *independent* of whether the rules $R$ have been defined following the so-called *constructor discipline*, where $\Delta$ is defined as those $f$ such that there is some rule in $R$ of the form $f(u_1, \ldots, u_n) \to v$, $\Omega$ is defined as $\Sigma - \Delta$, and for each rule $f(u_1, \ldots, u_n) \to v$ in $R$ the terms $u_1, \ldots, u_n$ are required to be $\Omega$-terms. The independence can be shown by the following three observations: (i) if $R$ consists of the single rule $n + m \to s(n+m)$ with $\Sigma = \{0, s, +\}$ unsorted, then $R$ follows the constructor discipline with $\Omega = \{0, s\}$, but $\Omega$ is *not* a signature of constructors for $R$ in *our* sense, since sufficient completeness fails; (ii) the rules $R = \{n + 0 \to n, n + m \to s(n+m), (n+m) + k \to n + (m+k)\}$ do *not* follow the constructor discipline, yet $\Omega = \{0, s\}$ is a (free) constructor signature for $R$ in our sense; and (iii) for $R = \{n + 0 \to n, n + m \to s(n+m), s(s(n)) = n\}$ the constructor discipline totally breaks down, because according to it we would have $\Delta = \{+, s\}$ and $\Omega = \{0\}$, and *all* rules would violate such discipline. But choosing $\Omega = \{0\}$ as a subsignature of constructors is nonsense: the correct signature of (non-free) constructors (in our sense) is $\Omega = \{s, 0\}$, which of course violates the constructor discipline but *is* sufficiently complete. What can rightfully be said is that: (a) if $R$ is defined following the constructor discipline, which defines a decomposition $\Sigma = \Omega \uplus \Delta$, and (b) the rules $R$ are sufficiently complete (in our sense) with respect to *that* $\Omega$, then (c) $\Omega$ is a subsignature of *free* constructors for $R$.

```
fmod NATS is
  sorts Nat NzNat Zero .
  subsorts Zero NzNat < Nat .
  op 0 : -> Zero [ctor] .
  op s_ : Nat -> NzNat [ctor] .
  op _+_ : Nat Nat -> Nat .
  op _*_ : Nat Nat -> Nat .
  vars N M : Nat .
  eq N + 0 = N .
  eq (s N) + (s M) = s s (N + M) .
  eq N * 0 = 0 .
  eq (s N) * (s M) = s (N + (M + (N * M))) .
endfm
```

Figure 3.8: Maude NATS Specification

$s_1 \ldots s_n s$ biggest possible among the typings of operators in $f_{[s]}^{[s_1] \ldots [s_n]} \subseteq \Delta$. Define $R_f = \{l \to r \in R \mid l = f(u_1 \ldots u_n) \;\; s.t. \;\; u_i \in T_\Omega(X),\ 1 \leqslant i \leqslant n\}$.

Then, it is immediate that if we can show that the symbolic difference of $\Sigma_\Delta$-patterns $\{f(x_1{:}s_1, \ldots, x_n{:}s_n)\} - \{l \mid l \to r \in R_f\}$ equals $\emptyset$, then for each $u_i \in T_{\Omega,s_i} \cap \mathit{Irr}_R$, $1 \leqslant i \leqslant n$, $f(u_1 \ldots u_n) \notin \mathit{Irr}_R$ and therefore, by Lemma 3.5, assuming $R$ terminating, $(\Sigma, R)$ is sufficiently complete with respect to $\Omega$. Note that if $\mathit{Irr}_R = T_\Omega$, the emptiness of this symbolic difference is a necessary and sufficient condition for the sufficient completeness of a terminating $(\Sigma, R)$.

Consider the Maude specification of the natural numbers in Figure 3.8. In a Maude specification each operator is preceded by the keyword `op`, and each operator which is a constructor is declared with the `ctor` attribute. In all, there are two defined operators and two constructors. For a further review of of Maude syntax, see Section 2.3.

To achieve the $\Sigma \mapsto \Sigma_\Delta$ transformation, in the above signature `NATS`, we would add a new top sort `dNat` and change the type of defining operators, giving the signature:

```
fmod NATS-DELTA is
  sorts dNat Nat NzNat Zero .
  subsorts Zero NzNat < Nat < dNat .
  op 0 : -> Zero [ctor] .
  op s_ : Nat -> NzNat [ctor] .
  op _+_ : dNat dNat -> dNat .
  op _*_ : dNat dNat -> dNat .
  vars N M : Nat .
  eq N + 0 = N .
  eq (s N) + (s M) = s s (N + M) .
  eq N * 0 = 0 .
  eq (s N) * (s M) = s (N + (M + (N * M))) .
endfm
```

Figure 3.9: Maude $\texttt{NATS}_\Delta$ Specification

To check that `_+_` is sufficiently complete we perform the query:

$$\{\texttt{N + M}\} - (\{\texttt{N + 0}\} \cup \{\mathrm{s\ N + s\ M}\}) \tag{3.22}$$

in the transformed signature $\Sigma_\Delta$ (of course itself internally transformed into $\Sigma_\Delta \cup \Sigma_\Delta^\#$), which yields the solution set:

$$\{\texttt{0 + s K}\} \tag{3.23}$$

This pattern represents an infinite set of constructor instances of the `_+_` operator for which `_+_` is not defined. Indeed, if we check the function definition on lines 9-10 of Figure 3.9, we have covered the cases where the second input is a zero and where both inputs are non-zero, but not the case where the first input is zero and second non-zero! Using the above information, adding the equation solves the issue:

$$\texttt{eq 0 + s N = s N .} \tag{3.24}$$

We can now compute in the transformed signature the difference: $\{\texttt{N + M}\} - (\{\texttt{N + 0}\} \cup \{\texttt{s N + s M}\} \cup \{\texttt{0 + s N}\})$ which is indeed empty.

In the same way, we can check the sufficient completeness of `_*_` by performing in $\Sigma_\Delta$ the symbolic difference:

$$\{\texttt{N * M}\} - (\{\texttt{N * 0}\} \cup \{\texttt{s N * s M}\}) \tag{3.25}$$

which yields the result set:

$$\{\texttt{0 * s K}\} \tag{3.26}$$

Reviewing our multiplication definition on lines 11-12 of Figure 3.9, we see that we have made the same mistake we did before when defining addition. Again, a single new equation completes the definition:

$$\texttt{eq 0 * s N = 0 .} \tag{3.27}$$

Happily, the symbolic difference below is also now empty:

$$\{\texttt{N * M}\} - (\{\texttt{N * 0}\} \cup \{\texttt{s N * s M}\} \cup \{\texttt{0 * s N}\}) \tag{3.28}$$

Let `NATS-FIXED` be the modified module which extends `NATS` by adding these additional equations. The above check means that we have now reduced the sufficient completeness of `NATS-FIXED` to proving that its equations, oriented as rewrite rules, are terminating.

Two final observations seem worth making:

1. Note that if we were to add to `NATS-FIXED` an equation that does not follow the constructor discipline, such as `(N + M)+ K = N +(M + K)` with `K` also of sort `Nat`, then, provided we show that termination of the equations still holds, sufficient completeness would still hold. This is because: (i) the above equation does *not* belong to the set $R_+$ of rules defining $+$ whose lefthand sides' immediate subterms are all constructor terms; and (ii) only lefthand sides of rules in $R_+$ need to be used in the sufficient completeness check.

2. Note that if, instead, we were to add to `NATS-FIXED` the equation `s s N = N`, provided that we show that termination of the equations still holds, sufficient completeness would still hold. This is because

```
fmod CL-SYNTAX is
  sorts Constant Name CL .
  subsorts Name Constant < CL .
  ops S K I : -> Constant [ctor] .
  op __ : CL CL -> CL [ctor gather (E e)] .
  op x : -> Name [ctor] .
  op ,_ : Name -> Name [ctor] .
endfm

fmod CL is including CL-SYNTAX .
  vars U V W : CL .
  eq S U V W = (U W) (V W) .
  eq K U V = U .
  eq I U = U .
endfm
```

Figure 3.10: Maude CL Theory

the equation `s s N = N` only involves constructors, and therefore does not belong to the sets $R_+$ or $R_*$, whose lefthand sides are the only ones used in the sufficient completeness check.

### 3.7.2 Eliminating the `otherwise` Feature

Various rule-based functional languages, such as Haskell [48], ASF+SDF [49], and Maude [6], offer the convenient feature of specifying the result of a function when none of the patterns in the lefthand sides of the rewrite rules defining such a function can be applied to the subterm being evaluated. In such a case, the programmer can specify how the function should be evaluated using the `otherwise` feature. This is very convenient *and* efficient for programming purposes, but it has a main drawback: the `otherwise` feature is really an *extra-logical* feature. This makes formal reasoning about programs using it quite challenging: unless an *equivalent* program not using the feature can replace the original program, formal tools typically cannot reason about such programs. To the best of our knowledge this problem has remained unsolved so far. This section is significant not only for illustrative purposes, but also because it provides a solution to this long-standing problem when the equational programs defining a function $f$ using the `otherwise` construct are left-linear, or are at least such that the pattern difference $\{f(x_1,\ldots,x_n)\} - \{f(u_1^i,\ldots,u_n^i)\}_{i \in I}$, with the $f(u_1^i,\ldots,u_n^i)$, $i \in I$, the lefthand sides defining $f$ and not using the `otherwise` feature, can be computed (is defined) using the pattern difference algorithm.

The use and convenience of the `otherwise` feature can be best illustrated with an example. Consider the following Maude specification of combinatory logic, where we specify its syntax in the module `CL-SYNTAX` and then its semantics in the module `CL`. Note that an infinite supply of names: `x ,x ,,x` and so on is provided by the subsort `Name`. The left associative parsing of the application operator is specified in Maude with the `gather (E e)` attribute.

We may wish to do some formal reasoning about combinatory logic requiring, in particular, characterizing the set of CL terms in normal form. This can be done by means of a predicate `has-redex`, so that $t$ is in normal form iff `has-redex`$(t)$ = `false`. Defining the positive case of `has-redex` is easy, but the negative case, when rules do not apply, is harder. Here the `otherwise` feature shines:

```
fmod CL-REDEX is protecting CL-SYNTAX .
  protecting BOOL-OPS .
  op has-redex : CL -> Bool .

  var N : Name . var C : Constant . vars U V W : CL .

  eq has-redex(N) = false .
  eq has-redex(C) = false .
  eq has-redex(S U V W) = true .
  eq has-redex(K U V) = true .
  eq has-redex(I U) = true .
  eq has-redex(U V) = has-redex(U) or has-redex(V) [owise] .
endfm
```

Figure 3.11: Maude CL `has-redex` Predicate

This is great. But remember that we wanted to use `has-redex` to do some formal reasoning about CL, so it becomes crucial to know that the definition of `has-redex` is correct. At the very least we would, for example, like to know that the specification `CL-REDEX` is confluent, terminating, and sufficiently complete. But `CL-REDEX` is *not* an equational theory, so such questions cannot even be posed. In what follows we show how this impasse can be overcome.

Consider the following general method to define a function $f$ using the `otherwise` feature. For simplicity assume that the subsort-polymorphic family of function symbols $f_{[s]}^{[s_1]\dots[s_n]}$ in the order-sorted signature $\Sigma$ is fully included in the subsignature $\Delta$ of defined function symbols, and that there is a biggest possible typing $f : s_1 \dots s_n \to s$ for $f$ (these two assumptions are not essential, but they will simplify the exposition). The function definition we then assume for $f$ decomposes into two sets of equations: $E_f^s = \{f(u_1^i, \dots, u_n^i) = r_i\}_{i \in I}$, the *standard* equations for $f$, where we assume that all terms $u_k^i$ are *constructor* terms in the constructor subsignature $\Omega$, and $E_f^o = \{f(v_1^j, \dots, v_n^j) = q_j \ \ \texttt{[owise]}\}_{j \in J}$, the *otherwise* equations, where we also assume that all $v_k^j$ are constructor terms. That is, we assume that $f$ has been defined following the *constructor discipline* explained in Footnote 7, but using also the `otherwise` feature.

Recall now the signature transformation $\Sigma \mapsto \Sigma_\Delta$ used for sufficient completeness checking. In $\Sigma_\Delta$ $f$ has the typing $f : d_{[s_1]} \dots d_{[s_n]} \to d_{[s]}$, and all terms in the original sorts of $\Sigma$ are now constructor terms. The semantics of the `[owise]` feature can now be made precise: the equations in $E_f^o$ should only be applied to terms in the set of $\Sigma_\Delta$-terms that are instances of the set of patterns difference:

$$\{f(x_1{:}s_1, \dots, x_n{:}s_n)\} - \{f(u_1^i, \dots, u_n^i)\}_{i \in I} \tag{3.29}$$

Let $\{f(w_1^l, \dots, w_n^l)\}_{l \in L}$ be such a set difference. Then an equation $f(v_1^j, \dots, v_n^j) = q_j \ \ \texttt{[owise]}$ will apply to a term iff such a term is an instance of such a set of patterns and also of $f(v_1^j, \dots, v_n^j)$. But this exactly means that we can obtain a semantically *equivalent* version of our program by keeping $E_f^s$ untouched, and replacing $E_f^o = \{f(v_1^j, \dots, v_n^j) = q_j \ \ \texttt{[owise]}\}_{j \in J}$ by the set of *standard* equations:

$$E_f^{o.s} = \bigcup_{j \in J} \{f(v_1^j, \dots, v_n^j)\alpha = q_j\alpha \mid \alpha \in \bigcup_{l \in L} DUnif_{\Sigma_\Delta}(f(v_1^j, \dots, v_n^j), f(w_1^l, \dots, w_n^l))\} \tag{3.30}$$

Let us apply this method to our `CL-REDEX` example. We first construct the pattern problem corresponding

```
fmod CL-REDEX-EQ is
  pr CL-SYNTAX .
  pr BOOL-OPS .
  op has-redex : CL -> Bool .

  vars N : Name .  var C : Constant .
  vars U V W X Y : CL .

  eq has-redex(N) = false .
  eq has-redex(C) = false .
  eq has-redex(S U V W) = true .
  eq has-redex(K U V) = true .
  eq has-redex(I U) = true .
  eq has-redex(N U) = has-redex(N) or has-redex(U) .
  eq has-redex(N U V) = has-redex(N U) or has-redex(V) .
  eq has-redex(N U V W) = has-redex(N U V) or has-redex(W) .
  eq has-redex(K U) = has-redex(K) or has-redex(U) .
  eq has-redex(K U V W) = has-redex(K U V) or has-redex(W) .
  eq has-redex(I U V) = has-redex(I U) or has-redex(V) .
  eq has-redex(I U V W) = has-redex(I U V) or has-redex(W) .
  eq has-redex(S U) = has-redex(S) or has-redex(U) .
  eq has-redex(S U V) = has-redex(S U) or has-redex(V) .
  eq has-redex((X Y) U V W) = has-redex((X Y) U V) or has-redex(W) .
endfm
```

Figure 3.12: Maude CL `has-redex` Predicate: Equational Definition

to the `has-redex` function. But note that by inference rule (3) of Case (1) of the order-sorted pattern difference algorithm, for a unary function symbol $f$, if $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\} = \{u_1, \ldots, u_m\}$, then $\{f(t)\} - \{f(t)\sigma_1, \ldots, f(t)\sigma_n\} = \{f(u_1), \ldots, f(u_m)\}$. Therefore, we can reduce the problem to computing the pattern difference:

$$\{\texttt{U V}\} - (\{\texttt{I U}\} \cup \{\texttt{K U V}\} \cup \{\texttt{S U V W}\} \cup \{\texttt{C}\} \cup \{\texttt{N}\}) \tag{3.31}$$

where `U`,`V`, and `W` are of sort `CL`, `C` is of sort `Constant`, and `N` is of sort `Name`. After passing the query to the tool, we obtain the pattern set:

$$\begin{gathered}\{\texttt{I U V}\} \cup \{\texttt{I U V W}\} \cup \{\texttt{K U}\} \cup \{\texttt{K U V W}\} \cup \\ \{\texttt{S U}\} \cup \{\texttt{S U V}\} \cup \{\texttt{N U}\} \cup \{\texttt{N U V}\} \cup \{\texttt{N U V W}\} \cup \{\texttt{(X Y) U V W}\}\end{gathered} \tag{3.32}$$

where `X` and `Y` are also of sort `CL`. Given this set of result patterns, we can construct the corresponding expanded equational module in Figure 3.12.

Note that some of the expanded calls to `has-redex` are redundant, since we can prove that they will always reduce to true/false. For example, since we have `has-redex(I U) = true`, we get `has-redex(I U V) = true`. The general point is that the righthand sides of the equations in the above module can be *simplified* by means of other equations in the module. In this way, we get the equivalent, simpler specification:

Since this transformed module is purely equational, we can use standard analysis techniques for equational modules, for example the Maude Sufficient Completeness Checker, Church-Rosser Checker, and Termination

```
fmod CL-REDEX-EQ-SIMPL is
  pr CL-SYNTAX .
  pr BOOL-OPS .
  op has-redex : CL -> Bool .

  vars N : Name .  var C : Constant .
  vars U V W X Y : CL .

  eq has-redex(N) = false .
  eq has-redex(C) = false .
  eq has-redex(S U V W) = true .
  eq has-redex(K U V) = true .
  eq has-redex(I U) = true .
  eq has-redex(N U) = has-redex(U) .
  eq has-redex(N U V) = has-redex(U) or has-redex(V) .
  eq has-redex(N U V W) = has-redex(U) or has-redex(V) or has-redex(W) .
  eq has-redex(K U) = has-redex(U) .
  eq has-redex(K U V W) = true .
  eq has-redex(I U V) = true .
  eq has-redex(I U V W) = true .
  eq has-redex(S U) = has-redex(U) .
  eq has-redex(S U V) = has-redex(U) or has-redex(V) .
  eq has-redex((X Y) U V W) = has-redex((X Y) U V) or has-redex(W) .
endfm
```

Figure 3.13: Maude CL `has-redex` Predicate: Simplified Equational Definition

```
mod R&W is
  sorts Nat Config .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  op <_,_> : Nat Nat -> Config .

  vars R W : Nat .

  rl [1] : < 0, 0 > => < 0, s(0) > .
  rl [2] : < R, s(W) > => < R, W > .
  rl [3] : < R, 0 > => < s(R), 0 > .
  rl [4] : < s(R), W > => < R, W > .
endm
```

Figure 3.14: Maude Readers/Writers Theory

Tool. After running these tools, we found the transformed module is sufficiently complete, confluent, and terminating.

### 3.7.3 Invariant Specification and Verification

In the semantic framework of rewriting logic [50], a concurrent system can be specified as the initial model $\mathcal{T}_{\mathcal{R}}$ of a *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$, where $(\Sigma, E)$ is an equational theory specifying the system states, and $R$ are rewrite rules specifying the system's concurrent transitions. Consider, for example, the following high-level specification in Maude of a protocol for the readers-writers problem, where access of readers, resp. writers, to a shared resource is kept track of by a pair of counters `< r, w >`, with `r` the current number of readers and `w` the current number of writers accessing the shared resource.

Maude's `mod`, resp. `endm`, keyword introduces, resp. ends, the specification of a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ with equations $E$ specified by the `eq` keyword, and rules $R$ specified by the `rl` keyword. In this module $E = \varnothing$, and $R$ consists of the above four rules. The system's initial state is the state `< 0, 0 >`. Two key correctness properties that we would like to verify about this protocol are:

1. **Mutual Exclusion**: readers and writers should never access the resource simultaneously.
2. **One Writer**: At most one writer should access the resource at any time.

Although extremely simple, the above system is *infinite state*, making explicit-state model checking verification of properties (1)-(2) impossible. Several model checking and/or deductive verification methods can of course be used. For example, in Chapter 12.4 of [6] an *equational abstraction* [54] making the system finite-state is used, so that explicit-state model checking becomes possible. The issue to be investigated here is whether the pattern algebra we have presented in this chapter could provide an alternative, easy to use method to verify *invariants.* of an infinite-state system, such as the above properties (1)-(2) in our example.

Here is an easy method of this nature that can be applied to any rewrite theory involving only rules, i.e., $\mathcal{R} = (\Sigma, \varnothing, R)$, and such that the rules $R$ are *topmost* [6], meaning that: (i) there is a chosen *State* sort for states (in this example the `Config` sort), and (ii) all rules in $R$ have sort *State* and are applied to the entire state (technically, this is expressed by the syntactic requirement that any proper subterm of a term of sort *State* can never be of sort *State*, which is obviously the case for terms of sort `Config` in our example). What needs to be verified about an invariant $Q$ is that: (i) initial states in a set $I$ of initial sates satisfy $Q$, and (ii) any state reachable from an initial state also satisfies $Q$.

The method is as follows:

1. Specify $Q$ and $I$ by respective sets of pattens.

2. Verify symbolically that $I \subseteq Q$.

3. Verify also symbolically for each rule $l \to r$ in $R$ and substitution $\theta$ that if $l\theta$ satisfies $Q$, then $r\theta$ satisfies $Q$.

The method is of course *inductive*, in that it only requires that the invariant is maintained for *one-step* transitions. Steps 1-2 are straightforward using the techniques of this chapter. $I$ is typically given with the problem, but there is plenty of freedom in practice to choose a suitable $Q$. Since, intuitively, an invariant is a safety property stating that "something bad will never happen," the "weakest" possible invariant, i.e., the one satisfied by the *largest* possible number of states, is the *negation* of such a bad state of affairs. One can of course "strengthen" the invariant $Q$, so that a smaller set of states satisfy $Q$. In our example, the weakest possible invariant $Q$ comprising properties (1)–(2) is precisely the pattern set:

$$\{\langle n,m\rangle\} - \{\langle s(x),s(y)\rangle, \langle x', s(s(y'))\rangle\} \tag{3.33}$$

where the variables $n, m, x, y, x', y'$ all have sort `Nat`. Step 3 is also easy to describe in detail. If $Q$ is described by the set of patterns $\{u_1,\ldots,u_k\}$ and $R = \{l_i \to r_i\}_{i\in I}$, what we need to check symbolically is the set containment:

$$\bigcup_{i\in I}\{r_i\theta \mid \theta \in \bigcup_{1\leqslant j\leqslant k} DUnif_{\Sigma\cup\Sigma^{\#}}(l_i,u_j)\} \subseteq \{u_1,\ldots,u_k\} \tag{3.34}$$

Let us apply this method to our readers-writers example. Firstly, we must compute invariant $Q$. Inputting pattern expression 3.33 into the tool, we obtain:

$$\texttt{< M:Nat, 0 > U < 0, s(0) >} \tag{3.35}$$

Now we check simultaneously: (i) that the initial pattern `< 0, 0 >` is a subset of invariant $Q$ and (ii) that the righthand sides of each rule instantiated by unifiers between $Q$ and each lefthand side are a subset of $Q$. Computing all of the unifiers between $Q$ and the lefthand sides of rules [1]–[4] expands into eight unification problems. For brevity we only show the unifiers computed for each rule: [1] : $id$, [2] : $\{\mathtt{R} \mapsto 0, \mathtt{W} \mapsto 0\}$, [3] : $id$, [4] : $\{\mathtt{R} \mapsto \mathtt{R}, \mathtt{W} \mapsto 0\}$. Thus, we obtain the corresponding righthand side instances: [1] : `< 0, s(0) >`, [2] : `< 0, 0 >`, [3] : `< s(R), 0 >`, [4] : `< R, 0 >`. Finally, we need to check each of the patterns in (i)–(ii) is a subset of $Q$ or, alternatively, that their union is a subset of $Q$. To check that such a union (call it $P$) is contained in $Q$, we can use our tool to check that the set difference $P - Q$ equals $\varnothing$ as follows:

Thus, we established the pattern containment, as required. We will revisit these ideas in chapters 5 and 6 and expand on the ideas here. For now, we just provide the intuition that often a constructor pattern alone

```
(solve-pattern({ < 0,0 > } U { < 0,s(0)> } U { < 0,0 > } U
                  { < s(R:Nat),0 > } U { < R:Nat,0 > }) -
                 ({ < M:Nat,0 > } U { < 0,s(0)> }) .)
Result:  mt
```

Figure 3.15: Tool Output for Invariant Pattern Problem

is *not* sufficient to describe invariants of complex systems; this will require us to move to the richer setting of constrained constructor patterns.

## 3.8 RELATED WORK AND CONCLUSIONS

On pattern operations the most closely related work is [7, 8, 9, 10, 11] and references there. On equational formulas in initial algebras the most closely related work is [14, 15, 16, 17, 18, 36] and references there. On regular languages and tree automata techniques the most comprehensive work is probably [55]; there are also important uses of tree automata technique in [18]. The relationships to work in these three areas have been discussed in detail in previous sections. We summarize them below.

Regarding pattern operations, the main new contributions in relation to work in [7, 8, 9, 10, 11] are in extending that work from the untyped setting to languages with types, subtypes, and subtype polymorphism. As explained in the Introduction, this extension is non-trivial, because it fails rather badly for an arbitrary order-sorted signature $\Sigma$ and requires the new idea of the signature transformation $\Sigma \mapsto \Sigma^{\#}$.

Regarding equational formulas in initial algebras, the main new contributions in relation to work in [14, 15, 16, 17, 18, 36] is in providing two useful new reductions: (i) from the validity problem for an equational formula in an initial order-sorted algebra to the corresponding validity problem for an equational formula in an initial many-sorted algebra; and (ii) from validity problem for a formula with membership constraints in an initial order-sorted algebra to the corresponding validity problem for an equational formula in an initial order-sorted algebra. In particular, reduction (ii) provides a simpler way to deal with both validity and satisfiability problems for formula with membership constraints in an initial order-sorted algebra and makes available the techniques in [36] for quantifier-free formulas.

Regarding regular languages and tree automata, the main new contributions in relation to work in [55, 18] are in identifying the class of *linear pattern languages* (essentially co-extensive with the sort expressions in [18]) and in showing that the usual operations on regular tree languages performed so far by tree automata do have, for linear pattern languages, the considerably simpler alternative of being performed as pattern operations with no recourse to any tree automata operations. A finer point, not apparent from this remark, is that, since tree automata and order-sorted signatures are *la même chose*, when tree automata operations are viewed as operations on order-sorted signatures, they can wreak havoc on the finer, yet essential, properties of order-sorted signatures, such as regularity or preregularity, which are crucial for efficient symbolic computation with order-sorted terms. This means that the advantage of completely replacing tree automata operations by pattern operations for linear pattern languages are not just those of a simpler, more direct method —not requiring any detour through automata and special tools to do so— but also those of always remaining within an order-sorted signature enjoying all the desired properties.

To conclude, we have shown that the untyped algorithms for pattern operations break down when performing the order-sorted pattern operations needed in current declarative languages supporting subtypes and subtype polymorphism, and have shown that such operations can be defined using a signature transformation

$\Sigma \mapsto \Sigma^{\#}$. We have also shown that this transformation yields new insights and a new, quite simple proof of the known decidability of the first-order theory of an initial order-sorted algebra. Furthermore, these results extend the many well-known applications of pattern operations to the richer order-sorted setting and open up new applications such as those discussed in Section 3.7. We plan to work on various other applications and to further advance the current implementation to make it part of the Maude formal tool environment.

# CHAPTER 4 METALEVEL ALGORITHMS FOR VARIANT SATISFIABILITY[1]

## 4.1 INTRODUCTION

SMT solving is at the heart of some of the most effective theorem proving and model checking formal verification methods that can scale up to impressive verification tasks. A current limitation, however, is its *lack of extensibility*: current SMT solvers support a (typically small) library of decidable theories. Although these theories can be combined by the Nelson-Oppen (NO) [19, 20] or Shostak [21] methods under some conditions, only the theories in the SMT solver library and their combinations are available to the user: any other theories extending the tool must be implemented by the tool builders.

In practice, of course, the problem a user has to solve may not be expressible using only the theories available in an SMT solver's library. Therefore, the goal of making SMT solvers *user-extensible*, so that a *user* can easily *define* new decidable theories and use them in the verification process is highly desirable, because it widens the range of decidable subproblems of a verification problem.

Recall that *E-unifiability* is a well-known *subproblem* of SMT solving, namely, satisfiability in the free $(\Sigma, E)$-algebra $T_{\Sigma/E}(X)$ on countably many variables $X$, but restricted to *positive* (i.e., negation-free) quantifier-free (QF) formulas. Until recently, unification tools also suffered from a lack of extensibility: such tools usually support a small library of theories $(\Sigma, E)$, combinable by methods similar to the NO method ([22] explicitly relates the NO method and combination algorithms for unification). Again, the *user* could not extend such *decidable* unifiability/unification algorithms by defining new theories and using a *theory-generic* algorithm. However, true user-extensibility has now been achieved for $E$-unification in theories $(\Sigma, E)$ satisfying the *finite variant property* (FVP) [23] thanks to *variant unification* based on *folding variant narrowing* [24]. In fact, variant unification for user-definable FVP theories is already supported by Maude 2.7.1.

This suggests an obvious question: could variant unification be generalized to *variant satisfiability*, so that, under suitable conditions on an FVP theory $(\Sigma, E)$, satisfiability of QF formulas in the initial algebra $T_{\Sigma/E}$ becomes *decidable* by a *theory-generic* satisfiability algorithm? This would then make satisfiability *user-extensible* as desired. This question has been positively answered in [25, 36] by giving general conditions under which satisfiability of QF formulas in the initial algebra $T_{\Sigma/E}$ of an FVP theory $(\Sigma, E)$ is decidable.

However, the results in [25, 36] do not really provide an *algorithm* in the full sense of the word, but rather a theoretical *skeleton* on which such an algorithm can be fleshed out. The goal of the present chapter is to design and prove correct several subalgorithms that are absolutely essential in order to obtain a complete variant satisfiability algorithm. We then also present a Maude implementation of the variant satisfiability algorithm and its subalgorithms using the reflective capabilities of Maude's `META-LEVEL` module. In Section 4.1.1 below we first summarize some of the key concepts from [25, 36]. Then, in Section 4.1.2, we summarize the variant satisfiability algorithm and subalgorithms presented in this chapter.

### 4.1.1 Variant Satisfiability in a Nutshell

Variant-based satisfiability [25, 36] is a theory-generic decision procedure that applies to the following, easily user-specifiable infinite class of equational theories $(\Sigma, E \cup B)$:

[1] Chapter content was originally available at `https://doi.org/10.1016/j.jlamp.2017.12.006`. Reprinted with permission.

1. $\Sigma$ is an order-sorted [33] signature of function symbols, supporting types, subtypes, and subtype polymorphism;

2. the equations $E \cup B$ have the *finite variant property* [23] (more on this below), where $B$ is a set of equational axioms—such as commutativity and/or associativity commutativity and/or identity—that have a finitary unification algorithm; and

3. $(\Sigma, E \cup B)$ *protects* a constructor subtheory $(\Omega, E_\Omega \cup B_\Omega)$—that is each ground $\Sigma$-term is $E_\Omega \cup B_\Omega$-equal to a ground $\Omega$-term—and furthermore $(\Omega, E_\Omega \cup B_\Omega)$ is an OS-*compact* theory [25, 36] (more on this below).

The procedure can then decide satisfiability in the initial algebra $T_{\Sigma/E\cup B}$, that is, in the *algebraic data type* specified by $(\Sigma, E \cup B)$. Conditions (1)–(3) above apply to a useful infinite class of *user-definable algebraic data types.*

The notions of *variant* and of *OS-compactness* mentioned above are defined in detail in Section 4.2. Here we give some key intuitions about each notion. Given $\Sigma$-equations $E \cup B$ such that the equations $E$ oriented as left-to-right rewrite rules are confluent and terminating modulo the equational axioms $B$, a *variant* of a $\Sigma$-term $t$ is a pair $(u, \theta)$ where $\theta$ is a substitution, and $u$ is the canonical form of the term instance $t\theta$ by the rewrite rules $E$ modulo $B$. Intuitively, the variants of $t$ are the fully simplified *patterns* to which the instances of $t$ can reduce. Some simplified instances are of course more general (as patterns) than others. $E \cup B$ has the *finite variant property* (FVP) if any $\Sigma$-term $t$ has a *finite* set of most general variants. We can first illustrate FVP by its absence:

**Example 4.1** *(Peano Addition). The theory* $(\Sigma, E)$ *with* $\Sigma = \{0, s, +\}$ *and* $E = \{x + 0 = x, x + s(y) = s(x + y)\}$ *the addition equations (so here* $B = \varnothing$*), is* not *FVP, since* $(x + y, id)$, $(s(x + y_1), \{y \mapsto s(y_1)\})$, $(s(s(x + y_2)), \{y \mapsto s(s(y_2))\})$, *…,* $(s^n(x + y_n), \{y \mapsto s^n(y_n)\})$, *…, are all* incomparable *variants of* $x + y$.

Here is a very simple, yet interesting, example of an FVP theory:

**Example 4.2** *(Presburger Arithmetic). This is a theory* $(\Sigma, E \cup B)$ *where* $\Sigma$ *has two sorts, Nat and Truth. Nat has constants* $0$ *and* $1$ *and a binary function symbol* $+$*, Truth has constants* $\bot$ *and* $\top$*, and we have order predicates* $\_ > \_, \_ \geqslant \_ :$ *Nat Nat* $\rightarrow$ *Truth defined by the equations* $E = \{1 + m + n > n = \top, m > m + n = \bot, m + n \geqslant m = \top, n \geqslant 1 + m + n = \bot\}$*. The equations* $B = \{n + m = m + n, (n + m) + k = n + (m + k), n + 0 = n\}$ *are the associativity and commutativity axioms for* $+$ *with identity* $0$*. This specification* is *FVP. For example, the term* $n > m$ *has three variants:* $(n > m, id)$*,* $(\top, \{n \mapsto m + n' + 1\})$*, and* $(\bot, \{m \mapsto n + n'\})$*.*

Of course, Presburger arithmetic is decidable. The variant satisfiability algorithm, however, is *specification-dependent.* Since we have seen that Peano addition is not FVP, variant satisfiability will *not* apply to a specification of Presburger arithmetic extending Peano addition with $>$ and $\geqslant$. But it *does* apply to the specification in Example 4.2. Let us further explain why Example 4.2 satisfies condition (2). FVP intuitively means (more on this in [23, 41]) that, given any term, we can give a *bound* on the number of rewrite steps required to reduce to canonical form any of its instances by irreducible substitutions. This is clearly the case for Example 4.2, since all terms of sort *Nat* are irreducible, and all terms of sort *Truth* are either irreducible or reducible in just *one* rewrite step. By contrast, in Example 4.1 there is *no bound* on the number of rewrite steps needed to reduce to canonical form any instance $(x + y)\sigma$ of the term $x + y$ by an irreducible substitution $\sigma$. Furthermore, Example 4.2, besides trivially satisfying condition (1), also satisfies condition

(3) as we now explain. The equations $E$ fully define $>$ and $\geqslant$. Therefore, the subsignature $\Omega$ of $\Sigma$ obtained by removing $>$ and $\geqslant$, gives us a *protecting subtheory inclusion* $(\Omega, B) \subset (\Sigma, E \cup B)$. The key property then ensuring decidability of quantifier-free (QF) formulas in $T_{\Sigma/E \cup B}$ is that the constructor subtheory $(\Omega, B)$ is *OS-compact.* Roughly speaking (see Section 4.2 for a precise definition) this means that $B$-equality is decidable and that a conjunction of $\Omega$-disequalities $\bigwedge_i u_i \neq v_i$ where all the variables range over *infinite sorts* in $T_{\Omega/B}$ (in our example over the sort *Nat*) is satisfiable in $T_{\Omega/B}$ iff for each $i$ we have $u_i \neq_B v_i$, where $=_B$ denotes provable $B$-equality. In fact, it is proved in [25, 36] that for $B$ any combination of associativity and/or commutativity and/or identity axioms, except associativity without commutativity, any $(\Omega, B)$ is OS-compact, and that then QF satisfiability in $T_{\Omega/B}$ is decidable.

The key idea of variant satisfiability is to *reduce* satisfiability of QF formulas in $T_{\Sigma/E \cup B}$ to satisfiability of semantically equivalent $\Omega$-formulas in $T_{\Omega/B_\Omega}$ (in Example 4.2, $B = B_\Omega$, but generally $B \supseteq B_\Omega$). This reduction is achieved as follows. Since any QF formula can be put in disjunctive normal form and a disjunction is satisfiable in $T_{\Sigma/E \cup B}$ iff one of the disjuncts is, it is enough to decide satisfiability of a conjunction of $\Sigma$-atoms of the form $\bigwedge G \wedge \bigwedge D$, where $G$ is a set of equalities and $D$ a set of disequalities. Then the variant satisfiability algorithm proceeds as follows:

- **Equation Elimination**. $\bigwedge G \wedge \bigwedge D$ is semantically equivalent to disjunction $\bigvee_{\alpha \in Unif^{\Omega}_{E \cup B}(\bigwedge G)} \bigwedge D\alpha$, where the FVP property is exploited to effectively compute the so-called *constructor* $E \cup B$*-unifiers* of the conjunction of equations $\bigwedge G$.
- **Constructor Variant Computation**. Each conjunction of $\Sigma$-disequalities $\bigwedge D\alpha$ so obtained is semantically equivalent to a disjunction of the form $\bigvee_j \bigwedge D'_j$, where each $\bigwedge D'_j$ is a so-called *constructor variant* of $\bigwedge D\alpha$. That is, a variant of $\bigwedge D\alpha$ for the equations $E \cup B$ when we view conjunction $\wedge$ and disequality $\neq$ as new, free function symbol, such that $\bigwedge D'_j$ is an $\Omega$-formula.
- **OS-Compact Satisfiability Check**. After instantiating variables in finite sorts, by OS-compactness each such $\bigwedge D'_j$, say with $\bigwedge D'_j = \bigwedge_i u_i \neq v_i$, can be automatically checked by checking $u_i \neq_B v_i$.

We can illustrate this algorithm by proving the linear order property of $\geqslant$, i.e., the formula $m \geqslant n = \top \vee n \geqslant m = \top$. This is equivalent to proving $m \geqslant n \neq \top \wedge n \geqslant m \neq \top$ unsatisfiable. In this case, we do not need to eliminate any equalities, so we can proceed to the second step of **Constructor Variant Computation**. There are three such constructor variants, namely, $(\top \neq \top \wedge \top \neq \top, \{n \mapsto m\})$, $(\top \neq \top \wedge \bot \neq \top, \{m \mapsto 1+n+k\})$, $(\top \neq \top \wedge \bot \neq \top, \{n \mapsto 1+m+k\})$. Since all are unsatisfiable, $m \geqslant n = \top \vee n \geqslant m = \top$ is a theorem of Presburger arithmetic.

### 4.1.2 The Variant Satisfiability Algorithm and its Subalgorithms

So, isn't the variant satisfiability algorithm summarized above already defined? As pointed out earlier, the answer is *no*: the algorithm's main skeleton is indeed defined in [25, 36]; but three essential components of the algorithm are not. The main three missing components, or subalgorithms, are:

1. *Constructor Variants Algorithm.* The constructor variants of a term are variants that are constructor terms. Their definition is given in [25]; but no algorithm is given there to compute a *complete finite set* of constructor variants of a given term. This problem is easy for Example 4.2, because there we can easily *syntactically distinguish* constructor terms from terms having defined symbols (in Example 4.2

the only defined symbols are $>$ and $\geqslant$). But the problem is considerably more subtle when, by subsort polymorphism, a symbol is *simultaneously* a constructor for some typings and a defined symbol for other typings. This is illustrated by our running example (Example 4.3 presented later in this section), where $+$ has two constructor typings, namely, $\_+\_ : \mathit{Nat}\ \mathit{Nat} \to \mathit{Nat}$ and $\_+\_ : \mathit{NzNat}\ \mathit{NzNat} \to \mathit{NzNat}$, with subsort inclusions $\mathit{NzNat} < \mathit{Nat} < \mathit{Int}$ of non-zero naturals into naturals, and of naturals into integers, but where $\_+\_ : \mathit{Int}\ \mathit{Int} \to \mathit{Int}$ is *not* a constructor, but a function *defined* by the three equations listed in Example 4.3.

2. *Constructor Unification Algorithm.* For the exact same reasons, although the notion of constructor unifier is defined in [25], no unification algorithm is given there to compute a finite *complete set* of constructor unifiers of a unification problem. This is again easy when no function symbols have both constructor and defined symbol typings and no axioms in $B$ can make a non-constructor term equal to a constructor term; but it is considerably more subtle when a symbol can be both a constructor for some typings and a defined symbol for other typings.

3. *Auxiliary OS-Compactness Algorithms.* Although the notion of OS-compact theory is defined in [25], and the theorem that all theories of the form $(\Omega, B)$ with $B$ any combination of associativity and/or commutativity an/or identity axioms, except associativity without commutativity, is also proved there, what are missing are algorithms to: (i) *automatically check* that a constructor subtheory is OS-compact; (ii) *automatically compute* which sorts in $T_{\Omega/B}$ are finite and which are infinite; (iii) for finite sorts, to *automatically compute* a *canonical representative* $t' \in [t]_B$ for each $B$-equivalence class in a finite sort; and (iv) *decide satisfiability* of a QF $\Omega$-formula in $T_{\Omega/B}$.

Without these three missing components, the variant satisfiability algorithm is only partially defined and cannot be implemented. The main goal of this chapter is to: (a) define such subalgorithms; (b) prove them correct, and (c) derive a reflective meta-level implementation of them and of the entire variant satisfiability algorithm using Maude's `META-LEVEL` module. Components (1)–(3) above have themselves other sub-components. The entire hierarchy of subalgorithms developed in this chapter is summarized in the tree below:

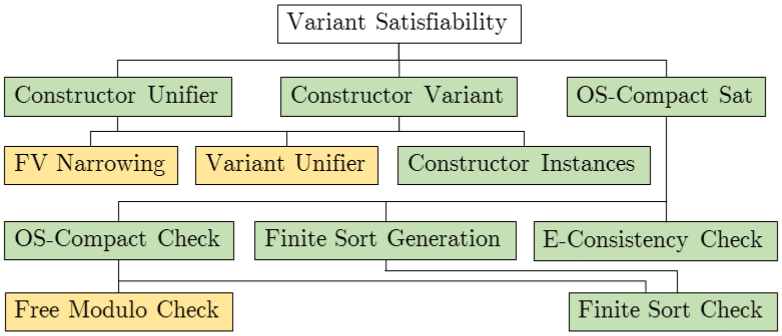


Figure 4.1: Variant Satsifiability Subalgorithm Hierarchy

Blocks highlighted in green represent the new *algorithmic* contributions of this chapter, while blocks highlighted in yellow represent existing algorithms already available: folding variant narrowing and variant unification are available in Maude 2.7.1, and the tree automata check for the freeness of constructors $\Omega$ modulo axioms $B$ is available in the CETA library underlying Maude's SCC tool [56, 57].

A few remarks can be appropriate to clarify in which sense the above algorithms are *metalevel* algorithms. The sense is the obvious one: we could just as well have described them, in different but equivalent words, as *theory-generic algorithms*: they are not defined for a *fixed* theory, say as associative-commutative unification is, but for an infinite class of theories. This necessarily means that they must be described *at the meta-level* of order-sorted equational logic. This is of course harmless and in a sense transparent at the theoretical level: we do this all the time. For example, syntactic unification, or congruence closure are likewise metalevel algorithms in this precise sense, since they are both generic on the signature Σ. Therefore, this places *no special demands* on the *theoretical definition and proof* of the algorithms: we can use standard mathematical notation. The reason why we emphasize the algorithms' metalevel nature is that, as further explained in Section 4.4, we can exploit the fact that both order-sorted equational logic and rewriting logic are reflective [58], and that their reflection is efficiently *reified* in Maude's `META-LEVEL` module [6], to easily derive a reflective Maude *implementation* of these algorithms by defining them as metalevel functions extending Maude's `META-LEVEL` module.

**Chapter Overview.** The main contributions of this chapter are: (i) the design of all the subalgorithms needed to have a full definition of the variant satisfiability algorithm; (ii) proofs of correctness for these subalgorithms; and (iii) a *reflective* Maude implementation of variant satisfiability. More concretely, all the subalgorithms marked in green in the above hierarchy are defined, proved correct, and implemented as extensions of Maude's `META-LEVEL` module. These subalgorithms plus existing *folding variant narrowing*, *variant unification*, and *freeness modulo checking* algorithms are the building blocks for our Maude implementation of variant satisfiability.

In this section, we make heavy use of the preliminaries on rewriting as equational deduction in Section 2.2.1; please refer liberally if needed. In this chapter, we assume all rewrite theories are unconditional, in sense of Definition 2.8 or 2.5.

**Example 4.3** *(Integers with Addition). The running example in this chapter is the theory of integers with addition* $\mathcal{Z}_+$. *The signature* Σ *and constructor subsignature* Ω *are specified in Figure 4.2 below. Both signatures have sorts Nat, NzNat, NzNeg, and Int, and subsorts NzNat < Nat and Nat NzNeg < Int, where NzNat (resp. NzNeg) denotes the non-zero naturals (resp. negatives). The constructor subsignature* Ω *has constants 0 of sort Nat and* 1 *of sort NzNat, and operators* $\_+\_ : \mathit{Nat}\ \mathit{Nat} \to \mathit{Nat}$, $\_+\_ : \mathit{NzNat}\ \mathit{NzNat} \to \mathit{NzNat}$, *and* $- : \mathit{NzNat} \to \mathit{NzNeg}$ *shown in blue. The signature* Σ *contains all the operators defined in* Ω *and adds one defined function symbol:* $\_+\_ : \mathit{Int}\ \mathit{Int} \to \mathit{Int}$ *shown in red. Let* $B$ *be the set of ACU axioms for* $(+)$ *with identity* 0 *and the equations* $E_0$ *defining* $(+)$ *be the following (with variables* $i\!:\!\mathit{Int}$, $n\!:\!\mathit{NzNat}$, *and* $m\!:\!\mathit{NzNat}$*)*

$$i + n + -(n) = i \tag{4.1}$$

$$i + -(n) + -(m) = i + -(n + m) \tag{4.2}$$

$$i + n + -(n + m) = i + -(m) \tag{4.3}$$

*Then* $(\Sigma, B, R)$ *is a decomposition of the theory* $(\Sigma, B \cup E_0)$ *with* $R = \overrightarrow{E_0}$. *Furthermore* $(\Sigma, B, R)$ *protects the constructor decomposition* $(\Omega, B, \varnothing)$*, i.e.* Σ *is sufficiently complete with respect to* Ω *modulo* $B$.

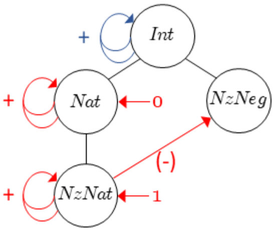


Figure 4.2: Signatures $\Sigma$ and $\Omega$ for theory $Z_+$

*Finally, we explain how Figure 4.2 corresponds to $Z_+$. For each sort $s \in S$, there is a circle which bears its name; for each subsort $(s, s') \in (<)$, there is a solid black line drawn from sort $s$ to sort $s'$. Finally, for each operator $f$, there are lines drawn from its argument sorts $s_1, \cdots, s_n$ that merge into one arrow entering its output sort $s$. Since constants have no argument sorts, they are just arrows entering a sort.* In the sequel, constructor operators (resp. defined operators) will be drawn in red (resp. blue).

## 4.2 VARIANT UNIFICATION AND VARIANT SATISFIABILITY

Folding variant narrowing [24] can be used as part of an $E$-unification algorithm, called *variant unification*, whenever theory $(\Sigma, E)$ has a decomposition $\mathcal{R} = (\Sigma, B, R)$ and $B$ has a finitary unification algorithm [24].

**Definition 4.1** *Let $\Sigma^\wedge$ denote the signature $\Sigma$ extended to represent the language $\Sigma$-equations, i.e. let $\Sigma^\wedge$ denote $\Sigma^\top$ extended by adding:*

1. *fresh sorts Lit and Conj with a subsort inclusion Lit < Conj and a binary conjunction operator $\_ \wedge \_ : Lit\ Conj \to Conj$, and*
2. *for each connected component $[s] \in \widehat{S_\top}$ with top sort $\top_{[s]}$, binary operators $\_ = \_ : \top_{[s]}\ \top_{[s]} \to Lit$ and $\_ \neq \_ : \top_{[s]}\ \top_{[s]} \to Lit$.*

*Given $\mathcal{R} = (\Sigma, B, R)$, let $\mathcal{R}^\wedge = (\Sigma^\wedge, B, R)$. Note that, since we are just adding free constructors in fresh new sorts, $\mathcal{R}^\wedge$ is FVP iff $\mathcal{R}$ is FVP.*

**Theorem 4.1** *(Variant Unification). [25] For $\mathcal{R} = (\Sigma, B, R)$ an FVP decomposition of $(\Sigma, E)$, where $B$ has a finitary $B$-unification algorithm, let $\phi$ be a system of $\Sigma$-equations viewed as a $\Sigma^\wedge$-term of sort Conj. Then, for any finite set $W$ of variables $W \supseteq vars(\phi)$, the set $VarUnif_E^W(\phi)$ of* variant $E$-unifiers *of $\phi$ away from $W$ is by definition the set:*

$$\{(\theta\gamma)|_{vars(\phi)} \mid (\phi', \theta) \in [\![\phi]\!]_{R,B}^W \ \wedge\ \gamma \in Unif_B^{W,\theta}(\phi') \ \wedge\ (\phi'\gamma, (\theta\gamma)|_{vars(\phi)}) \in [\![\phi]\!]_{R,B}^*\} \tag{4.4}$$

*Furthermore, the generation of $VarUnif_E^W(\phi)$ by folding variant narrowing of $\phi$ followed by $B$-unification* always terminates *with a* finite *set of unifiers, thus providing a finitary $E$-unification algorithm.*

As already summarized in Section 4.1.1, *variant satisfiability* [25, 36] extends the variant unification Theorem 4.1 above to a general method to decide satisfiability for *any* QF equational $\Sigma$-formula in $T_{\Sigma/E \cup B}$. Essentially, the method outlined in [25] divides the process of deciding the satisfiability of a conjunction of literals $\bigwedge G \wedge \bigwedge D$ with $G$ equalities and $D$ disequalities into three main steps:

1. **Equation Elimination**. The conjunction of equations $\bigwedge G$ is removed by computing its set $Unif_E^\Omega(\bigwedge G)$ of *constructor unifiers*, thus obtaining a semantically equivalent formula $\bigvee_{\alpha \in Unif_E^\Omega(\bigwedge G)} \bigwedge D\alpha$.

2. **Constructor Variant Computation**. Reduce the satisfiability of $\bigwedge D\alpha$ in $T_{\Sigma,E}$ to an equisatisfiable problem in the constructor subtheory $(\Omega, E_\Omega)$'s initial algebra $T_{\Omega,E_\Omega}$ using the *constructor variants* of $\bigwedge D\alpha$.

3. **OS-Compact Satisfiability Check**. Assuming $(\Omega, E_\Omega)$ belongs to the class of *OS-compact* equational theories for which satisfiability in the initial algebra is decidable, check whether any of the constructor variants $\bigwedge D'$ of $\bigwedge D\alpha$ is satisfiable in $T_{\Omega,E_\Omega}$.

Technically, the constructor unifier notion is not needed by the proofs; however, it provides an important gain in efficiency by throwing out unnecessary unifiers which are otherwise redundant. We first define the notions constructor variant and constructor unifier below and then show how they are used.

**Definition 4.2** *(Constructor Variant). [25] Let $\mathcal{R} = (\Sigma, B, R)$ be a decomposition of $(\Sigma, E)$, and let $\mathcal{R}_\Omega = (\Omega, B_\Omega, R_\Omega)$ be a constructor decomposition of $\mathcal{R}$. Then an $R, B$-variant $(u, \theta)$ of a $\Sigma$-term $t$ is called a* constructor $R, B$-variant *of $t$ iff $u \in T_\Omega(X)$. Let $[\![t]\!]^{\Omega}_{R,B}$ denote a complete set of constructor variants of a term $t$, i.e. for each constructor variant $(v, \beta)$ of $t$ there is a $(w, \alpha) \in [\![t]\!]^{\Omega}_{R,B}$ and a substitution $\gamma$ such that $v =_B w\gamma$, and $\beta =_B (\alpha\gamma)|_{vars(t)}$. Similarly, let $[\![t]\!]^{\Omega,W}_{R,B}$ denote a complete set of constructor variants "away from $W$" as explained in Definition 2.13.*

**Definition 4.3** *(Constructor Unifier, Constructor Variant Unifier). An $R, B$-normalized $E$-unifier $\theta$ of unification problem $\phi \equiv u_1 = v_1 \wedge \ldots \wedge u_n = v_n$ is a* constructor $E$-unifier *of $\phi$ iff terms $(u_i\theta)!_{R,B}, (v_i\theta)!_{R,B} \in T_\Omega(X)$, $1 \leqslant i \leqslant n$.*

*If there exists a* variant *$E$-unifier $\alpha$ of $\phi$ and substitution $\beta$ where $\theta =_B \alpha\beta$, we call $\theta$ a* constructor variant unifier. *Let $VarUnif^{\Omega,W}_E(\phi)$ denote a* complete *set constructor variant unifiers of a system of equations $\phi$ with $W \supseteq vars(\phi)$, i.e., for each constructor variant unifier $\theta$ of $\phi$ there is a $\rho \in VarUnif^{\Omega,W}_E(\phi)$ and a substitution $\gamma$ such that $\theta|_W =_B (\rho\gamma)|_W$.*

**Example 4.4** *(Integers with Addition, Constructor Variants/Unifiers). Recall the theory $\mathcal{Z}_+$ from Example 2.2. Even though we do not have an algorithm (yet) to compute $[\![t]\!]^{\Omega,W}_{R,B}$ for a term $t$, we can still manually compute these sets; we will show how they are generated in Section 4.3.2. The notions of constructor variant and constructor unifier become more subtle when, due to order-sortedness, a same subsort-polymorphic operator $f$ has some typings that are constructors and some other typings that are defined functions. Consider now the term $y + z$ with $y, z$ variables of sort $Int$. By folding variant narrowing, it is easy to show $y + z$ has twelve variants in general, but to simplify the example, we focus on its most simple variant, i.e. $u = (y + z, id)$ with $id$ the identity substitution. Note that $u$ is* not *a constructor variant in $\mathcal{Z}_+$, and there are variants* less general *than $(y + z, id)$ that are constructor variants. A* complete *set of constructor variants less general than $(y + z, id)$ is: (i) $(y', \{y \mapsto y', z \mapsto 0\})$, (ii) $(z', \{z \mapsto z', y \mapsto 0\})$, and (iii) $(y' + z', \{y \mapsto y' : Nat, z \mapsto z' : Nat\})$. Likewise, let $\phi$ be the equation $w = y + z$, with $w, y, z$ of sort $Int$. Then $\{w \mapsto y + z\}$ is a trivial $\mathcal{Z}_+$-unifier of $\phi$, but* not *a constructor unifier. A complete set $VarUnif^{\Omega,W}_E(\phi)$ of constructor $\mathcal{Z}_+$-unifiers of $\phi$ less general than $\{w \mapsto y + z\}$ is given by the unifiers: (i) $\{w \mapsto y, z \mapsto 0\}$, (ii) $\{w \mapsto z, y \mapsto 0\}$, and (iii) $\{w \mapsto y' + z', y \mapsto y' : Nat, z \mapsto z' : Nat\}$. Similarly, we show how to generate these unifiers in Section 4.3.2.*

Theorem 4.2 below states that constructor unifiers are enough.

**Theorem 4.2** *(Completeness of Constructor Unifiers). [25] Let $\mathcal{R} = (\Sigma, B, R)$ be a decomposition of $(\Sigma, E)$, and $\mathcal{R}_\Omega = (\Omega, B_\Omega, R_\Omega)$ a constructor decomposition of $\mathcal{R}$, and let $\phi \equiv u_1 = v_1 \wedge \ldots \wedge u_n = v_n$ be a system of $\Sigma$-equations with $Y = vars(\phi)$. Then: (i) if $\delta \in [Y \rightarrow T_\Omega]$ is a $R, B$-normalized ground $E$-unifier of $\phi$, then there is a constructor $E$-unifier $\theta$ of $\phi$ away from $W$ and a substitution $\gamma$ such that $\delta|_W =_B (\theta\gamma)|_W$ (ii) $T_{\Sigma/E} \models (\exists Y)\ \phi$ iff $\phi$ has a constructor $E$-unifier.*

Theorem 4.3 shows how to reduce satisfiability in the initial algebra $T_{\Sigma/E}$ to satisfiability in the algebra $T_{\Omega/E_\Omega}$. This reduction alone does not yet ensure decidability.

**Theorem 4.3** *(Descent Theorem). [25] Let $(\Sigma, E)$ be an OS equational theory having a decomposition $\mathcal{R} = (\Sigma, B, R)$ and protecting a constructor subtheory $(\Omega, E_\Omega)$ with constructor decomposition $\mathcal{R}_\Omega$. Then, a QF $\Sigma$-conjunction of disequalities $\phi$ is satisfiable in $T_{\Sigma/E}$ iff there is a constructor variant $(\phi', \theta)$ of $\phi$ such that $\phi'$ is satisfiable in $T_{\Omega/E_\Omega}$.*

To obtain decidability, we still need to decide satisfiability in the algebra $T_{\Omega/E_\Omega}$. For that, we turn to *OS-compact* theories. The relevant notions are defined below.

**Definition 4.4** *(E-Consistent). Call a $\Sigma$-disequality $u \neq v$ $E$-*consistent *iff $u \neq_E v$. Likewise, call a conjunction $\bigwedge_{i \in I} u_i \neq v_i$ of $\Sigma$-disequalities $E$-*consistent *iff for each $i \in I$, $u_i \neq v_i$ is $E$-consistent.*

**Definition 4.5** *(OS-Compactness). [25] An equational theory $(\Sigma, E)$ is called* OS-compact *iff: (i) for each sort $s$ in $\Sigma$ we can effectively determine whether $T_{\Sigma/E,s}$ is finite or infinite, and, if finite, can effectively compute a representative ground term $rep([u]) \in [u]$ for each $[u] \in T_{\Sigma/E,s}$ (ii) $=_E$ is decidable and $E$ has a finitary unification algorithm; and (iii) any $E$-consistent finite conjunction $\bigwedge D$ of $\Sigma$-disequalities whose variables all have infinite sorts is satisfiable in $T_{\Sigma/E}$.*

All of the pieces are now in place to describe the algorithm's main stages.

### 4.2.1 Variant Satisfiability Algorithm

Let (i) theory $(\Sigma, E)$ protect $(\Omega, E_\Omega)$ (ii) $(\Sigma, E)$ have an FVP decomposition $\mathcal{R} = (\Sigma, B, R)$ that protects a constructor decomposition $R_\Omega = (\Sigma, B, R_\Omega)$ of $(\Omega, E_\Omega)$ (iii) there be a finitary $B$-unification algorithm (iv) $(\Omega, E_\Omega)$ be OS-compact (v) $\phi$ be a QF equational $\Sigma$-formula. To decide satisfiability of $\phi$ in $T_{\Sigma/E}$, our algorithm applies a series of *satisfiability-preserving* transformations:

1. Apply disjunctive normal form (DNF) transformation such that $\phi^{dnf} \equiv \bigvee_{i \in I}(\bigwedge_{j \in J} u'_j = v'_j \wedge \bigwedge_{k \in K} u''_k \neq v''_k)$.

2. For $i \in I$, compute constructor unifiers $\Theta = Unif_E^{\Omega,W}(\bigwedge_{j \in J} u'_j = v'_j)$ with $W \supseteq vars(\phi^{dnf})$. Then let $\phi^{neg} \equiv \bigvee_{\theta \in \Theta}(\bigwedge_{k \in K} u''_k\theta \neq v''_k\theta)!_{R,B}$. By Theorem 4.2, $\phi^{neg}$ is satisfiable iff $\phi^{dnf}$ is.

3. For each $\theta \in \Theta$, compute $\Phi \equiv \llbracket \bigwedge_{k \in K} u''_k\theta!_{R,B} \neq v''_k\theta!_{R,B} \rrbracket^{\Omega,W}_{R,B}$. By Theorem 4.3, $\phi^{neg}$ is satisfiable in $T_{\Sigma/E}$ iff there is a variant $\phi^{var} \in \Phi$ with $\phi^{var}$ satisfiable in $T_{\Omega/E_\Omega}$.

4. For each $\phi^{var} \in \Phi$, generate $\Phi^{inf}$ where each variable $x : s \in vars(\phi^{var})$ with $s$ finite is replaced by a set of its unique representatives. By OS-compactness, this process will terminate.

5. Since each $\phi^{inf} \in \Phi^{inf}$ is a conjunction of $\Omega$-disequalities of infinite sorts, by OS-compactness, check $E_\Omega$-consistency to decide satisfiability in $T_{\Omega/E_\Omega}$.

The papers [25, 36] contain many examples of commonly used theories that have FVP specifications whose constructor decompositions are OS-compact. A first method to show OS-compactness is both very simple and widely applicable to constructor decompositions of FVP theories; it generalizes a similar result in [31] for the unsorted and $AC$ case:

**Theorem 4.4** *[25] If $\Omega$ is a free signature of constructors modulo $B_\Omega$, axioms $B_\Omega$ are subsort-polymorphic, and $B_\Omega \subseteq ACCU$, then theory $(\Omega, B_\Omega)$ is OS-compact and satisfiability of QF $\Omega$-formulas in $T_{\Omega/B_\Omega}$ is decidable.*

We now formally define the kinds of theories our algorithm supports:

**Definition 4.6** *Call equational theory $(\Sigma, E)$ simple iff (i) $\Sigma$ and $E$ are finite (ii) $(\Sigma, E)$ decomposes into $\mathcal{R} = (\Sigma, B, R)$ (iii) $\mathcal{R}$ is FVP, (iv) $B \subseteq ACCU$ and is subsort-polymorphic, and (v) $\mathcal{R}$ protects a constructor decomposition $R_\Omega = (\Omega, B_\Omega, \varnothing)$, i.e., theory $(\Sigma, E)$ protects $(\Omega, B_\Omega)$, a subsignature of free constructors modulo $B_\Omega$.*

Since simple theories have FVP decompositions, finitary $B$-unification algorithms, and OS-compact constructor decompositions, by the above algorithm, satisfiability of QF equational $\Sigma$-formulas in $T_{\Sigma/E} \cong C_{\mathcal{R}}$ is decidable. In the next section, we will see how the finiteness assumptions are used. Recall again the hierarchy of subalgorithms already shown in Figure 4.1. Blocks highlighted in green represent the new *algorithmic* contributions of this chapter, while blocks highlighted in yellow represent existing algorithms. As already explained in Section 4.1.2, none of the algorithms in the green blocks existed prior to this work. Their definition and proof of correctness is the subject of Section 4.3.

## 4.3 METALEVEL ALGORITHMS FOR VARIANT SATISFIABILITY

For *simple* OS equational theories $(\Sigma, E)$, solving *variant satisfiability problems* has been condensed in Section 4.2 down into two phases. The first phase is steps (1)-(3) in the high-level algorithm, which reduce satisfiability of a conjunction of $\Sigma$-literals in $T_{\Sigma/E} \cong C_{\mathcal{R}}$ to satisfiability of a conjunction of $\Omega$-disequalities in $T_{\Omega/B_\Omega} \cong C_{\mathcal{R}_\Omega}$. Then the second phase corresponds to steps (4)-(5) in the high-level algorithm, which decide satisfiability of conjunctions of $\Omega$-disequalities in $C_{\mathcal{R}_\Omega}$ when $(\Omega, B_\Omega)$ is OS-compact.

At a theoretical level we have a *skeleton* of a high-level algorithm for variant satisfiability. But at a concrete, algorithmic level several important questions, essential for obtaining an actual *algorithm*, remain unresolved: (a) how can we *automatically check* a theory is OS-compact and then decide satisfiability of equational formulas in that theory? (b) how can we *compute* constructor variants and constructor variant unifiers? (c) how can we *prove* that the auxiliary algorithms answering questions (a)-(b) are *correct*? and (d) how can we *implement* both the main algorithm and the auxiliary algorithms in a correctness-preserving manner?

Let us begin with question (c). The algorithm skeleton sketched in Section 4.2 is theory-generic and therefore manipulates metalevel entities like operators, signatures, terms, equations, and theories. Likewise, the checks for OS-compactness and the computation of constructor variants and constructor unifiers are problems fully expressible in terms of such metalevel entities. Therefore, both for mathematical clarity and for simplicity of the needed correctness proofs, the definitions of the auxiliary algorithms should be carried out at the metalevel of equational and rewriting logic. As already explained in Section 4.3, this just means that they should be expressed in *standard logical notation*, not for a fixed theory, but for theories in general.

This brings us to question (d), which has a simple answer: since both equational and rewriting logic are *reflective* [58], once we have defined and proved correct at the metalevel the auxiliary algorithms solving questions (a)-(b), we can derive correct implementations for them by *meta-representing* them at the logic's object level as equational or rewrite theories. In fact, as further explained in Section 4.4, this can be carried out in Maude by defining suitable meta-level theories extending the `META-LEVEL` module [6].

The previous paragraphs lead us to the main contributions of the present chapter. We answer question (a) and part of (c) in Section 4.3.1 by defining and proving correct at the metalevel a method to check if a theory is OS-compact and to decide satisfiability of formulas in such a theory. We answer question (b) and the other part of (c) in Section 4.3.2 by defining and proving correct at the meta-level a method to compute constructor unifiers and constructor variants.

Furthermore, for improved efficiency we also provide an optimized version of constructor variant and unifier generation in Section 4.3.3; and discuss also the method of *descent maps* in the sense of [25]—which can both increase efficiency and widen the scope of decidable theories—and some specific descent maps currently supported in Section 4.3.4. Finally, we answer question (d) by meta-representing both the auxiliary algorithms (proved correct in this section), and the main algorithm (already proved correct in [25, 36]) in Section 4.4.

### 4.3.1 OS-Compactness and Satisfiability

In this section we present a high-level description of the algorithms needed to check that an equational theory $(\Omega, B_\Omega)$ is OS-compact and check if a conjunction of $\Omega$-disequalities is satisfiable in $T_{\Omega/B_\Omega}$. Note that the conditions checked are *sufficient* to prove OS-compactness in many practical cases, but they do not cover every possibility.

As shown in Theorem 4.4, a sufficient condition for a theory $(\Omega, B_\Omega)$ to be OS-compact is for it to have a decomposition of the form $\mathcal{R}_\Omega = (\Omega, B_\Omega, \varnothing)$ where (i) $B_\Omega \subseteq ACCU$ and (ii) axioms $B_\Omega$ are subsort-polymorphic. Thus, a sufficient condition is to require: (1) $B_\Omega$ to be a set of subsort-polymorphic ACCU axioms, and (2) $\Omega$ to be a signature of *free constructors* modulo $B_\Omega$. Fortunately, both of these subgoals are quite simple to check. Goal (1) can be solved by iterating over each axiom and applying a case analysis against its structure. The Maude language used by our implementation implicitly lifts axioms to subsort-polymorphic versions so we can elide this particular check. Goal (2) can be solved by an application of propositional tree automata (PTA) modulo $ACCU$ axioms. In particular, if the rules $R$ in $\mathcal{R}$ are linear and unconditional, then constructor freeness modulo $B$ can be translated into a PTA emptiness problem; see [59] for further details.

The more challenging task still remains. Assuming that our theory $(\Omega, B_\Omega)$ meets the conditions of Theorem 4.4, we still need to decide if a conjunction of $\Omega$-disequalities is decidable. This corresponds to steps (4)-(5) in the high-level algorithm at the end of Section 4.2. We start with step (5) since it is simpler. At this stage, we have a conjunction of $\Omega$-disequalities where the sort of each variable is *infinite*. The formula will be satisfiable iff it is $B_\Omega$-consistent. *$B_\Omega$-consistency* of a conjunction of $\Omega$-disequalities $\bigwedge_{k\in K} u_k \neq v_k$ is easy to check: since $\Omega$ is a signature of free constructors modulo $B_\Omega$, we just need to check that $u_k \neq_{B_\Omega} v_k$ for each $k \in K$. Since $B_\Omega \subseteq ACCU$ and since equality modulo associativity, commutativity, and unit axioms is clearly decidable, this process will terminate.

To complete this subsection, we just need to provide a way to compute step (4) in the high-level algorithm. This reduces into two requirements: (a) given a sort $s$, we need to check if the set $T_{\Omega/B_\Omega,s}$ is finite or infinite; (b) if finite, we need a function that computes a unique representative $\mathit{rep}([t]_{B_\Omega})$ for each equivalence class of terms $[t]_{B_\Omega} \in T_{\Omega/B_\Omega,s}$. These two requirements are addressed in the two subsections below. For ease of presentation, we start by addressing (b) since its solution is used to solve (a). But first, we need a lemma and a few definitions.

**Lemma 4.1** *(Non-emptiness Checking). Given $\Omega = ((S, \leqslant), F)$, we define $\Omega_M = S \uplus \{*\} \uplus \{\_,\_\}$ an unsorted signature. Then (a) we can deterministically construct a rewrite theory $\mathcal{R}_M$ (see full definition in Appendix C.1) over signature $\Omega_M$ such that $(\forall s \in S)\ T_{\Omega,s} \neq \varnothing \Leftrightarrow \mathcal{R}_M \vdash s \to^+ *$ (b) checking $R_M \vdash s \to^+ *$ is* decidable *(c) given axiom set $B$, $T_{\Omega/B,s} \neq \varnothing$ iff $T_{\Omega,s} \neq \varnothing$.*

**Proof 4.1** *See Appendix C.1.*

**Definition 4.7** *(Bisimilarity, Local Equitermination). Given binary relations $R_1 \subseteq S_1 \times S_1$, and $R_2 \subseteq S_2 \times S_2$, we write $R_1 \cong R_2$ iff $R_1$ and $R_2$ are* bisimilar. *Given $S \subseteq S_1 \cap S_2$, we say $R_1$ and $R_2$ are* locally equiterminating *over $S$ and write $R_1 \xleftrightarrow{S} R_2$ iff for all $s \in S$, $(R_1, s)$ terminates iff $(R_2, s)$ terminates where, by definition, $(R, s)$* terminates *iff there is no infinite $R$-path starting from $s$.*

By Lemma 4.1, if $S \uplus F$ is finite, all sets defined in Definition 4.8 are computable.

**Definition 4.8** *(Restrictions, Sort Signature). Given signature $\Omega = ((S, \leqslant), F)$ the* non-empty restriction *of a set of sorts/operators and a signature are given by $S_{\supset\varnothing} = \{s \in S \mid T_{\Omega/B,s} \neq \varnothing\}$, $F_{\supset\varnothing} = \{f : s_1 \cdots s_n \to s \in F \mid \{s_1, \ldots, s_n\} \subseteq S_{\supset\varnothing}\}$, and $\Omega_{\supset\varnothing} = ((S_{\supset\varnothing}, <|_{S_{\supset\varnothing}}), F_{\supset\varnothing})$ respectively. Given $F \subseteq F'$, the* operator restriction *of a signature is defined by $\Omega|_{F'} = ((S, <), F')$. The* sort signature *$S_\Omega$ is defined by $S_\Omega = ((S, <), \{s : \to [s] \mid s \in S\})$.*

Finite Sort Representative Generation.

Here we require a method that can compute a unique representative $\mathit{rep}([t]_{B_\Omega})$ for each $[t]_{B_\Omega} \in T_{\Omega/B_\Omega,s}$ whenever $|T_{\Omega/B_\Omega,s}| < \aleph_0$. Our general strategy is to show how an $\Omega$-term parser can be seen as a certain kind of ground rewrite theory over the signature $\Omega$, and thus, term generation is simply the inverse theory obtained by reversing the direction of the rewrite rules. Then, by applying the term generation rewrite theory modulo axioms $B_\Omega$, we can obtain the unique representative $\mathit{rep}([t]_{B_\Omega})$.

Recall any order-sorted signature $\Omega$ can be viewed as a tree automaton such that the tree automaton accepts a term $t$ in final state $s$ iff $t \in T_{\Omega,s}$ [17]. It has long been known that tree automata are very simple *ground* rewrite theories [60]. Since (i) tree automata recognizable languages are a strict subset of all context-free languages and (ii) context-free grammar production rules can easily be seen as rewrite rules, this result is not surprising. Definition 4.9 shows how to construct a ground rewrite theory which parses $\Omega$-terms from order-sorted signature $\Omega$.

**Definition 4.9** *$\mathcal{R}_P(\Omega) = (\hat{\Omega}_{\supset\varnothing} \uplus S_\Omega, \varnothing, R_P)$ where $R_P$ is the smallest rewrite relation $R_P = R_{P,S} \uplus R_{P,NC} \uplus R_{P,C}$ such that:*

*(a) $s \to s' \in R_{P,S}$ if $s < s'$*

*(b)* $f(s_1, \cdots, s_k) \to s \in R_{P,NC}$ *if* $f : s_1 \cdots s_k \to s \in F_{\supset\varnothing} \wedge k \geqslant 1$

*(c)* $c \to s \in R_{P,C}$ *if* $c : \to s \in F_{\supset\varnothing}$

Note that, even though $\Omega_{\supset\varnothing} \subseteq \Omega$, we do not lose completeness for parsing, since any sort in $s \in S/S_{\supset\varnothing}$ necessarily satisfies $T_{\Omega,s} = \varnothing$. Furthermore, it is straightforward to show that $\Omega \uplus S_\Omega$ is sensible and preregular iff $\Omega$ is sensible and preregular and $(\forall s \in S_{\supset\varnothing})\ t \in T_{\Omega,s} \Leftrightarrow t \to^{+}_{R_P} s$.

We now turn to term generation.

**Definition 4.10** *Let* $\mathcal{R}_G(\Omega) = (\hat{\Omega}_{\supset\varnothing} \uplus S_\Omega, \varnothing, R_G)$ *with* $R_G = R_P^{-1}$. *Since* $R_P = R_{P,S} \uplus R_{P,NC} \uplus R_{P,C}$ *we will use the following notation* $R_{G,S} = R^{-1}_{P,S}$, $R_{G,NC} = R^{-1}_{P,NC}$, *and* $R_{G,C} = R^{-1}_{P,C}$.

Again, by only considering $\Omega_{\supset\varnothing} \subseteq \Omega$, we do not lose completeness for term generation. We immediately obtain the following corollary.

**Corollary 4.1** $(\forall s \in S_{\supset\varnothing})\ t \in T_\Omega \Leftrightarrow s \to^{!}_{R_G} t$

Furthermore, if $|T_{\Omega,s}| < \aleph_0$ and $\Omega$ has no empty sorts, this process will always terminate. Note that we can apply the rules $R_G$ modulo $B_\Omega$. Then the set $\mathrm{Rep}(T_{\Omega/B_\Omega,s}) = \{\mathrm{rep}([t]) \mid [t] \in T_{\Omega/B_\Omega,s}\}$ is exactly the set $\mathrm{Rep}(T_{\Omega/B_\Omega,s}) = \{t \mid s \to^{!}_{R_G,B_\Omega} t\}$.

**Example 4.5** *(Integers with Addition, Term Generation). Consider the theory* $\mathcal{Z}_+$ *defined in Example 4.3. We define the sort generation rewrite theory* $\mathcal{R}_G(\Omega)$. *Since* $\Omega$ *has no empty sorts,* $\Omega_{\supset}\varnothing = \Omega$. *We must also kind-complete* $\Omega$*; we add a fresh top sort sort called* $[\mathit{Int}]$ *and operators* $\_+\_ : [\mathit{Int}]\ [\mathit{Int}] \to [\mathit{Int}]$, $-\_ : [\mathit{Int}] \to [\mathit{Int}]$, *and constants* $0, 1$ *obtain new typings in* $[\mathit{Int}]$. *We also add fresh constants Nat, NzNat, NzNeg, and Int of sort* $[\mathit{Int}]$. *Our set of rewrite rules contains:*

$$\begin{array}{rlcrl}
\mathit{NzNat} & \to 1 & & & \\
\mathit{Nat} & \to 0 & & \mathit{Int} & \to \mathit{NzNeg} \\
\mathit{NzNat} & \to \mathit{NzNat} + \mathit{NzNat} & & \mathit{Int} & \to \mathit{Nat} \\
\mathit{Nat} & \to \mathit{Nat} + \mathit{Nat} & & \mathit{Nat} & \to \mathit{NzNat} \\
\mathit{NzNeg} & \to -(\mathit{NzNat}) & & &
\end{array}$$

Figure 4.3: Term Generation Rewrite Rules for $\mathcal{Z}_+$

*We did not include a production rule for the operator* $\_+\_ : \mathit{Int}\ \mathit{Int} \to \mathit{Int}$ *since that is not a constructor. To generate the term* $1+1$, *we could use the rewrite sequence* $\mathit{Nat} \to \mathit{NzNat} \to \mathit{NzNat} + \mathit{NzNat} \to 1 + \mathit{NzNat} \to 1 + 1$.

Finite Sort Classification.

We devise an algorithm to check if $|T_{\Omega/B_\Omega,s}| < \aleph_0$ in two phases: (1) we first show how to decide $|T_{\Omega,s}| < \aleph_0$ and (2) we then use this as a subroutine in an *approximate* algorithm to check $|T_{\Omega/B_\Omega,s}| < \aleph_0$ when $B_\Omega = ACCU$. In our implementation, if the approximate algorithm fails to classify some $s$ as either infinite or finite, the user must manually check sort finiteness and provide this information to the tool.

Note that using $\mathcal{R}_G$ we already trivially obtain a semi-decidable algorithm for (1): compute $S_{\supset\varnothing}$ via $\mathcal{R}_M$; if $s \in S_{\supset\varnothing}$, then return yes; otherwise compute $\{t \in T_{\Omega,s} \mid s \rightarrow^{!}_{R_G} t\}$; if the process terminates, then return yes. Of course, an efficient, decidable algorithm would be preferable. Nevertheless, $\mathcal{R}_G$ is not too far from our desired decidable solution.

To solve (1), our strategy is as follows: (i) give sufficient conditions so that termination of $\mathcal{R}_G$ corresponds to sort finiteness in $\Omega$, (ii) define a rewrite system $\mathcal{R}_F$ and give sufficient conditions to prove termination of $\mathcal{R}_F$, (iii) show $\mathcal{R}_F$ terminates if and only if $\mathcal{R}_G$ terminates, (iv) and finally, present a decidable rewriting-based algorithm characterize when $\mathcal{R}_F$ terminates. To ease the exposition, proofs of some lengthier and less interesting lemmas are in the Appendix.

**Lemma 4.2** *If $|S| + |F| < \aleph_0$ then $(\mathcal{R}_G, s)$ is non-terminating iff $|T_{\Omega,s}| = \aleph_0$*

**Proof 4.2** *See Appendix C.1.* □

**Definition 4.11** *Let $\mathcal{R}_F(\Omega) = (S_{\supset\varnothing}, \varnothing, R_F)$ where $R_F = R_{F,S} \cup R_{F,NC}$ is the smallest rewrite relation such that:*

*(a) $s' \rightarrow s \in R_{F,S}$ if $s < s'$*

*(b) $s' \rightarrow s \in R_{F,NC}$ if $f : s_1 \cdots s_n \rightarrow s' \in F_{\supset\varnothing} \wedge \{s\} \subseteq \{s_1, \cdots, s_n\}$*

Note that we only consider $S_{\supset\varnothing}$ and $F_{\supset\varnothing}$, because, implicitly, any sort $s \in S/S_{\supset\varnothing}$ trivially satisfies $|T_{\Omega,s}| < \aleph_0$ and any operator $f \in F/F_{\supset\varnothing}$ cannot contribute meaningfully to building a term $t \in T_{\Omega,s}$. Before we complete the main proof, we prove a lemma and add an additional definition.

**Lemma 4.3** *Given $|S_{\supset\varnothing}| < \aleph_0$ and $s \in S_{\supset\varnothing}$, then the following are equivalent:*

1. *$(\mathcal{R}_F, s)$ is non-terminating*
2. *$(\exists s' \in S_{\supset\varnothing})\ s \rightarrow^{*}_{R_F} s' \rightarrow^{+}_{R_F} s'$*
3. *there is an infinite $R_F$-rewrite path $s \rightarrow_{R_F} s_1 \rightarrow_{R_F} s_2 \cdots \rightarrow_{R_F} s_n \rightarrow_{R_F} \cdots$ and $s' \in S_{\supset\varnothing}$ occurring infinitely often in the sequence*

**Proof 4.3** *Obviously, (3) implies (2), since if $s'$ occurs infinitely often, we must have $s \rightarrow^{*}_{R_F} s' \rightarrow^{+}_{R_F} s'$. Also, (2) implies (1) since $s \rightarrow^{*}_{R_F} s' \rightarrow^{+}_{R_F} s' \rightarrow^{+}_{R_F} s' \rightarrow^{+}_{R_F} \cdots$ is a non-terminating sequence. Finally, (1) implies (3), since $|S_{\supset\varnothing}| \leqslant \aleph_0$, which forces some $s' \in S_{\supset\varnothing}$ to occur infinitely often in any infinite sequence.* □

**Example 4.6** *(Integers with Addition, Checking Finiteness). Consider the theory $Z_+$ defined in Example 4.3. We define the sort finiteness classification theory $\mathcal{R}_F(\Omega)$. Since $\Omega$ has no empty sorts, $S_{\supset}\varnothing = S$, which defines the states in our theory. We will be done when we add the rules $R_F$. For each subsort in the subsort relation, we add one rule. For each unique argument sort for non-constant operators, we also add one rule. Thus, obtain the two sets $R_{F,S}$ and $R_{F,NC}$ on the left and right side below respectively.*
*At this point, we see that for each sort $s \in S$, $(\mathcal{R}_F(\Omega), s)$ reaches a cycle, which implies $(\mathcal{R}_F(\Omega), s)$ is non-terminating. However, in the presence of axioms $B$, we cannot yet decide whether any set $T_{\Omega/B,s}$ is finite or infinite.*

$$\begin{array}{ccc} \mathit{Int} \to \mathit{NzNeg} & \qquad & \mathit{NzNat} \to \mathit{NzNat} \\ \mathit{Int} \to \mathit{Nat} & & \mathit{Nat} \to \mathit{Nat} \\ \mathit{Nat} \to \mathit{NzNat} & & \mathit{NzNeg} \to \mathit{NzNat} \end{array}$$

Figure 4.4: Finiteness Checking Rewrite Rules for $\mathcal{Z}_+$

**Definition 4.12** *Let $\Omega = ((S,<), NC \uplus C)$ have non-constants and constants $NC$ and $C$ respectively. Define $\mathcal{R}^{\star}_G(\Omega) = (\Omega_{\supset\varnothing}|_{NC} \uplus S_\Omega, \varnothing, R_{G,\star})$ such that $R_{G,\star} = R_{G,S} \uplus R_{G,NC}$.*

Observe that $\mathcal{R}^{\star}_G$ is identical to $\mathcal{R}_G$ except that $\mathcal{R}^{\star}_G$ contains neither constants nor rewrite rules over constants. Now we are ready to prove the main theorem, namely, the local equitermination of $\mathcal{R}^F$ and $\mathcal{R}_G$ over $S_{\supset\varnothing}$.

**Theorem 4.5** $\mathcal{R}^F \xleftrightarrow{S_{\supset\varnothing}} \mathcal{R}_G$

**Proof 4.4** *Our proof proceeds in two parts: (a) $\mathcal{R}_F \cong \mathcal{R}^{\star}_G$ and (b) $\mathcal{R}^{\star}_G \xleftrightarrow{S_{\supset\varnothing}} \mathcal{R}_G$. This is sufficient since bisimilarity of rewrite theories clearly preserves termination and, in particular, local equitermination.*

*We first prove the bisimilarity $\mathcal{R}_F \cong \mathcal{R}^{\star}_G$. Let $H \subseteq (S_{\supset\varnothing} \times T_{\Omega|_{NC}\uplus\hat{S}})$ be a relation where $(s,t) \in H$ iff $s \trianglelefteq t$. To prove $\mathcal{R}_F \cong \mathcal{R}^{\star}_G$, we show that given two arrows, we can find another two arrows to make the diagrams below commute.*

$$\begin{tikzcd} s \arrow[r, "R_F"'] \arrow[d, "H"'] & s' \arrow[d, dashed, "H"] \\ t \arrow[r, dashed, "R_{G,\star}"'] & t' \end{tikzcd} \qquad \begin{tikzcd} s \arrow[r, dashed, "R_F"'] \arrow[d, "H"'] & s' \arrow[d, dashed, "H"] \\ t \arrow[r, "R_{G,\star}"'] & t' \end{tikzcd} \tag{4.5}$$

*Suppose $s \trianglelefteq t$. If $(s,s') \in R_F$ then $(s,s') \in R_{F,S}$ or $(s,s') \in R_{F,NC}$. Assume $(s,s') \in R_{F,S}$. Then $s' < s$ in $\Omega_{\supset\varnothing}$. But then, by definition, $(s,s') \in R_{G,S}$. Thus, $t[s] \to_{R_{G,\star}} t[s']$ and $s' \trianglelefteq t[s']$, as required. Alternatively, assume $(s,s') \in R_{F,NC}$. Then $\exists f : s_1 \cdots s_n \to s' \in F_{\supset\varnothing}$ with $\{s\} \subseteq \{s_1,\ldots,s_n\}$. But then, by definition, $(s', f(s_1,\cdots,s_n)) \in R_{G,NC}$. Thus, $t[s] \to_{R^{NC}_G} t[f(s_1,\cdots,s_n)]$ and $s' \trianglelefteq t[f(s_1,\cdots,s_n)]$. Since we used only definitional equivalences, the other direction follows symmetrically.*

*To prove $\mathcal{R}^{\star}_G \xleftrightarrow{S_{\supset\varnothing}} \mathcal{R}_G$, given $s \in S_{\supset\varnothing}$, we must show $(\mathcal{R}^{\star}_G, s)$ terminates iff $(\mathcal{R}_G, s)$ terminates. To begin, note $R_G = R_{G,\star} \uplus R_{G,C}$. Thus, if $R_{G,\star}$ is non-terminating, $R_G$ must also be non-terminating. To see the other direction, note $R_{G,C}$ always terminates since each rule has the form $s \to c \in C$ and constants cannot be rewritten. We proceed by proving the contrapositive. Thus, assume $R_{G,\star}$ terminates. By Lemma 4.4 below, $s \to^n_{R_G} t$ iff $s \to^i_{R_{G,\star}} t' \to^j_{R_{G,C}} t$ with $n = i + j$. Since $R_{G,\star}$ and $R_{G,C}$ are terminating and finitely branching, there are maximum bounds on the size of $i$ and $j$, say, $i_{max}$ and $j_{max}$ respectively. But then any rewrite path $s \to^n_{R_G} t$ necessarily has $n \leqslant i_{max} + j_{max}$; thus $(R_G, s)$ is terminating.* □

**Lemma 4.4** $(\forall n \in \mathbb{N})\, s \to^n_{R_G} t \Leftrightarrow [(\exists i, j \in \mathbb{N})\, s \to^i_{R_{G,\star}} t' \to^j_{R_{G,C}} t \wedge n = i + j]$

**Proof 4.5** *See Appendix C.1.*

Thus, according to Lemmas 4.2 and 4.3 and Theorem 4.5, $(\mathcal{R}_F, s)$ will generate a rewrite path containing a cycle iff $|T_{\Omega,s}| = \aleph_0$. To complete the proof, for any $s \in S$, we just need to characterize when $(\exists s' \in$

$S_{\supset\varnothing}$) $s \to^*_{R_F} s' \to^+_{R_F} s'$ holds. Thus, define the set of *cycle sorts* by $cy(S_{\supset\varnothing}) = \{s \in S_{\supset\varnothing} \mid s \to^+_{R_F} s\}$. This set can be computed by search, since the sort set and rules are both finite. Then, we immediately obtain the following theorem characterizing infinite sorts:

**Theorem 4.6** $\forall s \in S_{\supset\varnothing}\ |T_{\Omega,s}| = \aleph_0$ *iff* $\bigvee_{s' \in cy(S_{\supset\varnothing})} R_F \vdash s \to^* s'$

**Proof 4.6** *By Lemmas 4.2 and 4.3 and Theorem 4.5, obtain* $|T_{\Omega,s}| = \aleph_0$ *iff the formula* $(\exists s' \in S_{\supset\varnothing})\ s \to^*_{R_F} s' \to^+_{R_F} s'$ *holds. But by definition, any* $s'$ *which satisfies the formula satisfies* $s' \in cy(S_{\supset\varnothing})$*, so reduce to* $\exists s' \in cy(S_{\supset\varnothing})\ s \to^*_{R_F} s'$*. Since* $S$ *is finite by assumption,* $cy(S_{\supset\varnothing})$ *is finite. So, reduce to* $\bigvee_{s' \in cy(S_{\supset\varnothing})} s \to^*_{R_F} s'$*, which holds iff* $\bigvee_{s' \in cy(S_{\supset\varnothing})} R_F \vdash s \to^* s'$ *holds, as required.* ▫

To solve goal (2) of this subsection, we use the algorithm from Theorem 4.6 as a subroutine in an *approximate* algorithm to check $|T_{\Omega/B_\Omega,s}| < \aleph_0$ when $B_\Omega = ACCU$. Since $T_{\Omega/B,s}$ is a set of $B$-equivalence classes $[t]$, each containing at least one $t' \in [t]$ with $t' \in T_{\Omega,s}$, if $|T_{\Omega,s}| < \aleph_0$, then $T_{\Omega/B,s} < \aleph_0$. Nevertheless, in general, it may be the case that $|T_{\Omega/B,s}| < \aleph_0$ but $|T_{\Omega,s}| = \aleph_0$.

**Example 4.7** *(Checking Finiteness Modulo B). Let* $\Omega = ((\{a,b\},\{(a,b)\}), 0 :\to a,\ 1 :\to b, \_+\_ : a\,a \to a, \_+\_ : b\,b \to b)$ *and* $B$ *contain a unit axiom for* $0$ *over* $(+)$*. Then* $|T_{\Omega,a}| = |T_{\Omega,b}| = \aleph_0$ *but* $|T_{\Omega/B,a}| = 1$ *and* $|T_{\Omega/B,b}| = \aleph_0$*.*

Under some conditions on $B$, our methods can also check finiteness of term equivalence classes $T_{\Omega/B,s}$. The two lemmas below define some easy cases.

**Lemma 4.5** *Suppose* $B$ *is a set of associativity and/or commutativity axioms,* $|\Omega| < \aleph_0$*, and that* $\Omega$ *is* $B$*-preregular. Then* $|T_{\Omega/B,s}| < \aleph_0$ *iff* $|T_{\Omega,s}| < \aleph_0$*.*

**Proof 4.7** *Since* $\Omega$ *is* $B$*-preregular, all axioms in* $B$ *are sort preserving. Then obtain* $[u]_B \in T_{\Omega/AC,s}$ *iff* $[u]_B \subseteq T_{\Omega,s}$*, proving* ($\Leftarrow$)*. To show* ($\Rightarrow$)*, note that for any combination of associativity and/or commutativity axioms,* $[u]_B$ *is a* finite *set. Since* $T_{\Omega/B,s}$ *is finite, then* $T_{\Omega,s}$ *is a finite union of finite sets and thus finite.* ▫

Let set of unit axioms $U$ have unit elements $e_1 :\to s_1, \cdots e_n :\to s_n$ in $\Omega$. Then define $\Omega - U = \Omega - \{e_1 :\to s_1, \cdots e_n :\to s_n\}$.

**Lemma 4.6** *Let* $B_0$ *be a set of associative and/or commutative axioms and* $U$ *a set of unit axioms in* $\Omega$*,* $B = B_0 \uplus U$*,* $|\Omega| < \aleph_0$*, and* $\Omega = ((S,<),F)$ *be* $B$*-preregular. If* $|T_{\Omega-U,s}| = \aleph_0$*, then* $|T_{\Omega/B,s}| = \aleph_0$*.*

**Proof 4.8** *We can orient a unit axiom* $f(x,e) = x$ *as a rewrite rule* $f(x,e) \to x$*, so that the set* $U$ *becomes a set of rewrite rules* $R(U)$*. In this way the theory* $(\Omega, B_0 \uplus U)$ *can be decomposed as a convergent rewrite theory* $(\Omega, B_0, R(U))$*. Observe* $T_{\Omega-U/B_0} \subseteq C_{\mathcal{R}_U}$ *and* $C_{\mathcal{R}_U} \cong T_{\Omega/B}$*. By Lemma 4.5,* $|T_{\Omega-U,s}| = \aleph_0$ *iff* $|T_{\Omega-U/B_0,s}| = \aleph_0$*. Thus,* $\aleph_0 = |T_{\Omega-U,s}| = |T_{\Omega-U/B_0,s}| \leqslant |C_{\mathcal{R}_B,s}| = |T_{\Omega/B,s}|$*. Since* $|T_{\Omega/B,s}| \leqslant \aleph_0$*, obtain* $|T_{\Omega/B,s}| = \aleph_0$*, as required.* ▫

**Example 4.8** *(Integers with Addition, Checking Finiteness Modulo B). Consider the theory* $Z_+$ *defined in Example 4.3. By applying Lemmas 4.6 and then 4.5 above, we conclude for each sort* $s \in S$*, the term set* $T_{\Omega,s}$ *is infinite. To see this, note when the identity axiom/element is removed, the set of terms can be shown infinite by applying Lemma 4.5 and observing for each* $s \in S$*,* $(R_F(\Omega), s)$ *can reach a cycle.*

### 4.3.2 Constructor Variants and Constructor Unifiers

We first show how to compute a *complete* set of constructor variants $[\![t]\!]^{\Omega,W}_{R,B}$ of a term $t$ (see Definition 4.2) and then show how to use this method to compute a complete set constructor variant unifiers $\mathit{VarUnif}^{\Omega,W}_{B}(\phi)$. Recall that a constructor variant is just an variant $(t, \theta)$ such that $t \in T_{\Omega}(X)$. Thus, $[\![t]\!]^{\Omega,W}_{R,B}$ can be computed in two steps: (1) computing a complete set of variants $[\![t]\!]^{W}_{R,B}$ (2) for each variant $(t', \theta) \in [\![t]\!]^{W}_{R,B}$, compute a complete set of its constructor instances, i.e. a set of instances "away from $W$" $\mathit{Inst}^{\Omega,W}_{B}(t') = \{t'\eta_1, \cdots, t'\eta_n\}$ where for any other instance $t'\alpha \in T_{\Omega}(X)$ with $\alpha \in [\mathit{var}(t') \to T_{\Omega}(X)]$ and $\mathit{ran}(\alpha) \cap W = \varnothing$, there exists two substitutions $\gamma$ and $\eta_i$ with $\alpha|_W =_B \eta_i\gamma|_W$. Note that (1) can be solved via folding variant narrowing, so we tackle (2) by a reduction to a $B$-unification problem via a signature transformation $\Sigma \mapsto \Sigma^c$. In this transformed signature, the instances $\mathit{Inst}^{\Omega,W}_{B}(t')$ correspond exactly to the solutions of a *single* $B$-unification problem.

The signature transformation $\Sigma \mapsto \Sigma^c$ splits into two steps: (i) we define an extension of the sort poset $(S, <)$ of $\Sigma$ and $\Omega$ in Definition 4.13 and (ii) use that in Definition 4.14 to define $\Sigma^c$ as well as several other intermediate signatures. Recall we assume $\Sigma$ (and thus $\Omega$) are finite; otherwise these transformations would not be effective.

#### Sensibility and Preregularity of $\Sigma^c$.

This subsection is dedicated to proving that the $\Sigma \mapsto \Sigma^c$ transformation preserves term sort information, sensibility, and preregularity modulo $B$. These three conditions together ensure that $\Sigma^c$ terms are (i) always well-typed and (ii) have a unique least typing modulo axioms $B$ that is less than or equal to its original typing in the sort poset of $\Sigma$. While these conditions are important in themselves as matter of understandability, they additionally enable us to have much more efficient $B$-unification algorithms, e.g. if we want to $B$-unify $t = x : s$, we need only check whether $t$'s least sort is a subsort of $s$. We complete our proof in three steps: (a) we provide sufficient conditions for sensibility and preregularity to be preserved, (b) we prove that sensibility and preregularity are preserved, and (c) we lift the proof in (b) to show that preregularity modulo $B$ is preserved.

**Definition 4.13** *A* constructor sort refinement *of $(S, <)$ is defined by the following: (a) a set $S^c = S \uplus S^{\downarrow}$ with $c : S \to S^{\downarrow}$ a bijection, (b) a relation $(<^c)$ the smallest strict order where: (i) $(\forall s, s' \in S)\, s < s' \Leftrightarrow [s <^c s' \wedge c(s) <^c c(s')]$ and (ii) $(\forall s \in S)\, c(s) <^c s$, and (c) functions $(^{\bullet}) : S^c \to S$ and $(_{\bullet}) : S^c \to S^{\downarrow}$ where:*

$$s^{\bullet} = \begin{cases} s & \text{if } s \in S \\ c^{-1}(s) & \text{otherwise} \end{cases} \qquad \text{and} \qquad s_{\bullet} = \begin{cases} s & \text{if } s \in S^{\downarrow} \\ c(s) & \text{otherwise.} \end{cases} \tag{4.6}$$

We let $(<^c)$ also ambiguously denote its extension to strings of sorts $(S^c)^*$. Also, note that $(<) \subseteq (<^c)$ by definition and functions $(^{\bullet})$ and that $(_{\bullet})$ have unique homomorphic extensions to free monoid homomorphisms denoted by: $(^{\bullet}) : (S^c)^* \to S^*$ and $(_{\bullet}) : (S^c)^* \to (S^{\downarrow})^*$. Likewise, $(^{\bullet})$ and $(_{\bullet})$ have unique extensions to powersets, $(^{\bullet}) : \mathcal{P}(S^c) \to \mathcal{P}(S)$ and $(_{\bullet}) : \mathcal{P}(S^c) \to \mathcal{P}(S^{\downarrow})$. Lastly, $(^{\bullet})|_{(S^{\downarrow})^*}$ and $(_{\bullet})|_{S^*}$ are bijective by definition and lift into poset and powerset isomorphisms.

**Definition 4.14** *Given $\Sigma = ((S, <), F)$ and $\Omega = ((S, <), F_{\Omega})$ where $\Omega \subseteq \Sigma$ and $(S^c, <^c, (^{\bullet}), (_{\bullet}))$ is a constructor sort refinement of $(S, <)$, define the sets $F^{\downarrow}_{\Omega} = \{f : w_{\bullet} \to s_{\bullet} \mid f : w \to s \in F_{\Omega}\}$, $X^{\downarrow} = \{X_s\}_{s \in S^{\downarrow}}$, and $X^c = X \uplus X^{\downarrow}$. Define the signatures below as follows:*

| Signature | Sort Poset | Operators | Variable Sorts |
|---|---|---|---|
| $\Omega(X)$ | $(S, <)$ | $F_{\Omega}$ | $S$ |
| $\Sigma(X)$ | $(S, <)$ | $F$ | $S$ |
| $\Sigma^{+}(X^{c})$ | $(S^{c}, <^{c})$ | $F$ | $S^{c}$ |
| $\Omega^{c}(X^{c})$ | $(S^{c}, <^{c})$ | $F_{\Omega} \uplus F_{\Omega}^{\downarrow}$ | $S^{c}$ |
| $\Sigma^{c}(X^{c})$ | $(S^{c}, <^{c})$ | $F \uplus F_{\Omega}^{\downarrow}$ | $S^{c}$ |
| $\Omega^{\downarrow}(X^{c})$ | $(S^{c}, <^{c})$ | $F_{\Omega}^{\downarrow}$ | $S^{c}$ |
| $\Omega^{\downarrow}(X^{\downarrow})$ | $(S^{c}, <^{c})$ | $F_{\Omega}^{\downarrow}$ | $S^{\downarrow}$ |
| $\Omega^{\downarrow}_{\bullet}(X^{\downarrow})$ | $(S^{\downarrow}, <^{c}\vert_{S^{\downarrow}})$ | $F_{\Omega}^{\downarrow}$ | $S^{\downarrow}$ |

Table 4.1: Constructor Sort Refinement Signatures Overview

*Call* $\Sigma^{c}(X^{c})$ *and* $\Omega^{\downarrow}(X^{c})$ *the* constructor sort refinements *of* $\Sigma$ *and* $\Omega$.

We can summarize Definition 4.14 in Figure 4.5 below where each arrow is a signature inclusion. The signature decorations are intended to be suggestive of the transformation: $\Sigma^{+}$ extends the subsort relation; $\Sigma^{c}$ copies each constructor; $\Omega^{\downarrow}$ shifts constructors below; and finally $\Omega^{\downarrow}_{\bullet}$ shifts constructors below and discards sorts $S$ by applying $(_{\bullet})$.

$$\begin{tikzcd}
\Sigma(X) \arrow[r, hook] & \Sigma^{+}(X^{c}) \arrow[r, hook] & \Sigma^{c}(X^{c}) & & & \\
\Omega(X) \arrow[u, hook] \arrow[rr, hook] & & \Omega^{c}(X^{c}) \arrow[u, hook] & \Omega^{\downarrow}(X^{c}) \arrow[l, hook'] & \Omega^{\downarrow}(X^{\downarrow}) \arrow[l, hook'] & \Omega^{\downarrow}_{\bullet}(X^{\downarrow}) \arrow[l, hook']
\end{tikzcd}$$

Figure 4.5: Constructor Sort Refinement Signatures Inclusion

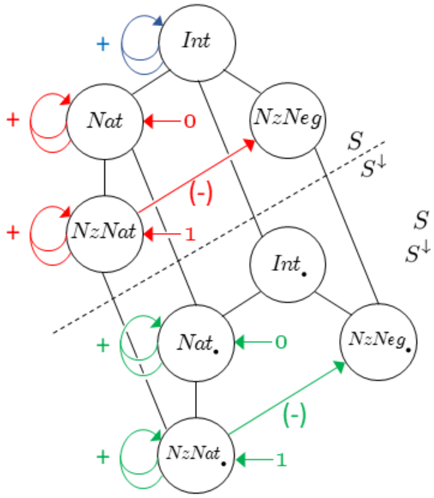
**Example 4.9** *(Integers with Addition, Constructor Sort Refinement). In the signature* Int *shown in Figure 4.6, the defined operators* $F - F_{\Omega}$ *are shown in blue, the constructors* $F_{\Omega}$ *are shown in red, and the* shifted *constructors modified by applying* $(_{\bullet})$ $F_{\Omega}^{\downarrow}$ *are shown in green. The sorts and subsort arrows shown above the dashed line define* $(S, <)$*; similarly, those below the dashed line define* $(S^{\downarrow}, <^{c}|_{S^{\downarrow}})$*. The union of both defines* $(S^{c}, <^{c})$. In the sequel, constructor sort refinement diagrams will use this same color-coding scheme.

**Remark 4.1** *Note that* $(^{\bullet})$ *and* $(_{\bullet})$ *naturally extend into signature morphisms. The sort mapping is either* $(^{\bullet})$ *or* $(_{\bullet})$*. For* $t \in T_{\Sigma^{c}}(X^{c})$*, the term mappings are:*

$$t^{\bullet} = \begin{cases} x : (s^{\bullet}) & \textit{if } t = x : s \in X^{c} \\ a & \textit{if } t = a \in T_{\Sigma^{c}} \\ f(t_1^{\bullet}, \cdots, t_n^{\bullet}) & \textit{if } t = f(t_1, \cdots, t_n) \in T_{\Sigma^{c}}(X^{c}) \end{cases} \tag{4.7}$$

$$t_{\bullet} = \begin{cases} x : (s_{\bullet}) & \textit{if } t = x : s \in X^{c} \\ a & \textit{if } t = a \in T_{\Sigma^{c}} \\ f(t_{1\bullet}, \cdots, t_{n\bullet}) & \textit{if } t = f(t_1, \cdots, t_n) \in T_{\Sigma^{c}}(X^{c}) \end{cases} \tag{4.8}$$

*Term mappings* $(^{\bullet})$ *and* $(_{\bullet})$ *naturally extend to substitutions* $\theta \in [X^{c} \to T_{\Sigma^{c}}(X^{c})]$ *such that, for each* $(x, t) \in \theta$, $(x^{\bullet}, t^{\bullet}) \in \theta^{\bullet}$ *and* $(x_{\bullet}, t_{\bullet}) \in \theta_{\bullet}$*. In particular, note (i)* $(_{\bullet}) : \Omega(X) \to \Omega^{\downarrow}_{\bullet}(X^{\downarrow})$ *is a signature*



| Operators | Color |
|---|---|
| $F - F_\Omega$ | Blue |
| $F_\Omega$ | Red |
| $F_\Omega^{\downarrow}$ | Green |

$S = \{NzNat, Nat, NzNeg, Int\}$
$S^{\downarrow} = \{NzNat_{\bullet}, Nat_{\bullet}, NzNeg_{\bullet}, Int_{\bullet}\}$

Figure 4.6: Overview of generated signatures for Int

*isomorphism with inverse* $(^{\bullet})$ *(ii)* $(^{\bullet}) : \Sigma^c(X^c) \to \Sigma(X)$ *is a signature morphism (iii) as sets of terms,* $T_{\Omega^{\downarrow}}(X^{\downarrow}) = T_{\Omega^{\downarrow}_{\bullet}}(X^{\downarrow})$ *and* $T_\Omega = T_{\Omega^{\downarrow}} = T_{\Omega^{\downarrow}_{\bullet}}$.

Our first goal in this subsection is to show that term sorting, sensibility, and preregularity are all preserved by constructor sort refinement, i.e., refinement in the sense that all existing sort information is preserved and only new sort information is added. Note that we trivially have preservation of term sorts by Remarks 4.1(i)-(iii) above since $(\forall s \in S^c\ \forall t \in T_{\Sigma^c}(X^c)_s)\, t^{\bullet} \in T_\Sigma(X)_{s^{\bullet}} \wedge s \leqslant^c s^{\bullet}$, the function $(^{\bullet})$ specializes to the identity when $t \in T_\Sigma(X)$, and thus we have $(\forall s \in S)\, t \in T_{\Omega^{\downarrow}_{\bullet}, s_{\bullet}} \Leftrightarrow t \in T_{\Omega,s}$. Thus, it is enough to prove preservation of sensibility and preregularity. However, the example below shows our current assumptions are too weak, i.e., the extended signature may not be preregular.

**Example 4.10** *(Preregularity Violations). Consider signature* $\Sigma_1$ *and its constructor subsignature* $\Omega_1$ *(resp. signature* $\Sigma_2$ *constructor subsignature* $\Omega_2$*) shown above the dashed line on the left (resp. right) side of Figure 4.7. While* $\Sigma_1$ *and* $\Omega_1$ *(resp.* $\Sigma_2$ *and* $\Omega_2$*) are preregular, constructor sort refinement* $\Sigma_1^c(X^c)$ *(resp.* $\Sigma_2^c(X^c)$*)* is not. *To witness this, consider the defined operator* $f$ *(in blue), the lowered constructor* $f$ *(in green), and a variable* $x$ *of sort* $A_{\bullet}$ *(shaded gray). The term* $f(x)$ *has typings* $A$ *and* $B_{\bullet}$ *(resp.* $A$ *and* $C_{\bullet}$*) using these operators, but neither sort is minimal.*

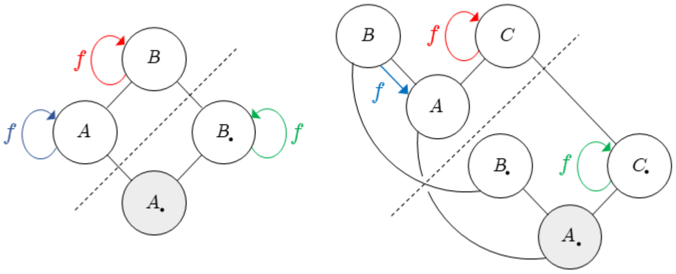


Figure 4.7: *Preregularity* violations for signatures (from left to right) $\Sigma_1/\Omega_1$ and $\Sigma_2/\Omega_2$

Note that in $\Sigma_1$ the violation occurred when a constructor $f$ had a subsort-overloaded defined operator $f$ below. In practice, this is a very bad idea, since defined symbols should be "evaluated away" while constructors should remain. However, $\Sigma_2$ shows that restricting subsort-overloading is not enough. In $\Sigma_2$, the defined symbol $f$ is not a subsort-overloaded version of the constructor $f$, and yet $\Sigma_2^c(X^c)$ still fails to be preregular. The invariant violated by both examples is that $\Omega_i$ is not *preregular below* $\Sigma_i$ for $i \in \{1,2\}$, in the sense that, in the signature $\Sigma_i$, when a term had a constructor typing, that typing was not *minimal.* For example, let $y$ be a variable of sort $A$. In $\Sigma_1$, $f(y)$ has a constructor typing in sort $B$, but the defined typing $A$ occurs below. Similarly, in $\Sigma_2$, $f(y)$ has a constructor typing $C$, but the defined typing $A$ occurs below. In order to formally specify our new invariant , we will need some additional notation.

Let $\Sigma = ((S,<),F)$ and $ty_\Sigma : T_\Sigma \to F$ be defined by $ty_\Sigma(c) = \{c : \to s \in F\}$ and $ty_\Sigma(f(t_1,\cdots,t_n)) = \{f : s_1 \cdots s_n \to s \in F \mid t_i \in T_{\Sigma_{s_i}}\}$. Additionally, let $ty_\Sigma : F \times S^* \to F$ denote the function $ty_\Sigma(f,w) = \{f : w' \to s \in F \mid w \leqslant w'\}$. Finally let $(P, \lhd)$ be an arbitrary poset and $\bigwedge I$ denote the greatest lower bound of $I$ in $(P,\lhd)$, if it exists. Then $min_\lhd : \mathcal{P}(P) \to P \uplus \{\varnothing\}$ is the function:

$$min_\lhd(I) = \begin{cases} \bigwedge I & \text{if } (\exists \bigwedge I) \wedge \bigwedge I \in I \\ \varnothing & \text{otherwise.} \end{cases} \tag{4.9}$$

**Definition 4.15** *(Preregular Below). Assume* $\Sigma = ((S,<),F)$ *has the subsignature* $\Omega = ((S,<),F_\Omega)$*.* $\Omega$ *is* preregular below $\Sigma$ *(written* $\Omega \prec \Sigma$*) iff* $\Omega$ *and* $\Sigma$ *are preregular and* $(\forall w \in S^*)\ ty_\Omega(f,w) \neq \varnothing \Rightarrow min_<(ty_\Sigma(f,w)) \in ty_\Omega(f,w)$ *where* $(F,<)$ *is the poset such that* $(f : w \to s) < (g : w' \to s') \Leftrightarrow s < s'$*.*

**Example 4.11** *(Integers with Addition, Preregularity Below). Recall the constructor sort refinements for theory* INT *shown in Figure 4.6. It is easy to check that* $\Omega \prec \Sigma$*; since the only overloaded operator is* $(+)$*, it is enough to consider all the typings of the term* $x + y$ *where variables* $x, y$ *have any sort* $S$ *and check, in each case, if a constructor typing exists, then the minimal typing is a constructor typing. For example, suppose* $x, y$ *both have sort NzNeg. Then there is no constructor typing, so the condition vacuously holds. If* $x, y$ *both have sort Nat, then* $x + y$ *has constructor typing Nat and defined typing Int, but Nat < Int.*

The syntactic notion of *preregular below* specified in Definition 4.15 is useful algorithmically because it is easily checkable for any finite signature $\Sigma$. However, there is an equivalent semantic, term-based notion[2] that is both easier to state and to use in formal proof. Lemma 4.7 shows how the syntactic notion implies the semantic notion (the other direction is left as an exercise for the reader).

**Lemma 4.7** *(Preregular Below, Semantic Version) Suppose that* $\Omega \prec \Sigma$*. Then* $\Omega$ *and* $\Sigma$ *are preregular and* $(\forall t \in T_\Sigma)\ t \in T_\Omega \Rightarrow ls_\Omega(t) = ls_\Sigma(t)$*.*

**Proof 4.9** *Follows by definition; see Appendix C.2.*

We now prove constructor sort refinements $\Omega^\downarrow(X^c)$ and $\Sigma^c(X^c)$ preserve sensibility and preregularity iff $\Omega$ and $\Sigma$ are sensible and $\Omega \prec \Sigma$. Note that, by definition, for any signature $\Sigma$, we have $ls_\Sigma(t) = min_<(ty_\Sigma(t))$ for the poset $(F,<)$ and to prove $\Sigma$ is preregular it is enough to show $(\forall t \in T_\Sigma)\, ls_\Sigma(t) \neq \varnothing$. We will need three lemmas. However, to preserve the logical flow of the argument, we state them here and give proofs in Appendix C.2. Lemma 4.8 states that only considering operators at the level of connected components preserves sensibility and is used to prove preservation of sensibility.

**Lemma 4.8** $\Sigma = ((S,<),F)$ *is sensible iff* $\widehat{\Sigma} = ((\widehat{S},\varnothing),\widehat{F})$ *is sensible where* $f : [s_1] \cdots [s_n] \to [s_0] \in \widehat{F}$ *iff* $\exists f : s'_1 \cdots s'_n \to s'_0 \in F$ *with* $s'_i \in [s_i]$ *for* $0 \leqslant i \leqslant n$*.*

[2] Preregularity as defined previously was also semantic, i.e. based on properties of $T_\Sigma$.

Lemmas 4.9 and 4.10 show how typings are preserved among various intermediate signatures and are used in proving preservation of preregularity.

**Lemma 4.9** $(\forall t \in T_{\Omega^{\downarrow}}(X^c)/X^c)\ ty_{\Omega^{\downarrow}(X^c)}(t) = ty_{\Omega^{\downarrow}(X^{\downarrow})}(t) = ty_{\Omega^{\downarrow}_{\bullet}(X^{\downarrow})}(t)$.

**Lemma 4.10** $(\forall t \in T_{\Sigma^c}(X^c)/X^c)\ ty_{\Sigma^+(X^c)}(t) = ty_{\Sigma(X)}(t^{\bullet})$.

We are now ready to prove that the constructor sort refinements preserve both preregularity sensibility:

**Theorem 4.7** *If* $\Omega < \Sigma$ *and* $\Omega$ *and* $\Sigma$ *are sensible, then the constructor sort refinements* $\Sigma^c(X^c) = ((S^c, <^c), F^c \uplus X^c)$ *and* $\Omega^{\downarrow}(X^c) = ((S^c, <^c), F^{\downarrow}_{\Omega} \uplus X^c)$ *are both sensible and preregular.*

**Proof 4.10** *Note that proving* $\Sigma^c$ *is sensible implies* $\Sigma^c(X^c)$ *is sensible, which implies* $\Omega^{\downarrow}(X^c)$ *is sensible. Then note that* $\widehat{\Sigma^c} \cong \widehat{\Sigma}$ *and signature isomorphism preserves sensibility, and finally apply Lemma 4.8.*

*We now prove that* $\Omega^{\downarrow}(X^c)$ *is preregular. By abuse of language, let* $X$ *also denote the signature* $((S, <), X)$. *Then note* $(\forall t \in T_{\Omega^{\downarrow}}(X^c))\ ty_{\Omega^{\downarrow}(X^c)}(t) = ty_{\Omega^{\downarrow}(X^{\downarrow})}(t) \uplus ty_X(t)$ *and* $\Omega^{\downarrow}(X^{\downarrow}) \cap X = \varnothing$. *Thus, by Lemma 4.9, we obtain that* $(\forall t \in T_{\Omega^{\downarrow}}(X^c)/X^c)\ ty_{\Omega^{\downarrow}(X^{\downarrow})}(t) = ty_{\Omega^{\downarrow}_{\bullet}(X^{\downarrow})}(t)$. *Thanks to the facts above,* $ls_{\Omega^{\downarrow}(X^c)} = ls_{\Omega^{\downarrow}_{\bullet}(X^{\downarrow})} \uplus ls_X$. *By signature isomorphism* $\Omega^{\downarrow}_{\bullet}(X^{\downarrow}) \cong \Omega(X)$, *this is equivalent to* $ls_{\Omega^{\downarrow}(X^c)} = (^{\bullet}); ls_{\Omega(X)}; (_{\bullet}) \uplus ls_X$, *where semicolon denotes in-order function composition. Since* $X$ *is preregular by definition and* $\Omega(X)$ *by assumption,* $ls_{\Omega^{\downarrow}(X^c)}$ *satisfies* $(\forall t \in T_{\Omega^{\downarrow}}(X^c))\ ls_{\Omega^{\downarrow}(X^c)}(t) \neq \varnothing$, *as required.*

*We now prove* $\Sigma^c(X^c)$ *is preregular. First let* $t \in X^c$. *Then* $t \in X \uplus X^{\downarrow}$. *If* $t = x : s \in X$ *then* $ls_{\Sigma^c(X^c)}(x{:}s) = ls_{\Sigma(X)}(x{:}s^{\bullet}) = s$. *Similarly, if* $t = x : s \in X^{\downarrow}$, $ls_{\Sigma^c(X^c)}(x{:}s) = ls_{\Omega(X)}(x{:}s^{\bullet})_{\bullet} = s$.

*Now let* $t \in T_{\Sigma^c}(X^c)/X^c$. *Note* $ty_{\Sigma^c(X^c)}(t) = ty_{\Omega^{\downarrow}(X^c)}(t) \uplus ty_{\Sigma^+(X^c)}(t)$, *i.e., the type of non-variable* $t$ *is from* $F^{\downarrow}_{\Omega}$ *or* $F$ *and* $ls_{\Sigma^c(X^c)}(t) = min_<(ty_{\Omega^{\downarrow}(X^c)}(t) \uplus ty_{\Sigma^+(X^c)}(t))$. *Suppose* $t \in T_{\Omega^{\downarrow}}(X^{\downarrow})/X^{\downarrow}$. *By Lemma 4.9 and* $\Omega^{\downarrow}_{\bullet}(X^{\downarrow}) \cong \Omega(X)$, *we obtain* $ty_{\Omega^{\downarrow}(X^c)}(t) = ty_{\Omega^{\downarrow}_{\bullet}(X^{\downarrow})}(t) = ty_{\Omega(X)}(t^{\bullet})_{\bullet}$. *By Lemmas 4.7 and 4.10, we can obtain the chain of equalities* $min_<(ty_{\Omega(X)}(t^{\bullet})) = min_<(ty_{\Sigma(X)}(t^{\bullet})) = min_<(ty_{\Sigma^+(X^c)}(t))$. *But then note that we have the chain of equalities* $ls_{\Sigma^c(X^c)}(t) = min_<(ty_{\Omega(X)}(t^{\bullet})_{\bullet} \uplus ty_{\Omega(X))}(t^{\bullet})) = ls_{\Omega(X)}(t^{\bullet})_{\bullet}$. *Finally, assume that* $t \in T_{\Sigma^c}(X^c)/T_{\Omega^{\downarrow}}(X^c)$. *Then we obtain* $ty_{\Omega^{\downarrow}(X^c)}(t) = \varnothing$ *and* $ls_{\Sigma^c(X^c)}(t) = min_<(ty_{\Sigma^+(X^c)}(t)) = min_<(ty_{\Sigma(X)}(t^{\bullet})) = ls_{\Sigma(X)}(t^{\bullet})$ *by Lemma 4.10. Thus, we have* $(\forall t \in T_{\Sigma^c}(X^c))\ ls_{\Sigma^c(X^c)}(t) \neq \varnothing$, *as required.*

□

**Corollary 4.2** *The functions* $ls_{\Omega^{\downarrow}(X^c)}$ *and* $ls_{\Sigma^c(X^c)}$ *are defined by:*

$$\forall t \in T_{\Sigma^c}(X^c)\ ls_{\Sigma^c(X^c)}(t) = \begin{cases} ls_{\Omega(X)}(t^{\bullet})_{\bullet} & \text{if } t \in T_{\Omega^{\downarrow}}(X^{\downarrow}) \\ ls_{\Sigma(X)}(t^{\bullet}) & \text{otherwise} \end{cases} \tag{4.10}$$

$$\forall t \in T_{\Omega^c}(X^c)\ ls_{\Omega^{\downarrow}(X^c)}(t) = \ ls_{\Sigma^c(X^c)}(t) \tag{4.11}$$

We now extend the above result to show that $B$-preregularity is preserved. The results below apply to all known of combinations of axioms that the Maude rewriting engine supports (which includes several state-of-the-art $B$-unification and $B$-matching algorithms) and infinitely many more.

**Remark 4.2** *Recall the kind-completion operator* $(\hat{\ })$ *we introduced in the preliminaries. Given a signature* $\Sigma = ((S, <), F)$, *the signature* $\widehat{\Sigma}$ *also contains, for each connected component of sorts* $[s] \in S/_<$, *a top sort* $\top_{[s]}$, *and for each operator* $f : s_1 \cdots s_n \to s$, *a top operator,* $f : \top_{[s_1]} \cdots \top_{[s_n]} \to \top_{[s]}$. *By an abuse of notation, we let* $(\hat{\ }) : \Sigma \to \widehat{\Sigma}$ *denote the signature morphism that maps each sort* $s$ *to* $\top_{[s]}$, *each*

*$f : s_1 \cdots s_n \to s$ to $f : \top_{[s_1]} \cdots \top_{[s_n]} \to \top_{[s]}$, and therefore each term $t(x_1 : s_1, \cdots, x_n : s_n) \in T_\Sigma(X)$ to $t(x_1 : \top_{[s_1]}, \cdots, x_n : \top_{[s_n]}) \in T_{\widehat{\Sigma}}(\widehat{X})$. This signature morphism extends to sets $E$ of $\Sigma$-equations, resp. sets of $\Sigma$-rewrite rules in the obvious sense: $E \ni (u = v) \mapsto (\hat{u} = \hat{v}) \in \widehat{E}$, resp. $R \ni (u \to v) \mapsto (\hat{u} \to \hat{v}) \in \widehat{R}$. Note that, by the subsort-polymorphic nature of the axioms $B$, as explained in Footnote 2, we have that $\Sigma$ is $B$-preregular if and only if $\widehat{\Sigma}$ is $\widehat{B}$-preregular, i.e. for any $u, v \in T_\Sigma(X)$, we have $u =_B v \Leftrightarrow u =_{\widehat{B}} v$.*

In this chapter, given theory $(\Sigma, E)$, we assume that it has a decomposition $(\Sigma, B, R)$ and that there exists a constructor decomposition $(\Omega, B_\Omega, R_\Omega)$ where $(\Sigma, B, R)$ protects $(\Omega, B_\Omega, R_\Omega)$. Recall the definition of decomposition requires that $\Sigma$ is $B$-preregular and $\Omega$ is $B_\Omega$-preregular. We impose a very weak requirement on such axioms which we define below:

**Definition 4.16** *Let $\Omega$ and $\Sigma$ be signatures such that $\Omega \subseteq \Sigma$ and let $\Sigma$ be $B$-preregular. By $B$-preregularity, the theory $(\Sigma, B)$ decomposes into $(\Sigma, B_0, R(B_1))$ where $B_0$ is sort-preserving and $R(B_1)$ is sort-decreasing. We say axioms $B$* respect constructors *iff for any $u = v \in B$ and sort specialization $\rho$, $u = v \in B_1 \Rightarrow [u\rho \in T_\Omega(X) \Rightarrow v\rho \in T_\Omega(X)]$ and $u = v \in B_0 \Rightarrow [u\rho \in T_\Omega(X) \Leftrightarrow v\rho \in T_\Omega(X)]$. In words, if an axiom can apply to a constructor term as an equation or as a rule, then the result is a constructor term.*

**Theorem 4.8** *Let (a) $(\Omega, B_\Omega) \subseteq (\Sigma, B)$ be a conservative theory extension, i.e. $(\forall u, v \in T_\Omega(X))\ u =_{B_\Omega} v \Leftrightarrow u =_B v$ (b) $\Sigma(X)$ be $B$-preregular and $\Omega(X)$ be $B_\Omega$-preregular, and (c) axioms $B$ respect constructors. Then $\Sigma^c(X^c)$ is $B$-preregular and $\Omega^\downarrow(X^c)$ is $B_\Omega$-preregular.*

**Proof 4.11** *By Remark 4.2 above and assumption (b), we know that signature $\widehat{\Sigma(X)}$ is $\widehat{B}$-preregular and $\widehat{\Omega(X)}$ is $\widehat{B_\Omega}$-preregular. Unpacking the definition of preregularity, observe decompositions $(\widehat{\Sigma(X)}, \widehat{B_0}, \widehat{R(B_1)})$ and $(\widehat{\Omega(X)}, \widehat{B_{\Omega_0}}, \widehat{R(B_{\Omega_1}})$ such that $\widehat{B} = \widehat{B_0} \uplus \widehat{B_1}$ and $\widehat{B_\Omega} = \widehat{B_{\Omega_0}} \uplus \widehat{R(B_{\Omega_1})}$ where $\widehat{R(B_1)}$ and $\widehat{R(B_{\Omega_1})}$ are sort-decreasing and $\widehat{B_0}$ and $\widehat{B_{\Omega_0}}$ are sort-preserving.*

*By Theorem 4.7, $\Sigma^c(X^c)$ and $\Omega^\downarrow(X^c)$ are preregular. By Remark 4.2 above, we must show that $\widehat{\Sigma^c(X^c)}$ is $\widehat{B}$-preregular and $\widehat{\Omega^\downarrow(X^c)}$ is $\widehat{B_\Omega}$-preregular. By definition of preregularity, this is equivalent to showing decompositions $(\widehat{\Sigma^c(X^c)}, \widehat{B_0}, \widehat{R(B_1)})$ and $(\widehat{\Omega^\downarrow(X^c)}, \widehat{B_{\Omega_0}}, \widehat{R(B_{\Omega_1}})$ exist and satisfy appropriate sort-decreasing and sort-preserving requirements. Since $\Sigma^c(X^c)$ and $\Omega^\downarrow(X^c)$ are sort refinements of $\Sigma(X)$ and $\Omega(X)$, the meaning of the kind-completion operator $(\hat{\ })$ is well-defined and identical in all signatures, so the above definition is unambiguous. Thus, we will be done if we can show $\widehat{R(B_1)}$ is sort-decreasing and $\widehat{B_0}$ is sort-preserving for $\Sigma^c(X^c)$ (since that implies the corresponding fact given $\widehat{R(B_{\Omega_1}})$ and $\widehat{B_{\Omega_0}}$ for $\Omega^\downarrow(X^c)$ holds by conservativity). Thus, we need only show $\widehat{R(B_1)}$ is sort-decreasing and $\widehat{B_0}$ is sort-preserving in $\Sigma^c(X^c)$ for each sort specialization $\rho$.*

*Recall that by Corollary 4.2 $(\forall t \in T_{\Sigma^c}(X^c))\ ls_{\Sigma^c(X^c)}(t) = ls_{\Sigma(X)}(t^\bullet)$ and that $(\forall t \in T_{\Omega^\downarrow}(X^c))\ ls_{\Omega^\downarrow(X^c)}(t) = ls_{\Omega(X)}(t^\bullet)$ whenever $t \notin T_{\Omega^\downarrow}(X^\downarrow)$. Thus, it is enough to consider, for sort specializations $\rho$ where $ran(\rho) \subseteq X^\downarrow$, whether axioms $\widehat{B_0}$ are sort-decreasing and rules $\widehat{R(B_1)}$ are sort-preserving. Letting $u = v \in \widehat{B_1}$, we first prove $\widehat{R(B_1)}$ is sort-decreasing by case analysis*

*(i) Let $u\rho \notin T_{\Omega^\downarrow}(X^\downarrow)$. By Corollary 4.2, $ls_{\Sigma^c(X^c)}(u\rho) = ls_{\Sigma(X)}(u\rho^\bullet)$. Since $\Sigma^c(X^c)$ is a sort refinement of $\Sigma(X)$, $ls_{\Sigma^c(X^c)}(v\rho) \leqslant ls_{\Sigma(X)}(v\rho^\bullet)$, as required.*

*(ii) Let $u\rho \in T_{\Omega^\downarrow}(X^\downarrow)$. Observe any sort specialization $\rho \in [X \to X_\downarrow]$ can be decomposed into $\rho = \alpha; (_\bullet)$ with substitution $\alpha \in [X \to X]$ and where the substitution $(_\bullet)$ lifts into signature isomorphism $(_\bullet) : \Omega(X) \to \Omega^\downarrow_\bullet(X^\downarrow)$. Then $u\alpha \in T_\Omega(X)$ and by assumption (c), $v\alpha \in T_\Omega(X)$. Furthermore, by assumption*

*(b), $ls_{\Omega(X)}(u\alpha) \geqslant ls_{\Omega(X)}(v\alpha)$ and additionally, $vars(u) = vars(v)$, i.e. axioms are always* regular. *By axiom regularity and signature isomorphism $(_\bullet)$, $ls_{\Omega^{\downarrow}_{\bullet}(X^{\downarrow})}(u\alpha; (_\bullet)) \geqslant ls_{\Omega^{\downarrow}_{\bullet}(X^{\downarrow})}(v\alpha; (_\bullet))$. Apply Lemma 4.9 (i.e. $(\forall t \in T_{\Omega^{\downarrow}}(X^c)/X^c)\ ty_{\Omega^{\downarrow}(X^c)}(t) = ty_{\Omega^{\downarrow}_{\bullet}(X^{\downarrow})}(t)$) and the definition of the least sort function to obtain $ls_{\Omega^{\downarrow}(X^c)}(u\alpha; (_\bullet)) \geqslant ls_{\Omega^{\downarrow}(X^c)}(v\alpha; (_\bullet))$.*

*Letting $u = v \in \widehat{B_0}$, we will be done if we prove axioms $\widehat{B_0}$ are sort-preserving. We proceed by cases:*

*(i) Let $u\rho \notin T_{\Omega^{\downarrow}}(X^{\downarrow})$. By Remark 4.1(iii), note $u\rho \notin T_{\Omega^{\downarrow}_{\bullet}}(X^{\downarrow})$. Then since signature isomorphism $(_\bullet) : \Omega(X) \to \Omega^{\downarrow}_{\bullet}(X^{\downarrow})$ has inverse $(^\bullet)$, $u\rho^\bullet \notin T_{\Omega}(X)$. By assumption (c), we know $v\rho^\bullet \notin T_{\Omega}(X)$. By Corollary 4.2, $ls_{\Sigma^c(X^c)}(u\rho) = ls_{\Sigma(X)}(u\rho^\bullet)$ and $ls_{\Sigma^c(X^c)}(v\rho) = ls_{\Sigma(X)}(v\rho^\bullet)$. Then, by assumption (b), observe $ls_{\Sigma(X)}(u\rho^\bullet) = ls_{\Sigma(X)}(v\rho^\bullet)$, as required.*

*(ii) Let $u\rho \in T_{\Omega^{\downarrow}}(X^{\downarrow})$. We must show that $ls_{\Sigma^c(X^c)}(u\rho) = ls_{\Sigma^c(X^c)}(v\rho)$. The argumentation proceeds as in case (ii) above by decomposing $\rho = \alpha; (_\bullet)$. Then $ls_{\Omega(X)}(u\alpha) = ls_{\Omega(X)}(v\alpha)$ holds by the sort-preserving requirement of $B$-preregularity, and finally we obtain $ls_{\Omega^{\downarrow}(X^c)}(u\alpha; (_\bullet)) = ls_{\Omega^{\downarrow}(X^c)}(v\alpha; (_\bullet))$ by regularity, signature isomorphism, Lemma 4.9, and the definition of the least sort function, completing the proof.*□

The following corollary lifts the result above to decompositions.

**Corollary 4.3** *Let rewrite theory $\mathcal{R} = (\Sigma, B, R)$ be convergent with constructor decomposition $\mathcal{R}_\Omega = (\Omega, B_\Omega, R_\Omega)$ and $\Omega < \Sigma$ and let axioms $B$ respect constructors. Then $\Sigma^c$ and $\Omega^{\downarrow}$ are sensible and $B$-preregular.*

**Proof 4.12** *Note that protecting a constructor decomposition implies that $(\Sigma, B)$ is a conservative extension of $(\Omega, B_\Omega)$. Then apply Theorem 4.8.*

Computing a Complete Set of Constructor Instances.

We have shown that, under mild conditions, the $\Sigma \mapsto \Sigma^c$ transformation preserves sensibility and $B$-preregularity. Thus, $B$-unification will be well-defined in our new signature. We now move to prove the main theorem of this section, which shows how $Inst_B^{\Omega,W}(t)$, a complete set of constructor instances of a term $t$ modulo $B$ may be obtained by solving a unification problem in $\Sigma^c(X^c)$. We first collect a number of essential facts relating $T_\Omega(X)$ to $T_{\Omega^{\downarrow}}(X^{\downarrow})$ that we use in the proof.

**Lemma 4.11** *Suppose that $\alpha, \beta \in [X \to T_\Omega(X)]$, $\alpha', \beta' \in [X^{\downarrow} \to T_{\Omega^{\downarrow}}(X^{\downarrow})]$, and $\theta, \gamma \in [X^c \to T_{\Sigma^c}(X^c)]$. Let $id^{\downarrow} \in [X^c \to X^{\downarrow}]$ where $id^{\downarrow}(x : s) = x : s_\bullet$. Then:*

*(a) $(\alpha_\bullet)^\bullet = \alpha \wedge (\alpha^\bullet)_\bullet = \alpha$*

*(b) $(\forall t, t' \in T_\Omega(X))\ t =_B t' \Leftrightarrow t_\bullet =_B t'_\bullet \ \wedge (\forall t, t' \in T_{\Omega^{\downarrow}}(X^{\downarrow}))\ t =_B t' \Leftrightarrow t^\bullet =_B t'^\bullet$*

*(c) $[\alpha =_B \beta \Leftrightarrow \alpha_\bullet =_B \beta_\bullet] \ \wedge \ [\alpha' =_B \beta' \Leftrightarrow \alpha'^\bullet =_B \beta'^\bullet]$*

*(d) $(\forall t \in T_{\Sigma^c}(X^c))\ t_\bullet = t(id^{\downarrow}) \wedge (id^{\downarrow})^\bullet = id$*

*(e) $(\forall t \in T_{\Sigma^c}(X^c))\ (t\theta)_\bullet = t_\bullet(\theta_\bullet) \wedge (t\theta)^\bullet = t^\bullet(\theta^\bullet) \wedge (\theta\gamma)_\bullet = \theta_\bullet(\gamma_\bullet) \wedge (\theta\gamma)^\bullet = \theta^\bullet(\gamma^\bullet)$*

**Proof 4.13** *Both (a) and (b) follow immediately since $T_{\Omega^{\downarrow}_{\bullet}}(X^{\downarrow}) = T_{\Omega^{\downarrow}}(X^{\downarrow})$ and by isomorphism $(^\bullet) : \Omega^{\downarrow}_{\bullet}(X^{\downarrow}) \to \Omega(X)$. Then (c) is an immediate application of (b). Finally, (d) and (e) have easy structural induction proofs.* □

We now give a precise construction of $Inst_B^{\Omega,W}$ using $B$-unification in $\Sigma^c(X^c)$.

**Theorem 4.9** *Suppose $\Sigma(X)$ and $\Omega(X)$ are sensible and $B$-preregular, $\Omega < \Sigma$, and $B$ respects constructors. Then (a) $\forall t \in T_\Sigma(X)_s \, \forall t' \in T_\Omega(X)_{s'}$ with $s \equiv_< s'$, $W = vars(t) \uplus \{x : c(s)\}$, $W_\bullet = vars(t)_\bullet \uplus \{x : c(s)\}$, $\alpha \in [vars(t) \to T_\Omega(X)]$, $ran(\alpha) \cap W = \varnothing$, $vars(t') \cap W = \varnothing$, and $x \notin vars(t)$, we have $t\alpha =_B t'$ iff $\exists \eta \in Unif_B^W(t = x : c(s'))$ and $\exists \theta \in [X \to T_\Omega(X)]$ such that $\eta^\bullet\theta|_W =_B \alpha$ and (b) the complete set of constructor instances of $t$ modulo $B$ is defined by $Inst_B^{\Omega,W}(t) = \{t(\eta^\bullet) \mid \eta \in Unif_B^W(t = x : ls_{\Sigma(X)}(t)_\bullet)\}$.*

**Proof 4.14** *We first prove (a). Let $\beta = \alpha_\bullet \uplus \{(x : s'_\bullet, t'_\bullet)\}$. Then observe:*

$$\begin{aligned} t\alpha =_B t' &\Leftrightarrow (t\alpha)_\bullet =_B t'_\bullet && (4.12)\\ &\Leftrightarrow t_\bullet(\alpha_\bullet) =_B t'_\bullet && (4.13)\\ &\Leftrightarrow t_\bullet\beta =_B x : s'_\bullet\beta && (4.14)\\ &\Leftrightarrow \exists \eta' \in Unif_B^{W_\bullet}(t_\bullet = x : s'_\bullet) \; \exists \theta' \in [X^\downarrow \to T_{\Omega^\downarrow}(X^\downarrow)] \; \eta'\theta'|_{W_\bullet} =_B \beta && (4.15)\end{aligned}$$

*which follow by Lemma 4.11 and because $B$ respects constructors so $t\alpha \in T_\Omega(X)$. Let id be the identity substitution. Observe that $x : (s'_\bullet)_\bullet = x : s'_\bullet$. Then by Lemma 4.11, obtain:*

$$\begin{aligned} \eta' \in Unif_B^{W_\bullet}(t_\bullet = x : s'_\bullet) &\Leftrightarrow \eta' \in Unif_B^{W_\bullet}(t_\bullet = x : (s'_\bullet)_\bullet) && (4.16)\\ &\Leftrightarrow \eta' \in Unif_B^{W_\bullet}(t(id^\downarrow) = x : s'_\bullet(id^\downarrow)) && (4.17)\\ &\Leftrightarrow id^\downarrow\eta' \in Unif_B^W(t = x : s'_\bullet) && (4.18)\end{aligned}$$

*as well as:*

$$\begin{aligned} \eta'\theta'|_{W_\bullet} =_B \beta &\Leftrightarrow (\eta'\theta)^\bullet|_W =_B \beta^\bullet && (4.19)\\ &\Leftrightarrow \eta'^\bullet(\theta'^\bullet)|_W =_B \beta^\bullet && (4.20)\\ &\Leftrightarrow \eta'^\bullet(\theta'^\bullet)|_{vars(t)} =_B \alpha \;\wedge \eta'^\bullet(\theta'^\bullet)(x) =_B t'. && (4.21)\end{aligned}$$

*Now let $\eta = id^\downarrow\eta'$ and $\theta = \theta'^\bullet$. Then we can derive equalities $\eta^\bullet\theta = (id^\downarrow\eta')^\bullet\theta = (id^\downarrow)^\bullet(\eta'^\bullet)\theta = id(\eta'^\bullet)\theta = \eta'^\bullet(\theta'^\bullet)$ as required. Finally (b) is an immediate application of (a).* □

Computing Constructor Variants and Constructor Variant Unifiers.

Using Theorem 4.9, we show in Corollary 4.4 below how to compute constructor variants. We then apply this result to terms in the signature $\Sigma^\wedge$ to see how to also compute constructor variant unifiers.

**Corollary 4.4** *Let $(\Sigma, B, R)$ be convergent and protect constructor decomposition $(\Omega, B_\Omega, R_\Omega)$ where $\Omega < \Sigma$ and $B$ respects constructors. Given any $t \in T_\Sigma(X)$, we have:*

$$[\![t]\!]_{R,B}^{\Omega,W} = \{(t'(\eta^\bullet), (\theta\eta^\bullet)|_{vars(t)}) \in [\![t]\!]_{R,B}^{*} \mid (t', \theta) \in [\![t]\!]_{R,B}^{W,x} \wedge \eta \in Unif_B^{W,x,t'}(t' = x : ls_{\Sigma(X)}(t')_\bullet)\} \qquad (4.22)$$

*is a complete set of most general constructor variants of $t$.*

**Proof 4.15** *Apply Corollary 4.3. Note that the requirement $(t'(\eta^\bullet), \theta\eta^\bullet) \in [\![t]\!]_{R,B}^{*}$ in the above definition of $[\![t]\!]_{R,B}^{\Omega,W}$ is equivalent to requiring $t'(\eta^\bullet) = t'(\eta^\bullet)!_{R,B} \wedge (\theta\eta^\bullet)|_{vars(t)} = (\theta\eta^\bullet)|_{vars(t)}!_{R,B}$. Note also that, by Theorem 4.9, $[\![t]\!]_{R,B}^{\Omega,W}$ is by construction a set of constructor variants of $t$. So we just need to show that it is* *complete. Let $(u,\gamma)$ be a constructor variant of $t$. Then, by completeness of $[\![t]\!]^{W,x}_{R,B}$ there is a $(t',\theta) \in [\![t]\!]^{W,x}_{R,B}$ and a substitution $\rho$ such that (i) $(\theta\rho)|_{vars(t)} =_B \gamma$, and (ii) $t'\rho =_B u$. But, since $u \in T_\Omega(X)$, by Theorem 4.9 there is substitution $\eta \in Unif^{W,x,t'}_B(t' = x : ls_{\Sigma(X)}(t')_\bullet)$ and a substitution $\delta$ such that $\eta^\bullet\delta|_W =_B \rho$, proving that $(t'(\eta^\bullet), (\theta\eta^\bullet)|_{vars(t)}) \sqsupseteq_B (u,\gamma)$, as desired.* ▫

Let us now see how, by applying Theorem 4.9, computing a complete set of constructor variant unifiers becomes simple. Recall that any unification problem $\phi$ is a $\Sigma^\wedge$-term in $T_{\Sigma^\wedge}(X)_{Conj}$. Then a complete set of constructor variant unifiers is defined by:

$$VarUnif^{\Omega,W}_E(\phi) = \{\alpha(\eta^\bullet) \mid \alpha \in VarUnif^W_E(\phi) \wedge \eta \in Unif^{W,\phi\alpha}_B((\phi\alpha)!_{R,B} = x : Conj_\bullet)\}. \tag{4.23}$$

**Example 4.12** *(Integers with Addition, Constructor Variants/Unifiers). Recall the constructor variants and unifiers for the theory $\mathcal{Z}_+$ we examined in Example 4.4. We can easily check that the ACU axioms for $\mathcal{Z}_+$ respect constructors. Consider again the term $x + y$ with $x, y$ variables of sort Int that has a complete set of twelve variants. Its most simple variant is $u = (x + y, id)$ where id is the identity substitution. Recall $u$ is* not *a constructor variant in $\mathcal{Z}_+$, because $\_+\_ : Int\ Int \to Int$ is a defined symbol; however, $(+)$ has typings which* are *constructors. We apply the method shown in Corollary 4.4 to compute a complete set of constructor variants* less general *than $u$, by solving the $B$-unification problem $Unif^{W,x,y,z}_B(y + z = x : ls_{\Sigma(X)}(y + z)_\bullet) = Unif^{W,x,y,z}_B(y + z = x : Int_\bullet)$ in the signature $\Sigma^c(X^c)$. Thus, we obtain the set: (i) $(y', \{y \mapsto y', z \mapsto 0\})$, (ii) $(z', \{z \mapsto z', y \mapsto 0\})$, and (iii) $(y' + z', \{y \mapsto y' : Nat, z \mapsto z' : Nat\})$. Likewise, let $\phi$ be the equation $w = y + z$, with $w, y, z$ of sort Int. Then $\{w \mapsto y + z\}$ is a trivial $\mathcal{Z}_+$-unifier of $\phi$, but* not *a constructor unifier. We can compute a complete set of constructor unifiers less general than $\{w \mapsto y + z\}$ by computing the $B$-unification problem $Unif^{W,x,y,z}_B((w = y + z) = x : Conj_\bullet)$ in the signature $(\Sigma^\wedge)^c(X^c)$. Thus, we obtain (i) $\{w \mapsto y, z \mapsto 0\}$, (ii) $\{w \mapsto z, y \mapsto 0\}$, and (iii) $\{w \mapsto y' + z', y \mapsto y' : Nat, z \mapsto z' : Nat\}$.*

### 4.3.3 Optimizing Constructor Variant and Unifier Generation

Constructor variant and constructor unifier generation can be substantially optimized for two reasons: (i) a variant $(t',\theta) \in [\![t]\!]^{W,x}_{R,B}$ may already be a constructor variant, so in such case there is no need for an additional unification step; and (ii) some variants $(t',\theta) \in [\![t]\!]^{W,x}_{R,B}$ may *never* have constructor instances, and this can be syntactically detected, so in that case we need not waste time failing in unification efforts. To better characterize those cases where there is no hope for a variant $(t',\theta)$ to have any constructor instances we make further assumptions about the regular and linear axioms $B = B_0 \uplus B_1$ which apply in many cases of interest and in particular in any combinations of $A$, $C$ and $U$ axioms. We assume that $B$ can be decomposed as a union $B = \bigcup_{f \in \Sigma} B_f$ where $B_f = B_{f,0} \uplus B_{f,1}$, and where, as required, the axioms in $B_{f,0}$ are sort-preserving and the rules in $R(B_{f,1})$ are sort decreasing and furthermore: (i) if $(u = v) \in B_{f,0}$ both $u$ and $v$ are not variables and $f$ is the only function symbol appearing anywhere in $u$ and $v$, and (ii) if $(u \to v) \in R(B_{f,1})$, then $v$ is a variable $x$ and $u$ is of the form $u = f(u_1, \ldots, u_n)$ where, besides $x$, the only other symbols appearing in $u$ are $f$ and possibly some constants in a set of constants $C_f$.

Instead of characterizing the set of terms $t'$ that cannot have a constructor term as an instance, let us characterize those that *might* do so. They are exactly the $\Sigma_1$-terms where $\Sigma_1 = \widetilde{\Omega} \uplus \{f \in (\Sigma - \widetilde{\Omega}) \uplus C_f \mid B_{f,1} \neq \varnothing\}$, where $\widetilde{\Omega}$ includes all operators in the same subsort-overloaded family of $\Omega$, i.e., we add to $\Omega$ its non-constructor typings if any. That this choice of $\Sigma_1$ is correct follows from the following lemma:

**Lemma 4.12** *Let $t \in T_{\Sigma}(X) - T_{\Sigma_1}(X)$. Then there is no $u \in T_{\Omega}(X)$ and substitution $\sigma$ such that $t\sigma =_B u$.*

**Proof 4.16** *Suppose $t \in T_{\Sigma}(X) - T_{\Sigma_1}(X)$ and let $u \in T_{\Omega}(X)$ and $\sigma$ be such that $t\sigma =_B u$. This means that $(t\sigma)!_{R(B_1),B_0} =_{B_0} u!_{R(B_1),B_0}$. Since $u!_{R(B_1),B_0}$ is a constructor term and the regular axioms in $B_0$ are symbol- and sort-preserving, this means that all symbols in $(t\sigma)!_{R(B_1),B_0}$ must be in $\Omega$. But this is impossible, since: (i) there must be a $g \in \Sigma - \Sigma_1$ occurring in $t$ and therefore in $t\sigma$, and (ii) such a $g$ cannot disappear either by application of symbol-preserving axioms in $B_0$ nor by application of symbol-erasing rules in some $(u \to v) \in R(B_{f,1})$ for some $f \in \Sigma_1$, since the* only *symbols any such rule can erase form a term after rewriting it are exactly $f$ and some constants in $C_f$, none of which symbols in $\Sigma - \Sigma_1$.* □

Thus, obtain the following optimized description of the constructor variants:

$$\begin{aligned}
[\![t]\!]^{\Omega,W}_{R,B} = &\left\{(t',\theta) \in [\![t]\!]^{W,x}_{R,B} \ \mid\ t \in T_{\Omega}(X)\right\} \cup \\
&\left\{(t'(\eta^{\bullet}), (\theta\eta^{\bullet})|_{vars(t)}) \in [\![t]\!]^{*}_{R,B} \ \mid \right. \\
&\left. (t',\theta) \in [\![t]\!]^{W,x}_{R,B} \wedge t' \in T_{\Sigma_1}(X) - T_{\Omega}(X) \wedge \eta \in \mathit{Unif}^{W,x,t'}_{B}(t' = x : ls_{\Sigma(X)}(t')_{\bullet})\right\}
\end{aligned} \tag{4.24}$$

Likewise, the following is an optimized description of constructor unifiers:

$$\begin{aligned}
\mathit{VarUnif}^{\Omega,W}_{E}(\phi) = &\left\{\alpha \in \mathit{VarUnif}^{W}_{E}(\phi) \mid (\phi\alpha)!_{R,B} \in T_{\Omega^{\wedge}}(X)\right\} \cup \\
&\left\{\alpha(\eta^{\bullet}) \ \mid\ \alpha \in \mathit{VarUnif}^{W}_{E}(\phi) \ \wedge \right. \\
&\left. (\phi\alpha)!_{R,B} \in T_{\Sigma_1^{\wedge}}(X) - T_{\Omega^{\wedge}}(X) \wedge \eta \in \mathit{Unif}^{W,\phi\alpha}_{B}((\phi\alpha)!_{R,B} = x : \mathit{Conj}_{\bullet})\right\}
\end{aligned} \tag{4.25}$$

Further optimizations are possible making the computation of constructor variants and constructor unifiers even easier in common cases. For example, when $\Omega = \widetilde{\Omega}$ (all symbols subsort-overloaded with symbols in $\Omega$ are constructors), and for each $f \in \Sigma - \Omega$ we have $B_{f,1} = \emptyset$, then $\Sigma_1 = \Omega$ and we obtain:

$$[\![t]\!]^{\Omega,W}_{R,B} = \{(t',\theta) \in [\![t]\!]^{W,x}_{R,B} \mid t \in T_{\Omega}(X)\} \tag{4.26}$$

Similarly, we also obtain:

$$\mathit{VarUnif}^{\Omega,W}_{E}(\phi) = \{\alpha \in \mathit{VarUnif}^{W}_{E}(\phi) \mid (\phi\alpha)!_{R,B} \in T_{\Omega^{\wedge}}(X)\}. \tag{4.27}$$

That is, when $\Sigma_1 = \Omega$, the computation of constructor variants, resp. constructor unifiers, is just a process of *filtering* them as a subset of the overall set of variants (resp. variant unifiers), essentially by syntactic checks.

### 4.3.4 Descent Maps

There are two ways in which the methods presented in this chapter may be insufficient to prove satisfiability of QF formulas in the initial algebra of an order-sorted equational theory having an FVP decomposition $\mathcal{R}$:

1. At the *theoretical* level, $\mathcal{R}$ may lack an OS-compact constructor decomposition, so that the methods presented here cannot be applied to $\mathcal{R}$.
2. At the *practical* level, even if $\mathcal{R}$ has an OS-compact constructor decomposition amenable to the methods and algorithms presented here, directly checking satisfiability of QF formulas in $\mathcal{R}$ may be quite inefficient. This can happen because: (i) $B$-unification itself may generate a large number of unifiers; and (ii) there may also be a large number of variant $R, B$-unifiers of a given unification problem $\varphi$.

Faced with any of these theoretical and/or practical limitations, the following notion of a *descent map*, presented in [25], may provide a way out of such limitations:

**Definition 4.17** *A* descent map *is a triple* $(\mathcal{R}, \bullet, \mathcal{D})$ *where* $\mathcal{R}$ *and* $\mathcal{D}$ *are decompositions of order-sorted equational theories, and* $\mathcal{R}$ *conservatively extends* $\mathcal{D}$*, and where* $\bullet$ *is a total computable function,* $\varphi \mapsto \varphi^\bullet$*, mapping each QF formula* $\varphi$ *in the theory decomposed by* $\mathcal{R}$ *into a corresponding QF formula* $\varphi^\bullet$ *in the theory decomposed by* $\mathcal{D}$ *and such that* $C_\mathcal{R} \models \exists\, \varphi \;\Leftrightarrow\; C_\mathcal{D} \models \exists\, \varphi^\bullet$*, where* $\exists\, \varphi$ *denotes the existential closure of* $\varphi$*.*

Limitation (1) can be overcome when $\mathcal{R}$ lacks an OS-compact constructor decomposition but $\mathcal{D}$ has one. And limitation (2) can be overcome because solving satisfiability in $C_\mathcal{D}$ of the QF formula $\varphi^\bullet$ may be considerably more efficient than solving satisfiability in $C_\mathcal{R}$ of the original formula $\varphi$. Since descent maps form a *category* and therefore can be *composed*, suitable compositions of such maps can greatly help in solving limitations (1) and (2). Furthermore, they can substantially extend the theoretical and practical reach of the variant satisfiability methods presented in this chapter.

In experimenting with the current implementation of the variant-satisfiability algorithms described in Section 4.4 for solving SMT problems for various automated deduction applications, we have found descent maps to be quite helpful in overcoming type (1) and (2) limitations, specifically in the context of Presburger arithmetic,[3] for the following reasons: (i) the simplest FVP specifications $\mathcal{Z}_{+,>,\geqslant}$, resp. $\mathcal{N}_{+,>,\geqslant}$, of Presburger arithmetic for the integers (resp. the naturals) fail to have OS-compact constructor decompositions [25]; (ii) solving satisfiability of QF formulas by variant satisfiability in the initial algebras of $\mathcal{Z}_{+,>,\geqslant}$, or even just in that of $\mathcal{Z}_{+}$ (the Abelian group of the integers) is quite inefficient due to a usually large number of variants modulo $ACU$; whereas (iii) solving satisfiability of QF formulas by variant satisfiability in the initial algebra of $\mathcal{N}_{+}$ (the Abelian monoid of the integers) is much more efficient, since, being free modulo $ACU$ and OS-compact, it essentially reduces to computing $ACU$ unifiers, which, although expensive for large terms, is efficiently supported by Maude 2.7.1.

In [25] three descent maps are defined: (i) $\mathcal{N}_{+,>,\geqslant} \xrightarrow{lit2at\ \delta_0} \mathcal{N}_{+}$, reducing natural Presburger arithmetic satisfiability to satisfiability in the Abelian monoid of the naturals; (ii) an entirely similar map $\mathcal{Z}_{+,>,\geqslant} \xrightarrow{lit2at\ \delta_0} \mathcal{Z}_{+}$, reducing integer Presburger arithmetic satisfiability to satisfiability in the Abelian group of the integers; and (iii) $\mathcal{Z}_{+} \xrightarrow{v\ -} \mathcal{N}_{+}$, reducing satisfiability in the Abelian group of the integers to satisfiability in the Abelian monoid of the naturals. These three maps ease limitations of type (1) and/or (2). Furthermore, it is shown in Theorems 14–15 of [25] that the descent maps (i)–(ii) can be modularly extended to FVP theory combinations

[3] This does not exclude the possibility of using not only descent maps, but also well-known *domain-specific* SMT solving algorithms for Presburger arithmetic. However, the applications we have experimented with, in which variant satisfiability algorithms are used, almost never involve just Presburger arithmetic alone. For such applications, the less efficient use of *theory-generic* variant satisfiability algorithms is compensated for by the trivial way in which various FVP theories can be combined by *theory union*, as opposed to by a more complex Nelson-Oppen theory combination infrastructure [19, 20]. Experimenting with such trade-offs between domain-specific and theory-generic algorithms and their various forms of composition is an important topic for future research.

where $\mathcal{N}_{+,>,\geqslant}$, resp. $\mathcal{Z}_{+,>,\geqslant}$, is a subspecification of a larger FVP theory. Although not explicitly treated in [25], descent map (iii) has a natural extension to a descent map (iv) $\mathcal{Z}_{+,>,\geqslant} \xrightarrow{v-} \mathcal{N}_{+,>,\geqslant}$, so that we get the following diagram of descent maps:

$$\begin{tikzcd}
\mathcal{Z}_{+,>,\geqslant} \arrow[r, "lit2at\ \delta_0"] \arrow[d, "v-"'] & \mathcal{Z}_{+} \arrow[d, "v-"] \\
\mathcal{N}_{+,>,\geqslant} \arrow[r, "lit2at\ \delta_0"] & \mathcal{N}_{+}
\end{tikzcd}$$

Figure 4.8: $\mathcal{Z}_{+,>,\geqslant}$ to $\mathcal{N}_{+}$ Descent Maps

We have implemented in Maude maps (i) and (iv) as meta-level functions and, as further explained in Section 4.5, have used those maps effectively in a considerable number of reachability logic verification tasks.

### 4.3.5 Variant Satisfiability Examples

At this point, we can put together all of the results in the previous sections to provide a *concrete* algorithm for solving variant satisfiability problems for simple theories (see Definition 4.6). We give two more extensive examples here.

**Example 4.13** *(Integers with Addition, Transitivity). Recall our running example $\mathcal{Z}_{+}$ originally defined in Example 4.3 and let us prove transitivity of $(<)$, i.e. the quantifier-free formula $\phi \equiv (i < j \wedge j < k) \Rightarrow i < k$ where $i, j, k\!:\!Int$. Even though our theory does not contain the operator $(<)$, this property is still expressible in the theory $\mathcal{Z}_{+}$ via descent maps (see Section 4.3.4 for details). This happens in two steps: (i) we exploit the fact $i < k \Leftrightarrow i \ngeqslant k$ to obtain $\phi' \equiv (i < j \wedge j < k) \Rightarrow i \ngeqslant k$ (ii) we rewrite the formula as $\phi'' \equiv (i + l = j \wedge j + m = k) \Rightarrow i \neq k + a$ where $l, m\!:\!NzNat$ and $a\!:\!Nat$. Because of the duality between validity and satisfiability in FOL, it is sufficient to prove $\neg\phi''$ is unsatisfiable. Our first step is to desugar the $(\Rightarrow)$ operator to obtain $i + l \neq j \vee j + m \neq k \vee i \neq k + a$. We then apply the negation and check unsatisfiability of $i + l = j \wedge j + m = k \wedge i = k + a$ which amounts to checking if any constructor variant unifiers exist; but there are none, so the original formula is valid.*

Here we provide another example which illustrates more clearly the advantages of the variant satisfiability technique, as well as makes use of the methods

**Example 4.14** *(Direct Theory Combination). Consider the theory* NatList *with signatures $\Omega$ and $\Sigma$ and their constructor sort refinements defined as in Figure 4.9. It is easy to show that $\Omega$ and $\Sigma$ are $B$-preregular for $B$ a set of ACU axioms on $(+)$ with identity $0$, $\Omega < \Sigma$, and that $B$ respects constructors. Thus, $B$-unification is well-defined in the extended signature $\Sigma^c(X^c)$. Furthermore, by applying Lemmas 4.5 and 4.6, it is easy to show that $s \in S/\{Bool\} \Leftrightarrow |T_{\Omega,s}| = \aleph_0$. Let $n, m\!:\!Nat$ and $l\!:\!List$. In addition to axioms $B$, our theory also contains the following equations $E_0$:*

$$n + m < n = ff \tag{4.28}$$

$$n < n + m + 1 = tt \tag{4.29}$$

$$hd(n\!:\!l) = n \tag{4.30}$$

$$tl(n\!:\!l) = l \tag{4.31}$$

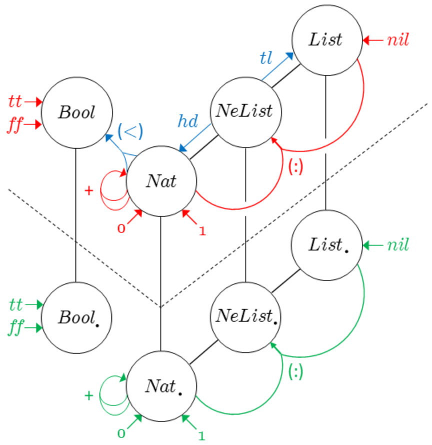


Figure 4.9: NatList signature and constructor sort refinement

*Then $\mathcal{R} = (\Sigma, E_0 \cup B)$ has a decomposition $(\Sigma, B, R)$ where $R = \vec{E_0}$ which protects the constructor decomposition $(\Omega, B, \varnothing)$. By folding-variant narrowing, it is easy to show that $\mathcal{R}$ is FVP. Clearly, $(\Sigma, E)$ is finite. Thus, $(\Sigma, E)$ is a* simple *theory and we can perform reasoning in it using variant satisfiability. Observe that this theory is an amalgamation of the theory of natural Presburger arithmetic and the theory of lists. However, due to the highly general nature of the variant satisfiability method, no Nelson-Oppen combination methods are required; we can reason directly in the theory $(\Sigma, E)$.*

*Suppose we wish to show $\phi \equiv hd(l) < hd(l') = b \wedge l = l' \wedge b \neq b' \wedge b' \neq tt$ is unsatisfiable where $l, l' : NeList$ and $b, b' : Bool$. This formula is already in DNF, so we first compute a complete set of constructor variant unifiers $VarUnif_E^{\Omega,W}(hd(l) < hd(l') = b \wedge l = l')$, which in our case, is the singleton $\alpha = \{b \mapsto ff, l \mapsto l_1, l' \mapsto l_2\}$ where $l_1, l_2 : NeList$. We then compute $\phi' \equiv (b \neq b' \wedge b' \neq tt)\alpha!_{R,B} = ff \neq b' \wedge b' \neq tt$. At this point, we need to compute a complete set of constructor variants of $\phi'$. In this case, we see that the most general constructor variant is $(\phi', id)$, since all terms are constructors. Next, we must compute the set of terms inhabiting each variable of finite sort. The only such variable is $b'$; our algorithm generates the two new formulas $ff \neq tt \wedge tt \neq tt$ and $ff \neq ff \wedge ff \neq tt$. Since the theory $(\Omega, B)$ is OS-compact, we can solve these formulas by checking for $E$-consistency. Clearly, neither is $E$-consistent; thus, the formula is unsatisfiable.*

## 4.4 REFLECTIVE IMPLEMENTATION OF VARIANT SATISFIABILITY

Here we describe our reflective Maude implementation of all the above metalevel algorithms. The complete codebase, with binaries and examples, can be downloaded from the Maude website: `http://maude.cs.illinois.edu/tools/var-sat/`.

We have already pointed out that both order-sorted equational logic and rewriting logic are reflective [58]. What this precisely means is that the metalevel of the logic, including theories and deduction, can be *reified*, i.e., *reflected at the object level* by means of a so-called *universal theory.* For rewriting logic this means that: (i) there is a finitary universal rewrite theory $\mathcal{U}$ such that any finitely presented rewrite theory $\mathcal{R}$ (including $\mathcal{U}$ itself) and any term $t$ in $\mathcal{R}$ can be represented as respective terms $\overline{\mathcal{R}}$ and $\overline{t}$ in $\mathcal{U}$, and (ii) for any formula $t \to t'$ in $\mathcal{R}$, its provability in $\mathcal{R}$ can likewise be represented as a formula $\overline{\mathcal{R} \vdash t \to t'}$ in $\mathcal{U}$, and (iii) we have an equivalence:

$$\mathcal{R} \vdash t \to t' \quad \Leftrightarrow \quad \mathcal{U} \vdash \overline{\mathcal{R} \vdash t \to t'} \tag{4.32}$$

so that $\mathcal{U}$ can faithfully simulate deduction in *any* finitary rewrite theory (including itself). Of course, simulating deductions in $\mathcal{U}$ can be quite inefficient. However, we can exploit the above equivalence by using it in the $(\Rightarrow)$ direction to greatly increase the efficiency of many deductions in $\mathcal{U}$ by performing them directly in the simulated theory $\mathcal{R}$. This is exactly the approach taken in Maude's `META-LEVEL` module, which efficiently implements various useful functions expressible in $\mathcal{U}$ by performing them directly in the simulated theory $\mathcal{R}$. Since equational logic is a sublogic of rewriting logic, all the above remarks apply likewise to equational theories and equational deduction.

Note that our metalevel algorithms and sub-algorithms can greatly benefit from a reflective implementation. This is not only because they are obviously theory-generic on the equational FVP theory decomposition $\mathcal{R}$ on which we want to decide QF satisfiability, but also because virtually all subalgorithms make crucial use of *theory and/or signature transformations* such as, for example, the $\mathcal{R} \mapsto \mathcal{R}^{\wedge}$, $\Sigma \mapsto \Sigma^c$, and $\Omega \mapsto \mathcal{R}_P(\Omega)$ transformations, to mention just three. Such transformations are easy to define by reflection as extensions of Maude's `META-LEVEL` module. Specifically, they can be equationally defined as functions of sort `Module`, whose elements are terms of the form $\overline{\mathcal{R}}$ meta-representing rewrite theories[4] $\mathcal{R}$

In summary, therefore, by using Maude's `META-LEVEL` module we can easily write functions over meta-level constructs to obtain a reflective implementation of our algorithms in a fairly direct way. Essentially, the reflective implementation of the variant satisfiability algorithm and its subalgorithms follows the outline sketched in Section 4.3. The only missing exception is that the finite sort checks for theories with unit axioms have not been implemented yet. The algorithm takes as input a reflected FVP decomposition $\overline{\mathcal{R}}$ and a conjunctive QF formula $\phi = \bigwedge G \wedge \bigwedge D$ in $\mathcal{R}$, and returns a boolean indicating whether or not the formula is satisfiable in the initial model of the decomposition $\mathcal{R}$. Thanks to mixfix parsing, we can use a more natural notation to write $\phi$ as:

$$\overline{u}_1 \ \mathtt{?=}\ \overline{v}_1 \ \mathtt{/\backslash}\ \cdots \ \mathtt{/\backslash}\ \overline{u}_k \ \mathtt{?=}\ \overline{v}_k \ \mathtt{/\backslash}\ \overline{u}'_1 \ \mathtt{!=}\ \overline{v}'_1 \ \mathtt{/\backslash}\ \cdots \ \mathtt{/\backslash}\ \overline{u}'_l \ \mathtt{!=}\ \overline{v}'_l \tag{4.33}$$

where each $\overline{u}_i, \overline{v}_i$ and $\overline{u}'_j, \overline{v}'_j$ for $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant l$ is a meta-term.

## 4.5 RELATED WORK AND CONCLUSIONS

We have presented the meta-level sub-algorithms needed to obtain a full-fledged variant satisfiability algorithm, proved them correct, and derived a Maude reflective implementation. Correctness has been the main concern, but efficiency has also been taken into account. Much work remains ahead. A crucial next step is experimentation. We have initiated such an experimentation by using the Maude reflective implementation of variant satisfiability as a key component to mechanize a new version of reachability logic for rewrite theories developed in [61], which further advances reachability logic ideas in [26, 29]. We have been able to verify various reachability properties for a substantial number of examples using the variant satisfiability algorithm as a backend procedure, as the next chapter will also attest. This is already helping us optimize the performance of the main algorithm and its subalgorithms, which has been an explicit theme in Sections 4.3.3–4.3.4. Further work is needed to experimentally evaluate our algorithm in a more systematic way. As pointed out in Footnote 3, this should also involve comparison with domain-specific algorithms when those are available, including a comparison of the trade-offs between different kinds of theory combination methods. Such comparisons will require developing new theory combination infrastructure not yet available

[4] A signature $\Sigma$ can be viewed as a theory $\Sigma$ with no axioms and meta-represented as $\overline{\Sigma}$ in the same way.

in our implementation (besides of course theory unions for FVP theories, which are fully supported already).

The most closely-related work is [25, 36], for which it provides the first full-fledged algorithm and implementation. Other related topics include folding variant narrowing [24], the FVP [23], and unsorted compactness [31]. Of course, this work occurs in the larger context of decidable satisfiability algorithms and the vast literature on SMT solving, e.g., [62, 63, 64, 65, 66, 62, 67, 68, 69], and additional references in [25, 36]. Finally, the literature on Maude's reflective algorithms and tools, e.g., [70, 6] is also closely related.

# CHAPTER 5 CONSTRUCTOR-BASED REACHABILITY LOGIC[1]

## 5.1 INTRODUCTION

The main past applications of reachability logic have been as a *language-generic* logic of programs [71, 26, 27]. In these applications, a $\mathbb{K}$ specification of a language's operational semantics by means of rewrite rules is assumed as the language's "golden semantic standard," and then a correct-by-construction reachability logic for a language so defined is automatically obtained [27]. This method has been shown effective in proving a wide range of properties of programs in real programming languages specified within the $\mathbb{K}$ Framework.

Although the original foundations of reachability logic are very general [26, 27], such foundations do not provide a straightforward answer to the following non-trivial questions: (1) Could a reachability logic be developed to verify not just conventional programs, but also *distributed system designs and algorithms* formalized as *rewrite theories* in rewriting logic [50, 28]? And (2) if so, what would be the most natural way to conceive such a *rewrite-theory-generic* logic? Since $\mathbb{K}$ specifications are just conditional rewrite theories [4], a satisfactory answer to questions (1)–(2) would move the verification game from the level of verifying *code* to that of verifying *both code and distributed system designs.* Since the cost of design errors can be several orders of magnitude higher than that of coding errors, questions (1) and (2) are of practical software engineering interest.

Although a first step towards a reachability logic for rewrite theories has been taken in [29], as explained in Section 5.6 and below, that first step still leaves several important questions open. The most burning one is: how can we prove *invariants* of a distributed system? Since invariants are the most basic safety properties, support for proving invariants is a *sine qua non* requirement. As explained below and in Section 5.3.1, if we apply the standard foundations of reachability logic—so that the logic's transition relation is instantiated to the given theory's rewrite relation—the whole enterprise collapses before what we call the *invariant paradox*: we cannot verify in this manner *any* invariants of a never-terminating system such as, for example, a mutual exclusion protocol.

A second, important open question is how to best take advantage of the wealth of equational reasoning techniques such as matching, unification, and narrowing modulo an equational theory $(\Sigma, E)$, e.g., [72, 73, 74, 75, 76, 77, 24, 78], as well as recent results on decidable satisfiability (and validity) of quantifier-free formulas in initial algebras, e.g., [14, 16, 31, 79, 18, 80, 81, 82, 83, 84, 36] to *automate* as much as possible reachability logic deduction. In this regard, the very general foundations of standard reachability logic—which assume any $\Sigma$-algebra $\mathcal{A}$ with a first-order-definable transition relation—provide no help at all for automation. As shown in this chapter and its prototype implementation, if we assume instead that the model in question is the *initial reachability model* $\mathcal{T}_{\mathcal{R}}$ of a rewrite theory $\mathcal{R}$ satisfying reasonable assumptions, large parts of the verification effort can be automated.

A third important issue is *simplicity.* Reachability logic has eight inference rules [26, 27]. Could a reachability logic for rewrite theories be simpler? The main goal of this chapter is to tackle head on these three open questions to provide a general reachability logic and a prototype tool suitable for reasoning about properties of *both* distributed systems and programs based on their rewriting logic semantics.

What all this really means requires some further explanations about both rewriting logic and reach-

[1] A previous version of the chapter content was originally available at https://doi.org/10.1007/978-3-319-94460-9_12; the current version will be published in a forthcoming issue of the journal *Fundamenta Informaticae.* Reprinted with permission.

ability logic. Rewriting logic is a *system specification* logic ideally suited for specifying concurrent systems. Instead, reachability logic is a *property specification* logic, where reachability properties of concurrent systems previously specified as rewrite theories can be defined and reasoned about. The pair (*Rewriting Logic*, *Reachability Logic*) is what is called a *tandem* in [85], where the left-side logic is used to specify the systems of interest, and the right-side logic to specify and verify relevant properties of those systems. The point of a well-designed tandem is that the property specification logic systematically exploits many features of the system specification logic to increase the effectiveness of verification. This is exactly what the *constructor-based* version of reachability logic we present here does by exploiting features of rewriting logic.

### 5.1.1 Rewriting Logic in a Nutshell

A distributed system can be designed and modeled as a *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$ [50, 28] in the following way: (i) the distributed system's *states* are modeled as elements of the initial algebra $T_{\Sigma/E}$ associated to the equational theory $(\Sigma, E)$ with function symbols $\Sigma$ and equations $E$; and (ii) the system's *concurrent transitions* are modeled by rewrite rules $R$, which are applied *modulo* $E$. Let us consider the QLOCK [86] mutual exclusion protocol, explained in detail in Section 5.1.5 and used later as a running example. QLOCK allows an unbounded number of processes, which can be identified by numbers. Such processes can be in one of three states: "normal" (doing their own thing), "waiting" for a resource, and "critical," i.e., using the resource. Waiting processes enqueue their identifier at the end of a waiting queue (a list), and can become critical when their name appears at the head of the queue. A QLOCK state can be represented as a tuple $< n \mid w \mid c \mid q >$ where $n$, resp. $w$, resp. $c$, denotes the set of identifiers for normal, resp. waiting, resp. critical processes, and $q$ is the waiting queue. QLOCK can be naturally modeled as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ where $\Sigma$ contains operators to build natural numbers, multisets of natural numbers, like $n$, $w$, and $c$, and lists of natural numbers like $q$, plus the above tupling operator. The equations $E$ include axioms such as the associativity-commutativity of multiset union, and the associativity of list concatenation, and identity axioms for $\varnothing$ and *nil*. QLOCK's behavior is specified by a set $R$ of five rewrite rules. For example, the rule *w2c* below specifies how a waiting process $i$ can pass from waiting to critical

$$\mathit{w2c} : < n \mid w\ i \mid c \mid i\ ; q > \to < n \mid w \mid c\ i \mid i\ ; q > .$$

Figure 5.1: QLOCK Example Rewrite Rule

### 5.1.2 Reachability Logic in a Nutshell

Reachability logic allows us to reason about the reachability properties of a concurrent system specified by rewrite theory $\mathcal{R}$. The constructor-based reachability logic we present in this chapter is *theory generic* in the precise sense that, as we explain in Section 5.4, its inference rules do not depend at all on the given theory $\mathcal{R}$: the reachability properties of any rewrite theory $\mathcal{R}$ in a wide class of so-called *suitable* theories can be reasoned about in our logic using the *same* inference rules. Such genericity is not enjoyed by other verification logics. For example, Hoare logic is *not* language generic: a different inference system must be hand-crafted and proved sound with respect to an operational semantics for each different programming language $\mathcal{L}$.

A *reachability formula* has the form $A \to^{\circledast} B$, where $A$ and $B$ are state predicates. Consider the easier to explain case where the formula has no parameters, i.e., $vars(A) \cap vars(B) = \varnothing$. We interpret such a formula in the initial reachability model $\mathcal{T}_{\mathcal{R}}$ of a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, whose states are $E$-equivalence classes $[u]$ of ground $\Sigma$-terms, and where a state transition $[u] \to_{\mathcal{R}} [v]$ holds iff $\mathcal{R} \vdash u \to v$ according to the rewriting logic inference system [50, 28] (computation = deduction). As a first approximation, $A \to^{\circledast} B$ is a Hoare logic *partial correctness* assertion of the form[2] $\{A\}\mathcal{R}\{B\}$, but with the slight twist that $B$ need not hold on a terminating state, but just *somewhere along the way*. Therefore, $B$ should not necessarily be called a "postcondition," but, more generally a *midcondition*. More precisely, $A \to^{\circledast} B$ holds in $\mathcal{T}_{\mathcal{R}}$ iff for each state $[u_0]$ satisfying $A$ and each *terminating* sequence $[u_0] \to_{\mathcal{R}} [u_1] \ldots \to_{\mathcal{R}} [u_{n-1}] \to_{\mathcal{R}} [u_n]$, i.e., $\nexists u\ (u_n \to_{\mathcal{R}} u)$, there is a $j$, $0 \leqslant j \leqslant n$, such that $[u_j]$ satisfies $B$. A key question is how to choose a good language of state predicates like $A$ and $B$. Here is where the potential for increasing the logic's automation resides.

As an example of state predicates $A$ and $B$ with *parameters*, i.e., $vars(A) \cap vars(B) \neq \varnothing$, consider a *counter system*, whose states are built using a state constructor $\langle \_ \rangle : Nat \to State$. The rewrite theory specifying the counter's behavior has two rewrite rules: $\langle n+1 \rangle \to \langle n \rangle$, and $\langle n+1 \rangle \to \langle n+1+1 \rangle$, i.e., a non-zero counter can increase or decrease by one unit. For $n, m$ variables of sort *Nat*, consider the reachability formula $\langle n+1 \rangle \mid n+1 > m \to^{\circledast} \langle m \rangle \mid \top$, which is parametric on $m$. This reachability formula uses a so-called *constrained constructor pattern* $\langle n+1 \rangle \mid n+1 > m$ to specify its precondition, and another $\langle m \rangle \mid \top$ specifying its midcondition. It states that, on all terminating paths, a non-zero counter of the form $\langle n+1 \rangle$ will pass through all states of the form $\langle m \rangle$ such that $m < n+1$ on its way to the terminating state $\langle 0 \rangle$. For this formula to have the desired semantics, the value of variable $m$ occurring in its precondition *and* its midcondition must of course be *the same*. We can reduce the parameterized case to the unparameterized one by considering this parametric formula as the *infinitary conjunction* of all the unparameterized instances where the parameter $m$ has been instantiated to a concrete number. Correct deductive reasoning about parameterized reachability formulas requires special handling of parameters.

We call our proposed logic *constructor-based* because our choice is to make $A$ and $B$ *positive* (only $\vee$ and $\wedge$) Boolean combinations of what we call *constrained constructor patterns* of the form $u \mid \varphi$, where $u$ is a *constructor* term[3] and $\varphi$ a quantifier-free (QF) $\Sigma$-formula. The state predicate $u \mid \varphi$ holds for a state $[u'] \in T_{\Sigma/E}$ iff there is a ground substitution $\rho$ such that $[u'] = [u\rho]$ and $T_{\Sigma/E} \models \varphi\rho$. This is crucially important, because the initial algebra of constructor terms is typically *much simpler* than the initial $(\Sigma, E)$-algebra $T_{\Sigma/E}$, and this can be systematically exploited for matching, unification, narrowing, and satisfiability purposes to automate large portions of reachability logic's inference system.

### 5.1.3 The Invariant Paradox

This is all very well, but how can we *prove invariants* in such a reachability logic? For example, we would like to prove that for QLOCK a mutual exclusion invariant holds. But, paradoxically, we cannot! The simple reason is that QLOCK, like many other protocols, *never terminates*, that is, has no terminating sequences whatsoever. But this has the ludicrous trivial consequence that QLOCK's initial reachability model $\mathcal{T}_{\mathcal{R}}$ vacuously satisfies *all* reachability formulas $A \to^{\circledast} B$. This of course means that it is in fact *impossible* to prove any invariants using reachability logic in $\mathcal{T}_{\mathcal{R}}$. But it does *not* mean that it is impossible using some

[2] The notation $\{A\}\mathcal{R}\{B\}$, and the relation to Hoare logic are explained in Section 5.3.2.

[3] That is, a term in a subsignature $\Omega \subseteq \Sigma$ such that each ground $\Sigma$-term is equal modulo $E$ to a ground $\Omega$-term.

other reachability model. In Section 5.3.1 we give a systematic solution to this paradox by means of a *simple theory transformation* allowing us to prove any invariant in the original initial reachability model $\mathcal{T}_{\mathcal{R}}$ by proving an equivalent reachability formula in the initial reachability model of the transformed theory.

### 5.1.4 Chapter Overview

Section 5.2 greatly increases the logic's potential for automation by making state predicates constructor-based. Reachability logic itself is introduced in Section 5.3. A systematic methodology to prove *invariants* by means of reachability formulas is developed in Section 5.3.1. The semantic relations of reachability logic to Hoare logic and to LTL are explained in Section 5.3.2. Rewriting logic's inference system, with just *three* inference rules (plus some auxiliary rules), and the proof of its soundness are presented in Section 5.4. A proof of concept of our approach is given by means of a prototype tool implemented in the Maude rewriting logic system and a suite of experiments verifying various properties of distributed system designs and imperative programs in Section 5.5. Related work and conclusions are discussed in Section 5.6. Proofs are relegated to Appendix D. The tool's command grammar is specified in detail in Appendix A.

### 5.1.5 A Running Example

Consider the following rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ modeling a dynamic version of the QLOCK mutual exclusion protocol [86], where $(\Sigma, B)$ defines the protocol's states, involving natural numbers, lists, and multisets over natural numbers. $\Sigma$ has sorts $S = \{\mathit{Nat}, \mathit{List}, \mathit{MSet}, \mathit{NeMSet}, \mathit{Conf}, \mathit{State}, \mathit{Pred}\}$ with subsorts $\mathit{Nat} < \mathit{List}$ and $\mathit{Nat} < \mathit{NeMSet} < \mathit{MSet}$ and also the set of operators $F = \{\mathit{0} : \to \mathit{Nat},\ s\_ : \mathit{Nat} \to \mathit{Nat},\ \mathit{nil} : \to \mathit{List},\ \_;\_ : \mathit{List}\ \mathit{List} \to \mathit{List},\ \varnothing : \to \mathit{MSet},\ \_\_ : \mathit{MSet}\ \mathit{MSet} \to \mathit{MSet},\ \_\_ : \mathit{NeMSet}\ \mathit{NeMSet} \to \mathit{NeMSet},\ \_|\_|\_|\_ : \mathit{MSet}\ \mathit{MSet}\ \mathit{MSet}\ \mathit{List} \to \mathit{Conf},\ <\_> : \mathit{Conf} \to \mathit{State},\ \mathit{tt} : \to \mathit{Pred},\ \mathit{ff} : \to \mathit{Pred},\ \mathit{dupl} : \mathit{MSet} \to \mathit{Pred},\ \mathit{dupl} : \mathit{NeMSet} \to \mathit{Pred}\}$, where any underscores denote operator argument placement. The axioms $B$ are the associativity-commutativity of the multiset union $\_\_$ with identity $\varnothing$, and the associativity of list concatenation $\_;\_$ with identity $\mathit{nil}$. The equations in $E$ are $\mathit{dupl}(s\ u\ u) = \mathit{tt}$ and $\mathit{dupl}(\varnothing) = \mathit{ff}$, They define the $\mathit{dupl}$ predicate by detecting a duplicated non-empty multiset $u$ in the multiset $s\,u\,u$ (where $s$ could be empty). $\mathit{dupl}$ is false for the empty multiset, and is not true (but not explicitly defined to be false) in all other cases not covered by the equation $\mathit{dupl}(s\ u\ u) = \mathit{tt}$. The *states* of QLOCK are $B$-equivalence classes of ground terms of sort $\mathit{State}$.

QLOCK [86] is a mutual exclusion protocol where the number of processes is unbounded. Furthermore, in the *dynamic* version of QLOCK presented below, such a number can grow or shrink. Each process is identified by a number. The system configuration has three sets of processes (normal, waiting, and critical) plus a waiting queue. To ensure mutual exclusion, a normal process must first register its name at the end of the waiting queue. When its name appears at the front of the queue, it is allowed to enter the critical section. The first three rewrite rules in $R$ below specify how a *normal* process $i$ first transitions to a *waiting* process, then to a *critical* process, and back to normal. The last two rules in $R$ specify how a process can dynamically join or exit the system.
where $\phi \equiv \mathit{dupl}(n\,i\,w\ c) \neq \mathit{tt}$, $i$ is a number, $n$, $w$, and $c$ are, respectively, normal, waiting, and critical process identifier sets, and $q$ is a queue of process identifiers. It is easy to check that $(\Sigma, E \cup B)$ satisfies the finite variant property—it has only a single predicate $\mathit{dupl}$—and that $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfies sub-requirements (1)–(3) of Definition 2.14. Note that *join* makes QLOCK an *open* system in the sense explained

$$\begin{array}{rllll}
\mathit{n2w} : & < n\ i \mid\ w \mid\ c \mid\ q > & \rightarrow & < n \mid w\ i \mid\ c \mid\ q\ ;\ i > & \\
\mathit{w2c} : & < n \mid w\ i \mid\ c \mid i\ ;\ q > & \rightarrow & < n \mid\ w \mid c\ i \mid i\ ;\ q > & \\
\mathit{c2n} : & < n \mid\ w \mid c\ i \mid i\ ;\ q > & \rightarrow & < n\ i \mid\ w \mid\ c \mid\ q > & \\
\mathit{join} : & < n \mid\ w \mid\ c \mid\ q > & \rightarrow & < n\ i \mid\ w \mid\ c \mid\ q > & \mathit{if}\ \phi \\
\mathit{exit} : & < n\ i \mid\ w \mid\ c \mid\ q > & \rightarrow & < n \mid\ w \mid\ c \mid\ q > &
\end{array}$$

Figure 5.2: QLOCK Rewrite Rules

above.

## 5.2 CONSTRAINED CONSTRUCTOR PATTERN PREDICATES

Given an OS equational theory $(\Sigma, E \cup B)$, the *atomic state predicates* appearing in the constructor-based reachability logic formulas of Section 5.3 will be pairs $u \mid \varphi$, called *constrained constructor patterns*, with $u$ a term in a subsignature $\Omega \subseteq \Sigma$ of constructors, and $\varphi$ a quantifier-free $\Sigma$-formula. Intuitively, $u \mid \varphi$ is a pattern describing the set of states that are $E_\Omega \cup B_\Omega$-equal to ground terms of the form $u\rho$ for $\rho$ a ground constructor substitution such that $T_{\Sigma/E \cup B} \models \varphi\rho$. Therefore, $u \mid \varphi$ can be used as a *symbolic description* of a, typically infinite, *set of states* in the canonical reachability model $\mathcal{C}_{\mathcal{R}}$ of a rewrite theory $\mathcal{R}$.

We are now ready to define constrained constructor pattern predicates and their semantics. In what follows, $X$ will always denote the countably infinite $S$-sorted set of variables used in the language of $\Sigma$-formulas.

**Definition 5.1 (Constrained Constructor Pattern Predicate)** *Let theory* $(\Omega, B_\Omega, \vec{E_\Omega})$ *be a* constructor decomposition *of* $(\Sigma, B, \vec{E})$. *An s-sorted* atomic constrained constructor pattern predicate *is an expression* $u \mid \varphi$ *with* $u \in T_\Omega(X)_s$ *and* $\varphi$ *a QF* $\Sigma$*-formula. The set* $PatPred(\Omega, \Sigma)_s$ *of s-sorted* constrained constructor pattern predicates *contains* $\bot$*, all s-sorted atomic constrained constructor pattern predicates, and is closed under disjunction* ($\vee$) *and conjunction* ($\wedge$). *Let* $PatPred(\Omega, \Sigma) = \bigcup_{s \in S} PatPred(\Omega, \Sigma)_s$. *Capital letters* $A, B, \ldots, P, Q, \ldots$ *range over* $PatPred(\Omega, \Sigma)$. *The* semantics *of a constrained constructor pattern predicate* $A$ *is the subset* $[\![A]\!] \subseteq C_{\Sigma/E,B}$ *defined inductively as follows:*

1. $[\![\bot]\!] = \varnothing$
2. $[\![u \mid \varphi]\!] = \{[(u\rho)!]_{B_\Omega} \in C_{\Sigma/E,B} \mid \rho \in [X \to T_\Omega] \ \wedge\ C_{\Sigma/E,B} \models \varphi\rho\}$.
3. $[\![A \vee B]\!] = [\![A]\!] \cup [\![B]\!]$
4. $[\![A \wedge B]\!] = [\![A]\!] \cap [\![B]\!]$.

Note that for any constructor pattern predicate $A$, if $\sigma$ is a (sort-preserving) bijective renaming of variables we always have $[\![A]\!] = [\![A\sigma]\!]$.

**Example 5.1 (Pattern Predicate Example)** *Recall that QLOCK states have the form* $< n \mid w \mid c \mid q >$ *with* $n$, $w$, $c$ *multisets of process identifiers and* $q$ *an associative list of process identifiers. From the five rewrite rules defining QLOCK, it is easy to prove that if* $< n \mid w \mid c \mid q > \rightarrow^* < n' \mid w' \mid c' \mid q' >$ *and* $n\,w\,c$

*is a set (has no repeated elements), then $n'\ w'\ c'$ is also a set. Of course, it seems very reasonable to assume that these process identifier multisets are, in fact, sets, since otherwise we could, for example, have a process $i$ that is* both *waiting and critical at the* same *time. We can rule out such ambiguous states by means of the pattern predicate $< n \mid w \mid c \mid q > \mid dupl(n\ w\ c) \neq tt$.*

Now that we have explained our notion of constrained constructor pattern predicate, it is worth pausing for a moment to explain why they do play a crucial role in the *constructor-based* reachability logic that we shall define in Sections 5.3 and 5.4. The answer is simple: they support *symbolic reasoning about reachability.* Why so? For six reasons: (i) the constructor subtheory $(\Omega, E_\Omega \cup B_\Omega)$ is often *much simpler* than the equational theory $(\Sigma, E \cup B)$; (ii) in particular, in practice $(\Omega, E_\Omega \cup B_\Omega)$ almost always has the *finite variant property* (FVP) [23, 24] and therefore, assuming a $B_\Omega$-unification algorithm, it has a $E_\Omega \cup B_\Omega$-*unification algorithm* computable by folding variant narrowing [24]; (iii) as we shall show in Lemma 5.5, under mild conditions the rewrite theory $\mathcal{R}$ can be transformed into a *semantically equivalent* rewrite theory $\hat{\mathcal{R}}$ whose rewrite rules have the form $l \to r\ \mathit{if}\ \phi$, with $l$ and $r$ $\Omega$-terms, and $\phi$ a QF $\Sigma$-formula; (iv) but this means that we can *symbolically describe* possibly infinite sets of states by means of constructor pattern predicates; (v) it also means that we can *effectively symbolically describe* how such sets of states are transformed by transitions with the rules $l \to r\ \mathit{if}\ \phi$ in $\hat{\mathcal{R}}$ using narrowing techniques [78, 87] based on $E_\Omega \cup B_\Omega$-unification (for more on this, see the explanation of the $\text{Step}^\forall$ inference rule in Section 5.4); and (vi) as explained below, many *logical and set-theoretic operations* on constructor pattern predicates can also be *effectively symbolically described* by corresponding constructor pattern predicates. The overall effect of (i)–(vi) is that in constructor-based reachability logic large parts of the formal reasoning process can be *automated by symbolic methods.*

### 5.2.1 Constrained Constructor Pattern Operations

Let $A$, $B$, and $C$ be pattern predicates. In the remainder of this section, we define the following operations and show how they can be automated:

1. Pattern subsumption: to show that $[\![A]\!] \subseteq [\![B]\!]$

2. Over-approximating a complement: finding $B$ where $[\![B]\!] \supseteq ([\![u \mid \top]\!] \setminus [\![u \mid \phi]\!])$

3. Pattern intersection: finding $C$ where $[\![C]\!] = [\![A \wedge B]\!]$

4. Parameterized pattern subsumption: subsumption with shared variables

5. Parameterized pattern intersection: intersection with shared variables.

These operations will help in automating our reachability logic inference system.

**Pattern Subsumption.** Given constructor patterns $u \mid \varphi$ and $v \mid \psi$, where, without loss of generality, we assume that $\mathit{vars}(u \mid \varphi) \cap \mathit{vars}(v \mid \psi) = \varnothing$, we are seeking a symbolic sufficient condition to check that $[\![u \mid \varphi]\!] \subseteq [\![v \mid \psi]\!]$. The key intuition is that if the term $u$ is an *instance* modulo $E_\Omega \cup B_\Omega$ of the term $v$ by some substitution $\beta$ and $T_{\Sigma/E \cup B} \models \varphi \Rightarrow (\psi\beta)$, then such a set containment will hold. However, since: (i) $u$ can be an instance of $v$ in *several* ways, and (ii) we could consider not just one $v$, but a family $\{v_i\}_{i \in I}$, this intuition can be further generalized in two ways. First, we can ask the more general question of when the pattern $u \mid \varphi$ is an instance, not of a single pattern $v \mid \psi$, but of a finite family $\{v_i \mid \psi_i\}_{i \in I}$ of such patterns.

Second, we can capture all the ways that $u$ can be an instance of some $v_i$ by defining, for a set $Y$ of variables called *parameters* (not needed now, so assume $Y = \varnothing$ for the moment, but needed later):

$$\text{MATCH}(u,\ \{v_i\}_{i\in I}, Y) \equiv \{(i,\beta) \mid \beta \in [vars(v_i)\backslash Y \to T_\Omega(X)] \wedge u =_{E_\Omega \cup B_\Omega} v_i\beta\} \tag{5.1}$$

as a *complete* set of (parameter-preserving) $E_\Omega \cup B_\Omega$-matches of $u$ against the $v_i$. Since these matching substitutions are defined up to $E_\Omega \cup B_\Omega$-equality, it is enough to choose a representative matching substitution $\beta$ in each equivalence class $[\beta]_{E_\Omega \cup B_\Omega}$. That is, we should think somewhat more abstractly of the elements of $\text{MATCH}(u,\ \{v_i\}_{i\in I}, Y)$ as pairs $(i, [\beta]_{E_\Omega \cup B_\Omega})$.

Then we can generalize our intuition of $u \mid \varphi$ being an *instance* of $v \mid \psi$ by defining the notion that the family of patterns $\{v_i \mid \psi_i\}_{i\in I}$ (thought of as a disjunction $\bigvee_{i\in I} v_i \mid \psi_i$) *subsumes* $u \mid \varphi$ , denoted $u \mid \varphi \sqsubseteq \bigvee_{i\in I} v_i \mid \psi_i$, iff $T_{\Sigma/E\cup B} \models \varphi \Rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\ \{v_i\}_{i\in I},\varnothing)} \psi_i\beta$. The fact that, indeed, if this symbolic condition holds, we have a set containment of the form $[\![u \mid \varphi]\!] \subseteq [\![\bigvee_{i\in I} v_i \mid \psi_i]\!]$ follows (for the case $Y = \varnothing$) from the more general Lemma 5.2 later in this section. Computationally, subsumption is a relatively cheap,[4] sufficient condition to check a set inclusion of the form $[\![v \mid \psi]\!] \subseteq [\![\bigvee_{i\in I} u_i \mid \varphi_i]\!]$, but of course it is not a necessary condition. For example, if $\langle \_, \_ \rangle$ is a pairing operator forming pairs of natural numbers in Peano notation, we have an inclusion $[\![\langle n,m\rangle \mid \top]\!] \subseteq [\![\langle x,0\rangle \mid \top\ \vee\ \langle y,s(z)\rangle \mid \top]\!]$, but of course $\langle n,m\rangle \mid \top \not\sqsubseteq \langle x,0\rangle \mid \top\ \vee\ \langle y,s(z)\rangle \mid \top$. Nevertheless, a simple "inductive" instantiation of the variable $m$ by 0 and $s(k)$ can yield a proof by subsumption for the above set inclusion.

**Over-Approximating Complements.** It follows trivially from the semantics of pattern predicates that for any QF $\Sigma$-formula $\varphi$ we always have an inclusion $[\![u \mid \varphi]\!] \subseteq [\![u \mid \top]\!]$. The reason why *negation* has been excluded from the above definition of pattern predicates is that the naive assumption that we would have a set-theoretic equality $[\![u \mid \top]\!]\backslash[\![u \mid \varphi]\!] = [\![u \mid \neg\varphi]\!]$ is false in general, even assuming that $vars(\varphi) \subseteq vars(u)$. We always have $[\![u \mid \top]\!] = [\![u \mid \varphi]\!] \cup [\![u \mid \neg\varphi]\!]$, but in general we only have $[\![u \mid \top]\!]\backslash[\![u \mid \varphi]\!] \subseteq [\![u \mid \neg\varphi]\!]$.

For a simple example, consider sorts *Elt* and *MSet* with subsort inclusion *Elt* < *MSet*, constants $a, b, c$ of sort *Elt*, an associative-commutative multiset union operator $\_,\_$ and variables $x, y$ of sort *Elt*. Then, enclosing multisets in parentheses for clarity, so that, e.g., the multiset $a, b, b, c$ is denoted $(a, b, b, c)$, we have:

$$\begin{aligned}[\![x, y, z \mid x \neq y]\!] = \{&(a,b,c), (a,a,b), (a,a,c),\\ &(b,b,a), (b,b,c), (c,c,a), (c,c,b)\}\end{aligned} \tag{5.3}$$

$$\begin{aligned}[\![x, y, z \mid x = y]\!] = \{&(a,a,a), (b,b,b), (c,c,c), (a,a,b), (a,a,c),\\ &(b,b,a), (b,b,c), (c,c,a), (c,c,b)\}\end{aligned} \tag{5.4}$$

$$[\![x, y, z \mid \top]\!]\backslash[\![x, y, z \mid x \neq y]\!] = \{(a,a,a), (b,b,b), (c,c,c)\}. \tag{5.5}$$

[4] This remark should be taken with several grains of salt. The matching involved can be quite cheap in practice if $E_\Omega = \varnothing$ and $B_\Omega$ consists of axioms such as associativity or associativity-commutativity and the terms involved are not too large. It is still possible and automatable in Maude when axioms $B_\Omega$ are like that, and $E_\Omega \cup B_\Omega$ has the finite variant property [23, 24]; but it will be more expensive. In general, the validity check $\varphi \Rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\ \{v_i\}_{i\in I},\varnothing)} \psi_i\beta$ may not be cheap and may even be undecidable, since it is an inductive property. However: (i) this check *is* automatable in Maude when $E \cup B$ has the finite variant property and $E_\Omega \cup B_\Omega$ is OS-compact [25, 88]; and (ii) even though the inductive validity of $\varphi \Rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\ \{v_i\}_{i\in I},\varnothing)} \psi_i\beta$ is generally undecidable, in practice the use of simplification techniques and of user-provided lemmas, to be later discharged as proof obligations, can nevertheless suffice for proving it. To begin with, the Boolean equivalences

$$A \Rightarrow B\ \equiv\ \neg(A) \vee B\ \equiv\ \neg(A) \vee B \vee B\ \equiv\ (A \wedge \neg B) \Rightarrow B \tag{5.2}$$

make such simplification techniques more effective by checking instead the equivalent inductive validity of $(\varphi \wedge \bigwedge_{(i,\beta)\in\text{MATCH}(u,\ \{v_i\}_{i\in I},\varnothing)} \neg\psi_i\beta) \Rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\ \{v_i\}_{i\in I},\varnothing)} \psi_i\beta$, which has a stronger condition.

Nevertheless, the set identity $[\![u \mid \varphi]\!] \cup [\![u \mid \neg\varphi]\!] = [\![u \mid \top]\!]$ gives us the *set containment* $[\![u \mid \top]\!] \backslash [\![u \mid \varphi]\!] \subseteq [\![u \mid \neg\varphi]\!]$. Therefore, $[\![u \mid \neg\varphi]\!]$ gives us a cheap, symbolic way to *over-approximate* the set difference $[\![u \mid \top]\!] \backslash [\![u \mid \varphi]\!]$.

More generally, the powerset of $u$-pattern-definable subsets of $[\![u \mid \top]\!]$ of the form $[\![u \mid \varphi]\!]$ is obviously closed under finite unions, $\bigcup_{1 \leqslant i \leqslant n} [\![u \mid \varphi_i]\!] = [\![u \mid \varphi_i \vee \ldots \vee \varphi_n]\!]$. Likewise, it is closed under finite intersections $\bigcap_{1 \leqslant i \leqslant n} [\![u \mid \varphi_i]\!] = [\![u \mid \varphi_i \wedge \ldots \wedge \varphi_n]\!]$ (were without loss of generality we assume for $1 \leqslant i < j \leqslant n$ that $(vars(\varphi_i) \backslash vars(u)) \cap (vars(\varphi_j) \backslash vars(u)) = \varnothing$). But since $[\![u \mid \varphi]\!] \backslash [\![u \mid \psi]\!] = [\![u \mid \varphi]\!] \cap ([\![u \mid \top]\!] \backslash [\![u \mid \psi]\!])$, we can symbolically define the *over-approximated set difference* $[\![u \mid \varphi]\!] \backslash\backslash [\![u \mid \psi]\!]$ by means of the equality:

$$[\![u \mid \varphi]\!] \backslash\backslash [\![u \mid \psi]\!] =_{def} [\![u \mid \varphi]\!] \cap [\![u \mid \neg\psi]\!] = [\![u \mid \varphi \wedge \neg\psi]\!] \tag{5.6}$$

assuming again that $(vars(\varphi) \backslash vars(u)) \cap (vars(\psi) \backslash vars(u)) = \varnothing$. Such computationally cheap set difference over-approximations are exploited by reachability logic's inference system (see Section 5.4).

**Intersecting Patterns by Unification.** Note that, assuming that $E_\Omega \cup B_\Omega$ has a finitary unification algorithm, any constrained constructor pattern predicate $A$ is semantically equivalent to a finite disjunction $\bigvee_i u_i \mid \varphi_i$ of constrained constructor patterns. This is because: (i) by (3)–(4) in Def. 5.1 we may assume $A$ is in disjunctive normal form; and (ii) it is easy to check that $[\![(u \mid \varphi) \wedge (v \mid \phi)]\!] = \bigcup_{\alpha \in Unif_{E_\Omega \cup B_\Omega}(u,v)} [\![u\alpha \mid (\varphi \wedge \phi)\alpha]\!]$, where we assume without loss of generality that $vars(u \mid \varphi) \cap vars(v \mid \phi) = \varnothing$, and that all variables in $ran(\alpha)$ are *fresh*.

**Parametrized Intersections.** In the above discussion of intersections it was assumed that the variables in the two constructor patterns are disjoint. But this may not always be what we want. Consider constrained patterns $u \mid \varphi$ and $v \mid \phi$ with $Y = vars(u \mid \varphi) \cap vars(v \mid \phi)$. The sharing of variables $Y$ may be intentional as *parameters* common to both $u \mid \varphi$ and $v \mid \phi$. Using the algebraic notation $\mathbb{N} = \{0, s(0), s(s(0)), \ldots\}$, this can be illustrated by two patterns describing triples of natural numbers, namely, $\langle 0, y, z\rangle \mid \top$ and $\langle x, s(y), s(0)\rangle \mid \top$ with shared parameter $y$. We can view these patterns *parametrically* as describing the $\mathbb{N}$-*indexed families* of sets: $\{\{\langle 0, n, z\rangle \mid z \in \mathbb{N}\}\}_{n \in \mathbb{N}}$ and $\{\{\langle x, s(n), s(0)\rangle \mid x \in \mathbb{N}\}\}_{n \in \mathbb{N}}$. Then their $\mathbb{N}$-*indexed intersection* $\{\{\langle 0, n, z\rangle \mid z \in \mathbb{N}\} \cap \{\langle x, s(n), s(0)\rangle \mid z \in \mathbb{N}\}\}_{n \in \mathbb{N}} = \{\varnothing\}_{n \in \mathbb{N}}$ can then be symbolically described by $\perp$, because the terms $\langle 0, y, z\rangle$ and $\langle x, s(y), s(0)\rangle$ have *no unifier*, although by renaming $\langle x, s(y), s(0)\rangle$ to $\langle x, s(y'), s(0)\rangle$ they *can be unified* into the term $\langle 0, s(y''), s(0)\rangle$, so that $[\![\langle 0, y, z\rangle \mid \top]\!] \cap [\![\langle x, s(y), s(0)\rangle \mid \top]\!] = [\![\langle 0, s(y''), s(0)\rangle \mid \top]\!]$.

This suggests that if $u \mid \varphi$ and $v \mid \phi$ are pattern predicates with shared parameters $Y = vars(u \mid \varphi) \cap vars(v \mid \phi)$, we can consider them as describing parameterized families of sets $\{[\![(u \mid \varphi)\rho]\!]\}_{\rho \in [Y \to T_\Omega]}$ and $\{[\![(v \mid \phi)\rho]\!]\}_{\rho \in [Y \to T_\Omega]}$. We can then define their $Y$-*parameterized conjunction* as the pattern predicate

$$(u \mid \varphi) \wedge_Y (v \mid \phi) \quad = \bigvee_{\alpha \in Unif_{E_\Omega \cup B_\Omega}(u,v)} (u \mid \varphi \wedge \phi)\alpha \tag{5.7}$$

where, to avoid any variable capture, all variables in $ran(\alpha)$ are assumed *fresh*.

To emphasize that this models a $Y$-parameterized intersection, we then use the notation, $[\![(u \mid \varphi) \wedge_Y (v \mid \phi)]\!] = [\![u \mid \varphi]\!] \cap_Y [\![v \mid \phi]\!]$. The specific sense in which $(u \mid \varphi) \wedge_Y (v \mid \phi)$ symbolically models the parameterized intersection of the families of sets $\{[\![(u \mid \varphi)\rho]\!]\}_{\rho \in [Y \to T_\Omega]}$ and $\{[\![(v \mid \phi)\rho]\!]\}_{\rho \in [Y \to T_\Omega]}$ can be made precise as follows:

**Lemma 5.1** *For $u \mid \varphi$ and $v \mid \phi$ pattern predicates, with $Y = vars(u \mid \varphi) \cap vars(v \mid \phi)$, the following set* *identity holds:*

$$\bigcup_{\rho\in[Y\to T_\Omega]} [\![(u \mid \varphi)\rho]\!] \cap [\![(v \mid \phi)\rho]\!] \quad = \quad [\![u \mid \varphi]\!] \cap_Y [\![v \mid \phi]\!]. \tag{5.8}$$

**Parametrized Containments.** The notion of set containment also makes sense for indexed families of sets. For example, given $\mathbb{N}$-*indexed families* of sets: $\{\{\langle s(s(x)), n, s(0)\rangle \mid x, y \in \mathbb{N}\}\}_{n\in\mathbb{N}}$ and $\{\{\langle s(x'), n, s(y')\rangle \mid x', y' \in \mathbb{N}\}\}_{n\in\mathbb{N}}$, we say that the first is *contained* in the second, denoted $\{\{\langle s(s(x)), n, s(0)\rangle \mid x, y \in \mathbb{N}\}\}_{n\in\mathbb{N}} \subseteq \{\{\langle s(x'), n, s(y')\rangle \mid x', y' \in \mathbb{N}\}\}_{n\in\mathbb{N}}$ iff, by definition,

$$\forall n \in \mathbb{N} \quad \{\langle s(s(x)), n, s(0)\rangle \mid x, y \in \mathbb{N}\} \subseteq \{\langle s(x'), n, s(y')\rangle \mid x', y' \in \mathbb{N}\}, \tag{5.9}$$

which is actually the case for this example. In reachability logic applications we will often encounter the case of two pattern predicates $u \mid \varphi$ and $\bigvee_{i\in I} v_i \mid \psi_i$ for which their shared variables $Y = \mathit{vars}(u \mid \varphi) \cap \mathit{vars}(\bigvee_{i\in I} v_i \mid \psi_i)$ are indeed parameters, so that the semantic meaning of their *set containment* is the containment of a parametric family of sets, i.e.,

$$\forall \rho \in [Y\to T_\Omega] \quad [\![(u \mid \varphi)\rho]\!] \subseteq [\![(\bigvee_{i\in I} v_i \mid \psi_i)\rho]\!]. \tag{5.10}$$

To distinguish this notion of set containment from the standard one, where we may always rename $u \mid \varphi$ and $\bigvee_{i\in I} v_i \mid \psi_i$ so that $\mathit{vars}(u \mid \varphi) \cap \mathit{vars}(\bigvee_{i\in I} v_i \mid \psi_i) = \varnothing$, we write it as follows: $[\![u \mid \varphi]\!] \subseteq_Y [\![\bigvee_{i\in I} v_i \mid \psi_i]\!]$. Under these assumptions, there is a natural notion of $Y$-*parameterized subsumption* of $u \mid \varphi$ by $\bigvee_{i\in I} v_i \mid \psi_i$, denoted $u \mid \varphi \sqsubseteq_Y \bigvee_{i\in I} v_i \mid \psi_i$, namely, such subsumption holds iff $T_{\Sigma/E\cup B} \models \varphi \Rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\ \{v_i\}_{i\in I},Y)} \psi_i\beta$. As for the unparameterized case, a parameterized subsumption $u \mid \varphi \sqsubseteq_Y \bigvee_{i\in I} v_i \mid \psi_i$ provides a relatively efficient, symbolic way of checking the parameterized inclusion $[\![u \mid \varphi]\!] \subseteq_Y [\![\bigvee_{i\in I} v_i \mid \psi_i]\!]$. Indeed, we have:

**Lemma 5.2** *Given pattern predicates $u \mid \varphi$ and $\bigvee_{i\in I} v_i \mid \psi_i$ with common parameters $Y$, if $T_{\Sigma/E\cup B} \models \varphi \Rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\ \{v_i\}_{i\in I},Y)} \psi_i\beta$, then $[\![u \mid \varphi]\!] \subseteq_Y [\![\bigvee_{i\in I} v_i \mid \psi_i]\!]$.*

The notions of parameterized intersection and parameterized containment will be used in Section 5.3.1 to reason about *parameterized* invariants and co-invariants, and in Section 5.4 to perform inferences in reachability logic.

## 5.3 CONSTRUCTOR-BASED REACHABILITY LOGIC

The constructor-based reachability logic we shall define is a logic to reason about reachability properties of the canonical reachability model $\mathcal{C}_\mathcal{R}$ of a topmost rewrite theory $\mathcal{R}$, where "topmost" captures the intuitive idea that all rewrites with the rules $R$ in $\mathcal{R}$ happen at the top of the term. Many rewrite theories of interest, including theories specifying distributed object-oriented systems and rewriting logic specifications of (possibly concurrent) programming languages, can be easily specified as topmost rewrite theories by a simple theory transformation (see, e.g., [78]). Besides satisfying the requirements in Definition 2.14, $\mathcal{R}$ should also satisfy the requirements in Definition 5.2 below.

**Definition 5.2 (Suitable Rewrite Theories)** *We say a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying the requirements in Definition 2.14 is* suitable for reachability analysis *or just* suitable *iff it satisfies the following additional conditions:*

1. *$(\Sigma, E \cup B)$ has a decomposition $(\Sigma, B, \vec{E})$ and a constructor decomposition $(\Omega, B_\Omega, \vec{E_\Omega})$ such that: (i) the equations $E_\Omega \cup B_\Omega$ are regular and there is a finitary $E_\Omega \cup B_\Omega$-unification algorithm*[5] *and (ii) the axioms $B_\Omega$ are linear.*

2. *$\Sigma$ has a sort State, the top sort of a connected component [State], and $\mathcal{R}$ is* topmost *for sort State in the sense that: (i) for rules $l \to r \in R$, $l$ and $r$ have sort State and (ii) for any $u \in T_\Omega(X)_{State}$ and any non-empty position $p$ in $u$, $u|_p \notin T_\Omega(X)_{State}$.*

3. *All rules $(l \to r \text{ if } \varphi) \in R$ have $l \in T_\Omega(X)$ and are* unforgetful,[6] *i.e. they satisfy $vars(l) \backslash (vars(r) \cup vars(\phi)) = \varnothing$.*

Requirements (1)–(3) in Definition 5.2 ensure that in the canonical reachability model $\mathcal{C}_\mathcal{R}$ if $[u] \to_\mathcal{R} [v]$ holds, then the $R, B$-rewrite $u \to_{R,B} u'$ such that $[u'!] = [v]$ *happens at the top* of $u$, i.e., uses a rewrite rule $l \to r \text{ if } \varphi \in R$ and a ground substitution $\sigma \in [Y \to T_\Omega]$, with $Y$ the rule's variables, such that $u =_{B_\Omega} l\sigma$ and $u' = r\sigma$. In the sequel, we assume that all rewrite theories satisfy the requirements in Definition 5.2, i.e., are *suitable for reachability analysis.* We are now ready to define the formulas of our constructor-based reachability logic for suitable theories $\mathcal{R}$.

**Definition 5.3 (Reachability Formulas)** *Let $\mathcal{R} = (\Sigma, E \cup B, R)$ be suitable. Recall the notion of an s-sorted constrained pattern predicate $PatPred(\Omega, \Sigma)_s$ in Definition 5.1. A* reachability formula *then has the form: $A \to^{\circledast} B$, with $A, B \in PatPred(\Omega, \Sigma)_{State}$, where $(\Omega, B_\Omega, \vec{E_\Omega})$ is the constructor decomposition of $(\Sigma, B, \vec{E})$ assumed in Definition 5.2. By definition, the* parameters *$Y$ of $A \to^{\circledast} B$ are the variables in the set $Y = vars(A) \cap vars(B)$, and $A \to^{\circledast} B$ is called* unparameterized *iff $Y = \varnothing$.*

The presentation of reachability logic in [27] considers *two* different semantics: (i) a *one-path* semantics, which we denote $\mathcal{R} \models^1 A \to^{\circledast} B$, and (ii) an *all-paths* semantics, which we denote $\mathcal{R} \models^\forall A \to^{\circledast} B$. Since the all-paths semantics is the most general and expressive, and the one-path semantics applies mostly to sequential systems, in this chapter we focus on the all-paths semantics.

The reachability logic in [26, 27] is based on *terminating* sequences of state transitions and is such that all reachability formulas are *vacuously true* when there are no terminating states. Our purpose is to extend reachability logic so as to be able to verify properties of general distributed systems specified as rewrite theories $\mathcal{R}$ which *may never terminate.* For this, as further explained in Section 5.3.1, we extend the rewrite theory $\mathcal{R}$ into a closely-related theory $\mathcal{R}_{stop}$ which does have terminating states. This allows a useful interpretation of reachability formulas, which have a non-vacuous meaning in $\mathcal{R}_{stop}$ and indirectly also a new meaning (the desired one) in the original non-terminating theory $\mathcal{R}$. Furthermore, we can generalize the satisfaction relation $\mathcal{R} \models^\forall A \to^{\circledast} B$ to a *relativized* satisfaction relation $\mathcal{R} \models^\forall_T A \to^{\circledast} B$, where $T$ is a constrained pattern predicate such that $[\![T]\!]$ is a subset of the set of terminating states.

The following terminological clarification may help the reader. In the canonical reachability model $\mathcal{C}_\mathcal{R}$ of a rewrite theory $\mathcal{R}$ we call a finite or infinite sequence of $\to_\mathcal{R}$-transitions *maximal* iff it cannot be extended.

[5] This is always guaranteed in practice if $(\Omega, B_\Omega, \vec{E_\Omega})$ is FVP and $B_\Omega$ has a finitary unification algorithm.

[6] Call a rule $l \to r \in R$ *forgetful* if it is not *unforgetful.* In the rewriting specification of an asynchronous fault-tolerant communication protocol where the state is specified as a multiset of objects (network nodes) and messages, the dropping of messages by the faulty environment can be modeled by the forgetful rule $M \to null$, where *null* is the empty multiset. For technical reasons, in reachability logic deduction it is useful to assume that all rewrite rules are unforgetful. But this entails no real loss of generality: any forgetful rule $l \to r \text{ if } \phi \in R$ with $vars(l) \backslash (vars(r) \cup vars(\phi)) = \{x_1, \ldots x_n\}$ can be replaced by the semantically equivalent unforgetful rule: $l \to r \text{ if } \phi \wedge x_1 = x_1 \wedge \ldots \wedge x_n = x_n$. For example, $M \to null$ can be replaced by $M \to null \text{ if } M = M$.

This can happen in exactly two ways; either: (i) the sequence is *infinite*, and is then called *non-terminating*, or (ii) the sequence is a *finite* sequence $[u] \rightarrow^{*}_{\mathcal{R}} [v]$ but it cannot be extended, i.e., $(\nexists[w])\ [v] \rightarrow_{\mathcal{R}} [w]$, and is then called *terminating*. $\mathcal{R}$ itself is called *never terminating* iff all maximal sequences in $\mathcal{C}_{\mathcal{R}}$ are infinite, and *terminating* iff they are all finite. In general, of course, $\mathcal{C}_{\mathcal{R}}$ may have both terminating and non-terminating sequences.

**Definition 5.4 ($T$-Terminating Sequence)** *Let $Term_{\mathcal{R}}$ denote the set of terminating states for theory $\mathcal{R}$, i.e., $Term_{\mathcal{R}} = \{[u] \in \mathcal{C}_{\mathcal{R},State} \mid (\nexists[v])\ [u] \rightarrow_{\mathcal{R}} [v]\}$. If $[\![T]\!] \subseteq Term_{\mathcal{R}}$, call $[u] \rightarrow^{*}_{\mathcal{R}} [v]$ a $T$-*terminating sequence *iff $[v] \in [\![T]\!]$. For reachability analysis purposes, we require that $T$ can be specified as a pattern predicate of the form $T = \bigvee_i t_i \mid \chi_i$, with $vars(\chi_i) \subseteq vars(t_i)$.*

In all the examples we present, the relation $\models^{\forall}_{T}$ is the standard relation $\models^{\forall}$, i.e., $[\![T]\!] = Term_{\mathcal{R}}$; but even in the standard case, giving an explicit specification of the set of terminating states is very useful for deduction purposes. Constructor-based techniques such as those proposed in [59] can be used to characterize the set $Term_{\mathcal{R}}$ of terminating states by means of a pattern predicate $T$ in many cases. In the relative case, where we just have an inclusion $[\![T]\!] \subseteq Term_{\mathcal{R}}$, we need to show that the containment $[\![T]\!] \subseteq Term_{\mathcal{R}}$ holds, which can often be achieved by showing, using unification and narrowing techniques, that no state $[w] \in [\![T]\!]$ can be rewritten at all by the rules in $\mathcal{R}$.

**Definition 5.5 (Semantics of Reachability Formulas)** *Given $T$ with $[\![T]\!] \subseteq Term_{\mathcal{R}}$, the* all-paths *satisfaction relation $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ asserting the satisfaction of the formula $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ in the canonical reachability model $\mathcal{C}_{\mathcal{R}}$ of a suitable rewrite theory $\mathcal{R}$ is defined as follows:*

*For $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ unparameterized, $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ holds iff for each $T$-terminating sequence $[u_0] \rightarrow_{\mathcal{R}} [u_1] \ldots [u_{n-1}] \rightarrow_{\mathcal{R}} [u_n]$ with $[u_0] \in [\![u \mid \varphi]\!]$ there exist $k$, $0 \leqslant k \leqslant n$ and $j \in J$ such that $[u_k] \in [\![v_j \mid \phi_j]\!]$. For $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ with parameters $Y$, $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ holds if $\mathcal{R} \models^{\forall}_{T} (u \mid \varphi)\rho \rightarrow^{\circledast} (\bigvee_{j \in J} v_j \mid \phi_j)\rho$ holds for each $\rho \in [Y \rightarrow T_{\Omega}]$.*

*Since a constrained pattern predicate is equivalent to a disjunction of atomic ones, we can define satisfaction on general reachability logic formulas as follows: $\mathcal{R} \models^{\forall}_{T} \bigvee_{1 \leqslant i \leqslant n} u_i \mid \varphi_i \rightarrow^{\circledast} A$ iff $\bigwedge_{1 \leqslant i \leqslant n} \mathcal{R} \models^{\forall}_{T} u_i \mid \varphi_i \rightarrow^{\circledast} A$, assuming same parameters $Y_i = vars(u_i \mid \varphi_i) \cap vars(A)$, i.e., $Y_i = Y_{i'}$ for $1 \leqslant i < i' \leqslant n$.*

$\mathcal{R} \models^{\forall}_{T} A \rightarrow^{\circledast} B$ is a *path-universal partial correctness assertion*: If state $[u]$ satisfies *precondition* $A$, then *midcondition* $B$ is satisfied *somewhere* along *each* $T$-terminating sequence from $[u]$, generalizing a Hoare formula $\{A\}\mathcal{R}\{B\}$, where $B$ is understood not just as a "midcondition," but as a "postcondition" satisfied by final states. To be consistent with the Hoare logic meaning of postconditions (see Section 5.3.2 for a generalized Hoare logic), we reserve the term *postcondition* for a midcondition $B$ in a reachability formula $A \rightarrow^{\circledast} B$ such that $[\![B]\!] \subseteq [\![T]\!]$.

**Implicit Quantification in Reachability Formulas**. Implicit in the above definition of satisfaction is the different way in which variables are quantified. It may be worthwhile making this explicit to clarify the *implicit universal and existential quantifications* involved in a (seemingly unquantified) reachability formula $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$. Let $U = vars(u \mid \varphi)$, $Z = vars(\bigvee_{j \in J} v_j \mid \phi_j)$, and $Y = U \cap Z$. Then, *all variables in* $U$ (and in particular all parameters in $Y$) *are universally quantified*, and *all variables in* $Z \backslash Y$ are *existentially quantified*, in the sense that $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ holds iff:

$$\begin{array}{l} \forall \gamma \in [U \to T_\Omega] \text{ s.t. } T_{\Sigma, E \cup B} \models \varphi\gamma \\ \qquad \forall\, [u_0] \to_{\mathcal{R}} [u_1] \ldots [u_{n-1}] \to_{\mathcal{R}} [u_n] \text{ s.t. } [u_0] = [(u\gamma)!] \wedge [u_n] \in [\![T]\!] \\ \qquad\quad \exists k \in \mathbb{N},\ 0 \leqslant k \leqslant n,\quad \exists j \in J \quad \exists \tau \in [Z \backslash Y \to T_\Omega] \\ \qquad\qquad \text{s.t. } [u_k] = [(v_j(\gamma|_Y \uplus \tau))!] \in [\![(v_j \mid \phi_j)\gamma|_Y]\!]. \end{array} \tag{5.11}$$

**Parameter Instantiation**. Assume again a reachability formula $u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ with $U = \mathit{vars}(u \mid \varphi)$, $Z = \mathit{vars}(\bigvee_{j \in J} v_j \mid \phi_j)$, and parameters $Y = U \cap Z$. In deductive reasoning, such a formula, and other formulas related to it, are often instantiated by a substitution $\alpha$ whose domain is a subset of $U$ and whose range is disjoint from $U \cup Z$. Then, $\alpha$ respects the formula's parameters in the expected way:

**Lemma 5.3** *(Parameter Instantiation Lemma). Under the above assumptions on $u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$, for any substitution $\alpha$ such that $\mathit{dom}(\alpha) \subseteq U$ and $\mathit{ran}(\alpha) \cap (U \cup Z) = \varnothing$, the formula $(u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j)\alpha$ has parameters $\mathit{vars}(\alpha(Y))$.*

Three simple classes of reachability formulas, called, respectively, *trivial*, *vacuous*, and *$T$-consistent*, play an important role in reachability logic deduction:

**Definition 5.6 (Trivial, Vacuous, $T$-consistent)** *Given a rewrite theory $\mathcal{R}$ with terminating states $T$ specified by the pattern predicate $T = \bigvee_i t_i \mid \chi_i$, a reachability formula $u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$, whose variables are without loss of generality assumed disjoint from those in $T$, and with (possibly empty) parameters $Y$ is called:*

1. trivial *iff* $[\![u \mid \varphi]\!] \subseteq_Y [\![\bigvee_{j \in J} v_j \mid \phi_j]\!]$.
2. vacuous *iff* $[\![u \mid \varphi]\!] = \varnothing$
3. $T$-consistent *iff* $(\forall i)(\forall \alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u, t_i))[\![(u \mid \varphi \wedge \chi_i)\alpha]\!] \subseteq_{\mathit{vars}(\alpha(Y))} [\![(\bigvee_{j \in J} v_j \mid \phi_j)\alpha]\!]$.

A trivial reachability formula $u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ is called so because all states in its precondition have already reached the midcondition, and therefore $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ trivially holds in 0 rewrite steps. A reachability formula whose precondition is empty is *vacuously* valid. Note that *vacuousness is a special case of triviality.* The meaning of a $T$-consistent formula can be best clarified by its negation: $u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ will be *$T$-inconsistent* iff there is a ground substitution $\rho$ of the parameters $Y$ and a final state $[w] \in [\![(u \mid \varphi)\rho]\!] \cap [\![T]\!]$ such that $[w] \notin [\![(\bigvee_{j \in J} v_j \mid \phi_j)\rho]\!]$. Therefore, $T$-consistency is a *necessary* condition for validity. It is also a sufficient condition when the states in the precondition are terminating states:

**Lemma 5.4** *If $u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ with parameters $Y$ is $T$-consistent and $[\![u \mid \varphi]\!] \subseteq \mathit{Term}_{\mathcal{R}}$, then $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$.*

Recall that in requirement (3) for a suitable rewrite theory $\mathcal{R}$ we assumed unforgetful topmost rewrite rules of the form $l \to r$ *if* $\phi$ with $l \in T_\Omega(X)$. For symbolic reasoning purposes it will be very useful to also require that $r \in T_\Omega(X)$. This can be done without any real loss of generality by means of a theory transformation[7] $\mathcal{R} \mapsto \hat{\mathcal{R}}$ defined as follows. If $\mathcal{R} = (\Sigma, E \cup B, R)$, then $\hat{\mathcal{R}} = (\Sigma, E \cup B, \hat{R})$, where the rules $\hat{R}$ are obtained from the rules $R$ by transforming each $l \to r \text{ } \mathit{if} \text{ } \phi$ in $R$ into the rule $l \to r' \text{ } \mathit{if} \text{ } \phi \wedge \hat{\theta}$, where: (i) $r'$ is the $\Omega$-*abstraction* of $r$ obtained by replacing each length-minimal position $p$ of $r$ such that $t|_p \notin T_\Omega(X)$ by a fresh variable $x_p$ whose sort is the least sort of $t|_p$, (ii) $\hat{\theta} = \bigwedge_{p \in P} x_p = t_p$, where $P$ is the set of all length-minimal positions in $r$ such that $t|_p \notin T_\Omega(X)$. Note that the transformation $\mathcal{R} \mapsto \hat{\mathcal{R}}$ preserves all suitable theory requirements (1)–(3). Its key semantic property can be expressed as follows:

**Lemma 5.5** *The canonical reachability models $\mathcal{C}_\mathcal{R}$ and $\mathcal{C}_{\hat{\mathcal{R}}}$ are* identical.

### 5.3.1 Invariants, Co-Invariants, and Never-Terminating Systems

The notion of an *invariant* makes sense for any transition system $\mathcal{S}$, that is, for any pair $\mathcal{S} = (S, \to_\mathcal{S})$ with $S$ its set of *states* and $\to_\mathcal{S} \subseteq S \times S$ its *transition relation*. Given a set of "initial states" $S_0 \subseteq S$, the set $\mathit{Reach}_\mathcal{S}(S_0)$ of states *reachable* from $S_0$ is defined as $\mathit{Reach}_\mathcal{S}(S_0) = \{s \in S \mid (\exists s_0 \in S_0) \; s_0 \to^*_\mathcal{S} s\}$, where $\to^*_\mathcal{S}$ denotes the reflexive-transitive closure of $\to_\mathcal{S}$. An invariant is a safety property about $\mathcal{S}$ with initial states $S_0$ and can be specified in two ways: (i) by a "good" property $P \subseteq S$, the *invariant*, that *always holds* from $S_0$, i.e., such that $\mathit{Reach}_\mathcal{S}(S_0) \subseteq P$, or (ii) as a "bad" property $Q \subseteq S$, the *co-invariant*, that *never holds* from $S_0$, i.e., such that $\mathit{Reach}(S_0) \cap Q = \varnothing$. Obviously, $P$ is an invariant iff $S \backslash P$ is a co-invariant. Sometimes it is easier to specify an invariant *positively*, as $P$, and sometimes *negatively*, as its co-invariant $S \backslash P$.

Invariants and co-invariants are much easier to prove if they are *inductive*. This can be expressed in terms of the notion of an $\mathcal{S}$-stable set. For $\mathcal{S} = (S, \to_\mathcal{S})$ a transition system, $U \subseteq S$ is *$\mathcal{S}$-stable* iff for each $s, s' \in S$, $(s \in U \wedge s \to_\mathcal{S} s') \Rightarrow s' \in U$. An invariant $P$ (resp. a co-invariant $Q$) for initial states $S_0$ is *inductive* iff $P$ is $\mathcal{S}$-stable (resp. $Q$ is $\mathcal{S}^{-1}$-stable, where $\mathcal{S}^{-1} = (S, (\to_\mathcal{S})^{-1})$). Equivalently, the following are equivalent:

1. $P$ is an inductive invariant (resp. $Q$ is an inductive co-invariant) for $S_0$.

2. $S_0 \subseteq P$ and $P$ is $\mathcal{S}$-stable (resp. $S_0 \cap Q = \varnothing$ and $Q$ is $\mathcal{S}^{-1}$-stable).

3. $S_0 \subseteq P$ and $P = \mathit{Reach}_\mathcal{S}(P)$ (resp. $S_0 \cap Q = \varnothing$ and $Q = \mathit{Reach}_{\mathcal{S}^{-1}}(Q)$).

All this is particularly relevant for the transition system $(\mathcal{C}_{\mathcal{R},\mathit{State}}, \to_\mathcal{R})$ associated to the canonical model $\mathcal{C}_\mathcal{R}$ of a rewrite theory $\mathcal{R}$. Here is an obvious question with a non-obvious answer. Suppose we have specified a distributed system as the canonical model $\mathcal{C}_\mathcal{R}$ of a suitable rewrite theory $\mathcal{R}$. Suppose further that we have specified constrained pattern predicates $S_0$ and $P$ (resp. and $Q$) and we want to prove that $[\![P]\!]$ (resp. $[\![Q]\!]$) is an *invariant* (resp. *co-invariant*) of the system $(\mathcal{C}_{\mathcal{R},\mathit{State}}, \to_\mathcal{R})$ from $[\![S_0]\!]$. Can we characterize such invariant, resp. co-invariant, property by means of *reachability formulas* and use the inference system of Section 5.4 to try to prove such formulas?

Suppose $\mathcal{R}$ specifies a *never-terminating system*, i.e., a system such that $\mathit{Term}_\mathcal{R} = \varnothing$. Many distributed systems are never-terminating. For example, QLOCK and other mutual exclusion protocols are never-terminating. Then the set $\mathit{Term}_\mathcal{R} = \varnothing$, and $\mathcal{R} \models^\forall_T A \to^\circledast B$ holds vacuously for *all* reachability formulas $A \to^\circledast B$ and *no* reachability formula can characterize an invariant (resp. co-invariant) over $\mathcal{R}$.

Nevertheless, reachability logic can indeed meaningfully reason about invariants and co-invariants of distributed systems, regardless of whether they are terminating, sometimes terminating, or never terminating.

[7] An even more general theory transformation $\mathcal{R} \mapsto \overline{\mathcal{R}}^{\Omega}_{\Sigma_1,l,r}$, used in some of the examples of Section 5.5, is presented in [43, 87].

We just need to first perform a simple *theory transformation*. To ease the exposition, we explain the transformation in case $\Omega$ has a single state constructor, say, $\langle \_, \ldots, \_ \rangle : s_1, \ldots, s_n \to \mathit{State}$. Extending to multiple constructors is straightforward.

**Invariant Theory Transformation**. The theory transformation has the form $\mathcal{R} \mapsto \mathcal{R}_{stop}$, where $\mathcal{R}_{stop}$ is obtained from $\mathcal{R}$ by just adding: (1) a new state constructor operator $[\_, \ldots, \_] : s_1, \ldots, s_n \to \mathit{State}$ to $\Omega$, and (2) a new rewrite rule $\mathit{stop} : \langle x_1{:}s_1, \ldots, x_n{:}s_n \rangle \to [x_1{:}s_1, \ldots, x_n{:}s_n]$ to $R$. Also, let $[\,]$ denote the pattern predicate $[x_1{:}s_1, \ldots, x_n{:}s_n] \mid \top$. Likewise, for any atomic constrained pattern predicate $B = \langle u_1, \ldots, u_n \rangle \mid \varphi$ we define the pattern predicate $[B] = [u_1, \ldots, u_n] \mid \varphi$ and extend this notation to any union $Q$ of atomic predicates.

Since $\langle \_, \ldots, \_ \rangle : s_1, \ldots, s_n \to \mathit{State}$ is the only state constructor, we can assume without loss of generality that any atomic constrained pattern predicate in $\mathcal{R}$ is semantically equivalent to one of the form $\langle u_1, \ldots, u_n \rangle \mid \varphi$. Likewise, any pattern predicate will be semantically equivalent to a union of atomic predicates of such form, called in *standard form*. Here is the main theorem:

**Theorem 5.1 (Invariants)** *For $S_0, P \in \mathit{PatPred}(\Omega,\Sigma)$ constrained pattern predicates in standard form with $\mathit{vars}(S_0) \cap \mathit{vars}(P) = \varnothing$, $[\![P]\!]$ is an invariant of $(\mathcal{C}_{\mathcal{R},\mathit{State}}, \to_{\mathcal{R}})$ from $[\![S_0]\!]$ iff $\mathcal{R}_{stop} \models^{\forall}_{[\,]} S_0 \to^{\circledast} [P]$.*

The notion of a *parametric invariant* can be reduced to the unparameterized one: if $Y = \mathit{vars}(S_0) \cap \mathit{vars}(P)$, then $[\![P]\!]$ is an *invariant* of $(\mathcal{C}_{\mathcal{R},\mathit{State}}, \to_{\mathcal{R}})$ from $[\![S_0]\!]$ *with parameters* $Y$ iff $\mathcal{R}_{stop} \models^{\forall}_{[\,]} S_0 \to^{\circledast} [P]$. That is, iff $[\![P\rho]\!]$ is an (unparameterized) invariant of $(\mathcal{C}_{\mathcal{R},\mathit{State}}, \to_{\mathcal{R}})$ from $[\![S_0\rho]\!]$ for each $\rho \in [Y {\to} T_{\Omega}]$. In this way, just by dropping the unparametricity requirement $\mathit{vars}(S_0) \cap \mathit{vars}(P) = \varnothing$ from the theorem's statement, Theorem 5.1 extends seamlessly to a reachability logic characterization of parametric invariants.

**Example 5.2 (Specifying Invariants for QLOCK)** *As an example, we consider how to specify invariants as reachability formulas using the QLOCK specification from Sections 5.1.5 and 5.2. Note that not only is QLOCK nonterminating: it is also* never *terminating. Thus, specifying any invariants as reachability formulas in the original theory is impossible. However, by applying the $\mathcal{R} \mapsto \mathcal{R}_{stop}$ theory transformation and Theorem 5.1, we can specify invariants by reachability formulas. Define the set of initial states containing only normal processes by the pattern predicate $S_0 = < n' \mid \varnothing \mid \varnothing \mid nil > \mid dupl(n') \mathrel{=\!\!\!|} tt$. Since QLOCK states have the form $< n \mid w \mid c \mid q >$, mutual exclusion means $|c| \leqslant 1$, which is expressible by the pattern predicate $< n \mid w \mid \varnothing \mid q > \vee < n \mid w \mid i \mid i \ ; \ q >$. We need also to ensure our multisets are actually* sets. *Thus, we define the constructor pattern predicates $P_1 = \big(< n \mid w \mid \varnothing \mid q > \mid dupl(n\ w) \neq tt\big)$ and $P_2 = \big(< n \mid w \mid i \mid i \ ; \ q > \mid dupl(n\ w\ i) \neq tt\big)$, so that the pattern predicate $P = P_1 \vee P_2$ specifies mutual exclusion. By Theorem 5.1, QLOCK ensures mutual exclusion from $[\![S_0]\!]$ iff $\mathcal{R}_{stop} \models^{\forall}_{[\,]} S_0 \to^{\circledast} [P]$ where here $[P]$ is $[P_1] \vee [P_2]$, i.e. $\big([\ n \mid w \mid \varnothing \mid q\ ] \mid dupl(n\ w) \neq tt \vee [\ n \mid w \mid i \mid i \ ; \ q\ ] \mid dupl(n\ w\ i) \neq tt\big)$.*

As pointed out above, proving inductive invariants is much easier than proving non-inductive ones. The following theorem provides a precise characterization of parametric invariants in reachability logic. Of the three equivalent characterizations we have given of inductive invariants, it uses Characterization (3). This theorem can, for example, be applied to prove the mutual exclusion of QLOCK.

**Theorem 5.2 (Parametric Inductive Invariants)** *Let $S_0, P \in \mathit{PatPred}(\Omega, \Sigma)$ both be constrained pattern predicates in standard form with $\mathit{vars}(S_0) \cap \mathit{vars}(P) = Y$. Then $[\![P]\!]$ is a parametric inductive invariant of $(\mathcal{C}_{\mathcal{R},\mathit{State}}, \to_{\mathcal{R}})$ from $[\![S_0]\!]$ with parameters $Y$ iff: (i) $[\![S_0]\!] \subseteq_Y [\![P]\!]$ (see Section 5.2.1), and (ii)*

$\mathcal{R}_{stop} \models^{\forall}_{[]} P \rightarrow^{\circledast} [P\sigma]$, *where* $\sigma$ *is a sort-preserving bijective renaming of variables such that* $\sigma$ *is the identity on* $Y$ *and* $vars(P) \cap vars(P\sigma) = Y$*.*

This leaves still open the question of whether reachability logic could directly express Characterization (2) of inductive invariants in terms of stable sets. The answer is *yes*, provided we assume without loss of generality, thanks to Lemma 5.5, that $\mathcal{R} = \hat{\mathcal{R}}$ and we use a slightly different theory transformation. Namely, we use the transformation $\mathcal{R} \mapsto \mathcal{R}_{stop1}$, where $\mathcal{R}_{stop1}$ is the theory obtained from $\mathcal{R}_{stop}$ by replacing the rules $R$ from $\mathcal{R} = \hat{\mathcal{R}}$ by the set of rules $[R]$ obtained by replacing each rule $l \rightarrow r \;\; if \;\; \phi$ in $R$ by the rule $l \rightarrow [r] \;\; if \;\; [\phi]$ where, by convention, (i) if the constructor term $r$ has the from $\langle v_1, \ldots, v_n \rangle$, then $[r] = [v_1, \ldots, v_n]$ and $[\phi] = \phi$, and (ii) otherwise, $r$ must be a variable $S$ of sort *State*, and then $[r] = [x_1, \ldots, x_n]$, where the variables $x_1, \ldots, x_n$ are *fresh* of the input sorts $s_1, \ldots, s_n$ for $[\_, \ldots, \_]$, and where $[\phi] = \phi \wedge S = \langle x_1, \ldots, x_n \rangle$. The proof of the following corollary uses Characterization (2) and, being totally analogous to, and even simpler than, that of Theorem 5.2 is left to the reader.

**Corollary 5.1 (Parametric Inductive Invariants)** *Assume* $\mathcal{R} = \hat{\mathcal{R}}$ *and also let* $S_0, P \in PatPred(\Omega, \Sigma)$ *be constrained pattern predicates in standard form with* $vars(S_0) \cap vars(P) = Y$*. Then* $[\![P]\!]$ *is a parametric inductive invariant from* $[\![S_0]\!]$ *with parameters* $Y$ *for* $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ *iff: (i)* $[\![S_0]\!] \subseteq_Y [\![P]\!]$ *(see Section 5.2.1), and (ii)* $\mathcal{R}_{stop1} \models^{\forall}_{[]} P \rightarrow^{\circledast} [P\sigma]$, *where* $\sigma$ *is a sort-preserving bijective renaming of variables such that* $\sigma$ *is the identity on* $Y$ *and* $vars(P) \cap vars(P\sigma) = Y$*.*

What is attractive about Corollary 5.1 is that $\mathcal{R}_{stop1}$ is a very simple theory: in $\mathcal{R}_{stop1}$ all $\mathcal{R}$-terms of sort *State* terminate and all their associated terminating sequences have length 1. It is also useful to point out that the parametric inclusion $[\![S_0]\!] \subseteq_Y [\![P]\!]$ is semantically equivalent to $\mathcal{R}_{stop1} \models^{\forall}_{[]} [S_0] \rightarrow^{\circledast} [P\sigma]$, since this will allow us to use the SUBSUMPTION inference rule in Section 5.4 to discharge this proof obligation.

Let us now turn to the case of inductive co-invariants. Suppose we have specified constrained pattern predicates $S_0$ and $Q$ and we want to prove that $[\![Q]\!]$ is an *inductive co-invariant* of the system $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$. Can this property be characterized by some reachability formula or formulas? More generally, can we characterize in reachability logic when $Q$ is a *parametric* inductive co-invariant from $[\![S_0]\!]$? Parametric co-invariants are entirely analogous to parametric invariants. Given constrained pattern predicates $S_0, Q \in PatPred(\Omega, \Sigma)$ in standard form, with $Y = vars(S_0) \cap vars(Q)$, we call $Q$ a *parametric co-invariant* in $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ for initial states $[\![S_0]\!]$ *with parameters* $Y$ iff for each $\rho \in [Y \rightarrow T_\Omega]$ $[\![Q\rho]\!]$ is an (unparameterized) co-invariant in $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ for initial states $[\![S_0\rho]\!]$. The key idea to characterize inductive co-invariants in reachability logic is to use the rules of $\mathcal{R}$ *backwards*. Assume, without loss of generality thanks to Lemma 5.5, that $\mathcal{R} = \hat{\mathcal{R}}$. Then, if $\mathcal{R} = (\Sigma, E \cup B, R)$, define $\mathcal{R}^{-1} = (\Sigma, E \cup B, R^{-1})$, where $R^{-1} = \{r \rightarrow l \; if \; \varphi \mid (l \rightarrow r \; if \; \varphi) \in R\}$. Then, if $\mathcal{R}$ satisfies the suitability conditions (1)–(3) and, assuming the rules $R^{-1}$ are ground coherent[8] with the equations $E$ modulo $B$ and have been made unforgetful if necessary by adding trivial equalities for the forgotten variables to their conditions, then $\mathcal{R}^{-1}$ also satisfies the suitability conditions (1)–(3). Here is the main theorem characterizing parametric inductive co-invariants in reachability logic (the unparameterized case is the case $Y = \varnothing$):

**Theorem 5.3 (Parametric Inductive Co-invariants)** *Assume* $\mathcal{R} = \hat{\mathcal{R}}$ *and let* $S_0, Q \in PatPred(\Omega, \Sigma)$ *be constrained pattern predicates in standard form with* $vars(S_0) \cap vars(Q) = Y$*. Then* $[\![Q]\!]$ *is a parametric*

[8] Ground coherence of $R^{-1}$ may be problematic because, while a rule's lefthand side $l$ is assumed to be a constructor term, that assumption does not hold in general for its righthand side $r$. However, its *does* if we assume (without loss of generality thanks to Lemma 5.5) that $\mathcal{R} = \hat{\mathcal{R}}$. Coherence-type critical pairs almost never arise in practice between a constructor-based rule and equations $E$. Therefore, assuming $\mathcal{R} = \hat{\mathcal{R}}$, the ground coherence assumption for $R^{-1}$ is very reasonable.

*inductive co-invariant in* $(\mathcal{C}_{\mathcal{R},State}, \to_{\mathcal{R}})$ *for initial states* $[\![S_0]\!]$ *with parameters* $Y$ *iff: (i)* $[\![S_0]\!] \cap_Y [\![Q]\!] = \varnothing$ *(see Section 5.2.1), and (ii)* $(\mathcal{R}^{-1})_{stop} \models^{\forall}_{[]} Q \to^{\circledast} [Q\sigma]$*, where* $\sigma$ *is a sort-preserving bijective renaming of variables such that* $\sigma$ *is the identity on* $Y$ *and* $vars(Q) \cap vars(Q\sigma) = Y$*.*

In complete analogy with Corollary 5.1 we get the following corollary in terms of Characterization (2), whose proof simplifies and follows closely that of Theorem 5.3 and is left to the reader.

**Corollary 5.2 (Parametric Inductive Co-invariants)** *Assume* $\mathcal{R} = \hat{\mathcal{R}}$ *and let* $S_0, Q \in PatPred(\Omega, \Sigma)$ *be constrained pattern predicates in standard form with* $vars(S_0) \cap vars(Q) = Y$*. Then* $[\![Q]\!]$ *is a parametric inductive co-invariant in* $(\mathcal{C}_{\mathcal{R},State}, \to_{\mathcal{R}})$ *for initial states* $[\![S_0]\!]$ *with parameters* $Y$ *iff: (i)* $[\![S_0]\!] \cap_Y [\![Q]\!] = \varnothing$ *(see Section 5.2.1), and (ii)* $(\mathcal{R}^{-1})_{stop1} \models^{\forall}_{[]} Q \to^{\circledast} [Q\sigma]$*, where* $\sigma$ *is a sort-preserving bijective renaming of variables such that* $\sigma$ *is the identity on* $Y$ *and* $vars(Q) \cap vars(Q\sigma) = Y$*.*

### 5.3.2 Relationships to Hoare Logic and Universally Quantified LTL

It is both natural and helpful to compare reachability logic to other property logics such as Hoare logic or linear time temporal logic (LTL). Let us begin with Hoare logic [89].

**Relationship to Hoare Logic**. A Hoare logic is traditionally associated to a programming language; but the desired comparison should apply not just to programming languages but to any systems specifiable by topmost rewrite theories. This suggests defining Hoare logic in this more general setting.

**Definition 5.7 (Hoare Logic)** *Let* $\mathcal{R} = (\Sigma, E \cup B, R)$ *be a suitable theory, and let* $\Omega$ *be its constructor subsignature. A* Hoare triple *for* $\mathcal{R}$ *is then a triple of the form:*

$$\{A\}\ \mathcal{R}\ \{B\} \tag{5.12}$$

*where* $A, B \in PatPred(\Omega, \Sigma)_{State}$*. Let* $Y = vars(A) \cap vars(B)$*. By definition, when* $Y = \varnothing$*, a Hoare triple* $\{A\}\ \mathcal{R}\ \{B\}$ *is satisfied by the initial reachability model* $\mathcal{T}_{\mathcal{R}}$*, denoted* $\mathcal{T}_{\mathcal{R}} \models \{A\}\ \mathcal{R}\ \{B\}$*, iff for each* $[u] \in [\![A]\!]$ *and each terminating sequence* $[u] \to_{\mathcal{R}}![v]$*,* $[v] \in [\![B]\!]$*. If* $Y \neq \varnothing$*, then* $\mathcal{T}_{\mathcal{R}} \models \{A\}\ \mathcal{R}\ \{B\}$ *iff* $\mathcal{T}_{\mathcal{R}} \models \{A\rho\}\ \mathcal{R}\ \{B\rho\}$ *for each* $\rho \in [X \to T_{\Omega}]$*.*

Since the rewriting logic semantics of a programming language $\mathcal{L}$ can be specified by a topmost rewrite theory $\mathcal{R}_{\mathcal{L}}$, the *standard* Hoare logic for $\mathcal{L}$ becomes the special case where in the above notation we represent a Hoare triple $\{\varphi\}\ p\ \{\psi\}$ as the Hoare triple $\{\langle p : init\rangle \mid \widetilde{\varphi}\}\ \mathcal{R}_{\mathcal{L}}\ \{\langle skip : S\rangle \mid \widetilde{\psi}\}$, where *init* is the initial program state, of sort *ProgState*, *skip* is the empty program continuation, and where *configurations* of a program (or, more generally, a continuation) $p$ and a program state $S$ are represented as pairs $\langle p : S\rangle$. Explaining how the QF $\Sigma_{\mathcal{L}}$-formulas $\widetilde{\varphi}$ and $\widetilde{\psi}$ are derived from the original $\varphi$ and $\psi$ is essentially straightforward, but becomes complicated by the regrettable systematic confusion of *program* variables with *mathematical* variables in $\varphi$ and $\psi$. This can be best illustrated with an example. Consider the Hoare triple $\{n \geqslant 0\}$ `x :=` $n$ `; factp` $\{\mathtt{y} = n!\}$, which specifies that a factorial program `factp` with its variable `x` initialized to the integer $n \geqslant 0$ will have upon termination the value $n!$ stored in its variable `y`. For $\mathcal{R}_{\mathcal{L}}$ this can be expressed as the Hoare triple $\{\langle \mathtt{x}\ \mathtt{:=}\ n\ \mathtt{;}\ \mathtt{factp} : init\rangle \mid n \geqslant 0\}\ \mathcal{R}_{\mathcal{L}}\ \{\langle skip : S\rangle \mid S[\mathtt{y}] = n!\}$, where $S$ is a variable of sort *ProgState* and $S[v]$ is an auxiliary function extracting the value in state $S$ of program variable $v$. Of course, conversely, a Hoare triple $\{A\}\ \mathcal{R}\ \{B\}$ has also in a sense a *standard* interpretation, since we can view $\mathcal{R}$ as a *program* in a rewriting logic language with user-definable data types such as Maude.

The comparison with reachability logic is now straightforward: Hoare logic is essentially a sublogic of reachability logic, namely, in a Hoare triple $\{A\}\ \mathcal{R}\ \{B\}$, since $B$ is a *postcondition*, we may assume without loss of generality that $[\![B]\!] \subseteq [\![T]\!] = \mathit{Term}_{\mathcal{R}}$. Then, $\{A\}\ \mathcal{R}\ \{B\}$ is just syntactic sugar for the reachability formula $A \rightarrow^{\circledast} B$. Of course, the Hoare triple $\{A\}\ \mathcal{R}\ \{B\}$ is parametric with parameters $Y$ iff $A \rightarrow^{\circledast} B$ is so. Indeed, we then have:

$$\mathcal{T}_{\mathcal{R}} \models \{A\}\ \mathcal{R}\ \{B\} \quad\Leftrightarrow\quad \mathcal{R} \models^{\forall}\ A \rightarrow^{\circledast} B. \tag{5.13}$$

When the above comparison is applied to programming languages (see also [90]), it can be easy to miss the obvious, namely, the two crucial advantages that reachability logic has in this comparison. Besides being more general than Hoare logic and having abilities comparable to those of separation logic [91] to express and verify—through matching modulo associative-commutative axioms $B$—the properties of heap-intensive programs, the two crucial advantages of reachability logic are that:

1. unlike Hoare logic, reachability logic is *language-generic*; that is, instead of having to tailor a different Hoare logic for each different programming language, the need for language-specific Hoare rules completely evaporates:[9] only reachability logic's few inference rules (see Section 5.4), which are rewrite-theory-generic and, *a fortiori*, programming-language-generic, are needed; and

2. there is no need whatsoever for defining a so-called *axiomatic semantics* and proving it correct with respect to an *operational semantics*, which is crucially needed in the Hoare logic approach: all that is needed is the simple, theory-generic semantics of reachability logic given in Definition 5.5, which reduces it to the, again simple and generic, rewriting logic semantics of the rewrite theory $\mathcal{R}_{\mathcal{L}}$ defining the semantics of language $\mathcal{L}$ [92, 4].

It is even quite possible to miss the obvious *pragmatic consequences* of advantages (1)–(2). Developing a Hoare logic axiomatic semantics for a real programming language, say, Java or C, as opposed to a toy one, is a big effort requiring careful formalization and typically resulting in a large number of Hoare rules. But, relatively speaking, this is actually the easiest part of the job. The real challenge is to *prove* that such an axiomatic semantics is *correct* with respect to an operational semantics. This can be a daunting task, and sometimes even an impossible one due to the absence of a complete operational semantics for the language in question. For example, not until [93] was a complete operational semantics for C given, as a rewrite theory expressed in $\mathbb{K}$. As a consequence, some Hoare logics are never proved correct, so their trustworthiness becomes anybody's guess. The fact that in reachability logic a *single semantic object*, namely the rewrite theory $\mathcal{R}_{\mathcal{L}}$, is needed, and that this semantic object is *executable*, becomes a big pragmatic advantage.

**Relationship to LTL**. The comparison with LTL requires making explicit the atomic predicates and the Kripke structure $\mathcal{K}_{\mathcal{R}}$ associated to a suitable rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ on which the comparison is based. The atomic predicates are $\mathit{PatPred}(\Omega, \Sigma)_{\mathit{State}}$, and $\mathcal{K}_{\mathcal{R}}$ is the Kripke structure $\mathcal{K}_{\mathcal{R}} = (C_{\Sigma/E,B,\mathit{State}}, (\rightarrow_{\mathcal{R}})^{\bullet}, L_{\mathcal{R}})$, where the relation $(\rightarrow_{\mathcal{R}})^{\bullet}$ is the totalization of the one-step rewrite relation and $L_{\mathcal{R}}$ is the labeling function:

$$C_{\Sigma/E,B,\mathit{State}} \ni [u] \mapsto \{A \in \mathit{PatPred}(\Omega,\Sigma)_{\mathit{State}} \mid [u] \in [\![A]\!]\} \in \mathcal{P}(\mathit{PatPred}(\Omega,\Sigma)_{\mathit{State}}). \tag{5.14}$$

Note the useful fact that $\mathcal{K}_{\mathcal{R}}$ can give semantics not only to *propositional* LTL formulas $\varphi$, but also to *universal quantifications* $(\forall Y)\,\varphi$ of propositional LTL formulas $\varphi$, where $Y$ is a (possibly empty) finite set

[9] See [27] for strong evidence about the advantages of the language-generic nature of reachability logic applied to programming languages within the $\mathbb{K}$ framework.

of variables typed in the signature $\Sigma$ of $\mathcal{R}$. Indeed, we can *define*, for each $[u] \in C_{\Sigma/E,B,State}$,

$$\mathcal{K}_{\mathcal{R}}, [u] \models_{LTL} (\forall Y)\, \varphi \quad \Leftrightarrow_{def} \quad \forall \rho \in [Y \rightarrow T_{\Omega}]\; \mathcal{K}_{\mathcal{R}}, [u] \models_{LTL} \varphi\rho. \tag{5.15}$$

The comparison with LTL then becomes straightforward: reachability logic is essentially a sublogic of quantified LTL: a reachability formula $A \rightarrow^{\circledast} B$ with parameters $Y$ is syntactic sugar for the LTL formula $(\forall Y)\ A \rightarrow (\Diamond(B) \vee \Box en_{\mathcal{R}})$, where if $R = \{l_i \rightarrow r_i \textit{ if } \varphi_i\}_{i \in I}$, then $en_{\mathcal{R}}$ is the "enabledness" pattern predicate $en_{\mathcal{R}} = \bigvee_{i \in I} l_i \mid \varphi_i$. Indeed, we have:

$$\mathcal{R} \models^{\forall} A \rightarrow^{\circledast} B \quad \Leftrightarrow \quad \mathcal{K}_{\mathcal{R}} \models_{LTL} (\forall Y)\ A \rightarrow (\Diamond(B) \vee \Box en_{\mathcal{R}}). \tag{5.16}$$

Of course, when the semantics of $A \rightarrow^{\circledast} B$ is relativized to a pattern predicate $T$ of terminating states, we get instead the LTL formula $(\forall Y)\ A \rightarrow (\Diamond(B) \vee \Box\neg T)$.

Note that, thanks to the results in Section 5.3.1, reachability logic can also express universal LTL *safety formulas* of the form: $(\forall Y)\ A \rightarrow \Box B$ (with $A, B \in \mathit{PatPred}(\Omega, \Sigma)_{State}$ and $Y = \mathit{vars}(A) \cap \mathit{vars}(B)$), since we have:

$$\mathcal{R}_{stop} \models^{\forall}_{[\,]} A \rightarrow^{\circledast} [B] \quad \Leftrightarrow \quad \mathcal{K}_{\mathcal{R}} \models_{LTL} (\forall Y)\ A \rightarrow \Box B. \tag{5.17}$$

Furthermore, reachability logic can also express universal LTL *stability formulas* of the form: $(\forall Y)\ B \rightarrow \bigcirc B$, which are very useful for specifying and proving parametric inductive invariants. Indeed, we have:

$$\mathcal{R}_{stop1} \models^{\forall}_{[\,]} B \rightarrow^{\circledast} [B] \quad \Leftrightarrow \quad \mathcal{K}_{\mathcal{R}} \models_{LTL} (\forall Y)\ B \rightarrow \bigcirc B. \tag{5.18}$$

While constructor-based reachability logic can only express an (admittedly quite useful) subset of (quantified) LTL properties, this is compensated for by other advantages. For example, as shown in Section 5.4, reachability logic enjoys a built-in notion of *circularity* that is very useful for reasoning about repetitive behavior in systems. As another example, since Kripke models have no native notion of constructor, the symbolic methods extensively exploited in this chapter cannot be used as generic (quantified) LTL proof methods for arbitrary Kripke structures. However, an interesting question for future research is how the symbolic proof methods presented in this chapter could be extended to a bigger fragment of LTL for Kripke structures of the form $\mathcal{K}_{\mathcal{R}}$.

In summary, we can close our comparisons with Hoare logic and with quantified LTL by remarking that:

1. In comparison with Hoare logic, constructor-based reachability logic amounts to a vast *generalization* of an already highly expressive logic in *three* different dimensions: (i) from programming languages to rewrite theories which can specify *both* programming languages and distributed system designs; (ii) from *language-specific* Hoare logics that have to be hand crafted and proved sound for each programming language to a *rewrite theory generic* logic whose soundness is proved once and for all; and (iii) from *pre-post condition* properties to considerably more general and expressive *pre-mid condition* properties.

2. In comparison with quantified LTL the key point is that, not only are *safety properties* such as *parametric* invariants of the form $(\forall Y)\ A \rightarrow \Box B$ and *parametric* stability properties $(\forall Y)\ B \rightarrow \bigcirc B$ supported, but so are also *parametric* eventuality properties such as $(\forall Y)\ A \rightarrow (\Diamond(B) \vee \Box en_{\mathcal{R}})$, stating that all terminating paths starting at $A$ eventually reach $B$ for each ground instantiation of the parameters $Y$.

## 5.4 REACHABILITY LOGIC'S INFERENCE SYSTEM

We present our inference system for all-path reachability logic, parametric on a suitable rewrite theory $\mathcal{R}$ with unforgetful rules $R = \{l_j \to r_j \text{ if } \phi_j\}_{j \in J}$ such that $l_j, r_j \in T_\Omega(X)$, $j \in J$. *Variables of rules in $R$ are always assumed disjoint from variables in reachability formulas*; this can be ensured by renaming. The inference system has three proof rules: (i) the Subsumption proof rule discharges trivial formulas (recall Definition 5.6) by means of vacuousness or subsumption checks; (ii) the Step$^\forall$ proof rule allows taking one step of (symbolic) rewriting along all paths according to the rules in $\mathcal{R}$; and (iii) the Axiom proof rule allows the use of a trusted reachability formula to summarize multiple rewrite steps, and thus to handle repetitive behavior.

The proof rules derive sequents of the form $[\mathcal{A},\ \mathcal{C}] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, which are always checked for $T$-consistency, where $\mathcal{A}$ and $\mathcal{C}$ are finite sets of $T$-consistent reachability formulas and $T$ is a pattern predicate defining a set of $T$-terminating ground states. Formulas in $\mathcal{A}$ are called *axioms* and those in $\mathcal{C}$ are called *circularities.* We furthermore assume that in all reachability formulas $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ we have $vars(\psi_i) \subseteq vars(v_i) \cup vars(u \mid \varphi)$ for each $i$. According to the implicit quantification of the semantic relation $\models_T^\forall$ this means that any variable in $\psi_i$ is either universally quantified and comes from the precondition $u \mid \varphi$, or is existentially quantified and comes from $v_i$ only. This property is an invariant preserved by the three inference rules.

Proofs always begin with a set $\mathcal{C}$ of $T$-consistent formulas that we want to *simultaneously* prove, so that the proof effort only succeeds if *all* formulas in $\mathcal{C}$ are eventually proved. $\mathcal{C}$ contains the main properties we want to prove as well as any (as yet *unproved*) auxiliary lemmas that may be needed to carry out the proof. We can also use an additional set $\mathcal{L}$ of *already proved*, and therefore valid, lemmas as *axioms* that are always available for use. In such case, the initial set of goals we want to prove is $[\mathcal{L},\ \mathcal{C}] \vdash_T \mathcal{C}$, which is a shorthand for the set of goals $\{[\mathcal{L},\ \mathcal{C}] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i \bigm| (u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i) \in \mathcal{C}\}$. Thus, we start only with the *already proved* lemmas $\mathcal{L}$ as axioms, but we shall be able to also use all the formulas in $\mathcal{C}$ *as axioms* in their own derivation *after* taking at least one step with the rewrite rules in $\mathcal{R}$ using the Step$^\forall$ rule.

A very useful feature of the inference system is that sequents $[\mathcal{L},\ \mathcal{C}] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, whose formulas $\mathcal{C}$ have been *postulated* (as the conjectures we want to prove) but not yet justified, are transformed by Step$^\forall$ into sequents of the form $[\mathcal{L} \cup \mathcal{C},\ \varnothing] \vdash_T\ u' \mid \varphi' \to^{\circledast} \bigvee_i v_i' \mid \psi_i'$, where now the formulas in $\mathcal{C}$ *can be assumed valid*, and can be used in derivations with the Axiom rule.

**Example 5.3 (Conjectures for QLOCK's Mutual Exclusion)** *By Theorem 5.2, the mutual exclusion of QLOCK can be verified as an inductive invariant by: (i) using pattern subsumption to check the trivial inclusion $[\![S_0]\!] \subseteq [\![P]\!]$, and (ii) proving $\mathcal{R}_{stop} \models_{[]}^\forall P \to^{\circledast} [P\sigma]$, where $\sigma$ is a sort-preserving bijective renaming of variables such that $vars(P) \cap vars(P\sigma) = \varnothing$. For QLOCK, we had the initial state $S_0 = <n \mid \varnothing \mid \varnothing \mid nil> \mid dupl(n) \neq tt$ and invariant defined by pattern predicate $P = P_1 \vee P_2$ where $P_1 = \big(<n \mid w \mid \varnothing \mid q> \mid dupl(n\ w) \neq tt\big)$ and $P_2 = \big(<n \mid w \mid i \mid i\ ;\ q> \mid dupl(n\ w\ i) \neq tt\big)$. Since $P$ is a disjunction, in our inference system, the formula $P \to^{\circledast} [P\sigma]$ naturally splits into two corresponding reachability formulas $P_1 \to^{\circledast} [P\sigma]$ and $P_2 \to^{\circledast} [P\sigma]$ shown below:*

$$<n \mid w \mid \varnothing \mid q> \mid \psi \to^{\circledast} [<n' \mid w' \mid i' \mid i'\ ;\ q'> \mid \varphi' \vee <n' \mid w' \mid \varnothing \mid q'> \mid \psi'] \tag{5.19}$$

$$<n \mid w \mid i \mid i\ ;\ q> \mid \varphi \to^{\circledast} [<n' \mid w' \mid i' \mid i'\ ;\ q'> \mid \varphi' \vee <n' \mid w' \mid \varnothing \mid q'> \mid \psi'] \tag{5.20}$$

*where $\varphi = dupl(n\ w\ i) \neq tt$, $\psi = dupl(n\ w) \neq tt$, and $\varphi', \psi'$ are their obvious renamings. More generally,* *if our invariant is of the form* $P = \bigvee_{i \in I} P_i$*, then we have initial formulas to be proved* $\mathcal{C} = \{P_j \rightarrow^{\circledast} \bigvee_{i \in I}[P_i \sigma]\}_{j \in I}$.

Recall from Definition 5.6 that a $T$-inconsistent formula is invalid. Therefore, proof goals or subgoals involving any such formulas are nonsense. Before explaining in detail our inference system we explain the requirement of restricting all inferences to $T$-consistent goals. This is an *invariant* of the inference system that is assumed and that must be ensured before applying any inference rule. To maintain this invariant, our implementation—indeed, *any* implementation—must perform a $T$-consistency check before applying any inference step.

**The Importance of Checking $T$-Consistency**. Since any $T$-inconsistent formula is invalid and would therefore invalidate any further proof attempts based on it, all formulas in $\mathcal{C}$ and any further sequents derived by the inference system are always checked for $T$-consistency using parameterized subsumption, and *if the check can show the formula $T$-inconsistent, the user is immediately notified, and the proof search is abandoned.*

However, since, as noted in Section 5.2, not all set containments between pattern predicates can be checked by parameterized subsumption, as soon as a reachability formula $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ is encountered such that: (i) the set intersection $[\![u \mid \varphi]\!] \cap [\![T]\!]$ *cannot be shown to be empty*, and (ii) any of the set containments $[\![(u \mid \varphi \wedge \chi_j)\alpha]\!] \subseteq_{vars(\alpha(Y))} [\![(\bigvee_{j \in J} v_j \mid \phi_j)\alpha]\!]$ in the definition of $T$-consistency *cannot be established by parameterized subsumption*, the formula is declared *$T$-dubious*. In our implementation these $T$-dubious formulas are immediately indicated to the user, who is then given two options: (a) to continue the proof effort leaving the check of either: (i) the emptiness of $[\![u \mid \varphi]\!] \cap [\![T]\!]$, or (ii) the set inclusions $[\![(u \mid \varphi \wedge \chi_j)\alpha]\!] \subseteq_{vars(\alpha(Y))} [\![(\bigvee_{j \in J} v_j \mid \phi_j)\alpha]\!]$ as a *proof obligation* to be subsequently discharged, or (b) abandon the proof search in case the $T$-dubious formula is deemed to be $T$-inconsistent. In summary:

> *All formulas ever encountered or produced by the inference system should be automatically checked for $T$-consistency, so that if they are shown or deemed to be $T$-inconsistent, the proof search is abandoned; otherwise, the proof obligations essential for showing the $T$-consistency of a $T$-dubious formula are displayed and must be later discharged by the user.*

The *reasons* for performing the $T$-consistency check on reachability goals can be explained as follows. Any reachability goal is either $T$-consistent or $T$-inconsistent. But if it is $T$-inconsistent, it is then *invalid*. Therefore, detecting $T$-inconsistent goals is very useful for three complementary reasons:

1. Since all goals in any correct proof tree must be valid and therefore $T$-consistent, checking that all generated goals are $T$-consistent is a very useful *invariant* to be maintained along the proof search. For this reason, as explained later, $T$-consistency is made into a basic requirement of any proof goal and any correct proof tree. Indeed, the $T$-consistency requirement on proof trees is explicitly used in the proof of Theorem 5.4.

2. As shown by an example in Section 5.4.2, the Axiom inference rule is so powerful that, if unwisely used, it can generate invalid, and indeed $T$-inconsistent, subgoals *from* valid ones.[10] This is a further reason to always check that all goals are $T$-consistent.

[10] This of course can happen for many perfectly correct inference rules where some formulas have to be *guessed*. For example, to prove an implication $A \Rightarrow C$ we may apply a chain inference rule by guessing a middle formula $B$ to try to reduce the proof of $A \Rightarrow C$ to that of the subgoals $A \Rightarrow B$ and $B \Rightarrow C$. But a bad choice of $B$ may make either $A \Rightarrow B$ or $B \Rightarrow C$ invalid, while the original goal $A \Rightarrow C$ may be perfectly valid. As we shall see, when using the Axiom rule, the "middle formulas" guessed are instances of patterns in the midcondition of the chosen axiom formula.

3. As soon as a $T$-inconsistent goal is detected, no proof of the original set of goals is possible; therefore, the user should be immediately notified and the proof search should be stopped.

Let us first explain the SUBSUMPTION inference rule. Its purpose is to discharge goals that are trivial formulas in the sense of Definition 5.6, and therefore valid. It is a conditional rule of the form:

**Subsumption**

$$\frac{}{[\mathcal{A},\ \mathcal{C}] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i} \tag{5.21}$$

subject to the condition of showing that $u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \psi_j$ is a *trivial* formula by either: (i) showing that $\varphi$ is *unsatisfiable*[11] in $T_{\Sigma/E \cup B}$ (the vacuousness subcase), or (ii) checking the parameterized subsumption condition $u \mid \varphi \sqsubseteq_Y \bigvee_{j \in J} v_j \mid \psi_j$, where $Y$ are the formula's parameters. As explained in Section 5.3, parameterized subsumption is a sufficient condition for proving the parametric inclusion $[\![u \mid \varphi]\!] \subseteq_Y [\![\bigvee_{j \in J} v_j \mid \psi_j]\!]$, and therefore the formula's triviality. But checking either unsatisfiability of $\varphi$ in $T_{\Sigma/E \cup B}$ or a parameterized subsumption may sometimes require the use of formula simplification techniques and user-provided lemmas as explained in Footnote 4. In particular, the application of this extremely useful inference rule may sometimes fail for a formula where the containment $[\![u \mid \varphi]\!] \subseteq_Y [\![\bigvee_{j \in J} v_j \mid \psi_j]\!]$ actually holds. To remedy this limitation, the SPLIT, CASE ANALYSIS and SUBSTITUTION auxiliary rules explained in Section 5.4.1 can be invoked to help achieve a successful application of SUBSUMPTION.

Before explaining the STEP$^{\forall}$ proof rule we introduce some notational conventions associated to a reachability formula $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ with parameters $Y$.

Let $R = \{l_j \to r_j \text{ if } \phi_j\}_{j \in J}$. We define:

$$\textsc{unify}(u \mid \varphi',\ R) \equiv \{(j, \alpha) \mid \alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u, l_j)\} \tag{5.22}$$

a complete set of $E_\Omega \cup B_\Omega$-unifiers[12] of a pattern $u \mid \varphi'$ with the lefthand-sides of the rules in $R$.

Consider now the rule:

**Step$^{\forall}$**

$$\frac{\bigwedge_{(j,\alpha) \in \textsc{unify}(u \mid \varphi',\ R)} [\mathcal{A} \cup \mathcal{C},\ \varnothing] \vdash_T\ (r_j \mid \varphi' \wedge \phi_j)\alpha \to^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha}{[\mathcal{A},\ \mathcal{C}] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i} \tag{5.23}$$

where the above conjunction symbol abbreviates a *set of goals*[13] that need to be proved as hypotheses, and where $\varphi' \equiv \varphi \wedge \bigwedge_{(i,\beta) \in \textsc{match}(u,\ \{v_i\}, Y)} \neg(\psi_i \beta)$. This inference rule allows us to take one step with the rules in $\mathcal{R}$. The following remarks can help clarify this inference rule's meaning:

[11] If $\mathcal{R}$'s equational theory $(\Sigma, E \cup B)$ is FVP and has an OS-compact constructor subtheory $(\Omega, E_\Omega \cup B_\Omega)$, variant satisfiability makes satisfiability of quantifier-free formulas in $T_{\Sigma/E \cup B}$ decidable [25]. In general, however, we can only assume $\vec{E}$ convergent modulo $B$, so that satisfiability of a QF formula $\varphi$ in $T_{\Sigma/E \cup B}$ becomes in general undecidable. Likewise, the, in general undecidable, checking of satisfiability/validity in $T_{\Sigma/E \cup B}$ also arises for constraints involved in the application of the AXIOM rule: such checks must be either replaced by safe but incomplete checks, or, under user control, become explicit proof obligations to be discharged by an inductive theorem prover backend.

[12] Without loss of generality, all $E_\Omega \cup B_\Omega$-unifiers will be assumed to: (i) have as their domain exactly the variables of the terms that they unify, and (ii) introduce *fresh* variables, i.e., all variables in $ran(\alpha)$ will be *new variables*, different from all other variables in the formulas that originated the need for unification. We call this the *freshness assumption* on unifiers. Furthermore, we also assume that the rules $R$ have been *renamed with fresh variables*, so that, after renaming, the rules $R$ do not share any variables with the sequent $[\mathcal{A},\ \mathcal{C}] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ appearing in the STEP$^{\forall}$ rule.

[13] Recall that all goals, to be properly so called, *must be checked for $T$-consistency.* Therefore, both the hypothesis goals and the conclusion are assumed $T$-consistent. The inference rule's application is automatically blocked, so that the proof process cannot be continued, if it generates a $T$-inconsistent goal.

1. Note that, from the definitions of $\textsc{match}(u,\ \{v_i\}, Y)$ and of parameterized subsumption in Section 5.2, it follows easily that we have a parametric inclusion:

$$[\![u \mid \varphi \wedge \bigvee_{(i,\beta)\in\textsc{match}(u,\ \{v_i\},Y)} \psi_i\beta]\!] \subseteq_Y [\![\bigvee_i v_i \mid \psi_i]\!] \tag{5.24}$$

   and that, by definition of $\varphi'$, we have a union decomposition:

$$[\![u \mid \varphi]\!] = [\![u \mid \varphi \wedge \bigvee_{(i,\beta)\in\textsc{match}(u,\ \{v_i\},Y)} \psi_i\beta]\!] \cup [\![u \mid \varphi']\!]. \tag{5.25}$$

   But since states in the left subset have already reached $[\![\bigvee_i v_i \mid \psi_i]\!]$, it is enough for us to prove the sequent $[\mathcal{A},\ \mathcal{C}] \vdash_T\ u \mid \varphi' \to^{\circledast} \bigvee_i v_i \mid \psi_i$.

2. Also, since $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ has been checked $T$-consistent, a fortiori $u \mid \varphi' \to^{\circledast} \bigvee_i v_i \mid \psi_i$ is $T$-consistent, and therefore it holds iff it does for all $T$-terminating sequences *of length 1 or more* starting in a state in $[u_0] \in [\![u \mid \varphi']\!]$. That is, $[u_0]$ has an $\mathcal{R}$-successor $[u_1]$ in such a sequence.

3. But using *constrained narrowing* (in the sense of [87]) of $u \mid \varphi'$ with the (possibly conditional) rules $R$ modulo $E_\Omega \cup B_\Omega$, we can symbolically compute the new set of preconditions $\{(r_j \mid \varphi' \wedge \phi_j)\alpha \mid (j,\alpha) \in \textsc{unify}(u \mid \varphi',\ R)\}$ obtained by one-step transitions from states in $u \mid \varphi'$ with the rules $R$. Therefore, instead of proving the sequent $[\mathcal{A},\ \mathcal{C}] \vdash_T\ u \mid \varphi' \to^{\circledast} \bigvee_i v_i \mid \psi_i$, it is enough for us to prove the conjunction of sequents in the upper part of the $\textsc{Step}^{\forall}$ rule.

Note the crucial fact that, for the new goals generated by $\textsc{Step}^{\forall}$, the formulas in $\mathcal{C}$ are added to $\mathcal{A}$, so that from now on they can be used by $\textsc{Axiom}$.

**Axiom**

$$\frac{\bigwedge_j [\mathcal{A},\ \mathcal{C}] \vdash_T\ v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \to^{\circledast} \bigvee_i v_i \mid \psi_i}{[\mathcal{A},\ \mathcal{C}] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i} \tag{5.26}$$

where $(u' \mid \varphi' \to^{\circledast} \bigvee_j v'_j \mid \psi'_j) \in \mathcal{A}$ has parameters $Y'$, and the substitution $\alpha$ has $dom(\alpha) = vars(u' \mid \varphi') = U'$ and $ran(\alpha) \subseteq vars(u \mid \varphi) = U$ and is such that $u =_{E_\Omega \cup B_\Omega} u'\alpha$ and $T_{\Sigma/E\cup B} \models \varphi \Rightarrow \varphi'\alpha$. That is, the matching substitution $\alpha$ gives us a subsumption $u \mid \varphi \sqsubseteq u' \mid \varphi'$, and therefore an inclusion $[\![u \mid \varphi]\!] \subseteq [\![(u' \mid \varphi')\alpha]\!]$. We assume that $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ and $u' \mid \varphi' \to^{\circledast} \bigvee_j v'_j \mid \psi'_j$ do not share variables, which can always be guaranteed by renaming. This inference rule is subject to the additional parameter preservation conditions that, for $Z = vars(\bigvee_i v_i \mid \psi_i)$ and $Y = U \cap Z$, (i) $Y = vars(\alpha(Y'))$, and (ii) for each $j$, $Y = vars((v'_j \mid \psi'_j)\alpha) \cap vars(\bigvee_i v_i \mid \psi_i)$. This is required for correct implicit quantification. $\textsc{Axiom}$ allows us to use a trusted formula in $\mathcal{A}$ to summarize multiple transition steps. Since $\varphi$ is stronger than $\varphi'\alpha$, for each $j$ we add $\varphi$ to $(v'_j \mid \psi'_j)\alpha$ (the result of using axiom $u' \mid \varphi' \to^{\circledast} \bigvee_j v'_j \mid \psi'_j$). To find the matching substitution $\alpha$ more easily, in automatic applications of $\textsc{Axiom}$ we require that $vars(\varphi') \subseteq vars(u')$, so that all the variables in $vars(\varphi')$ are matched. However, this syntactic requirement is not always met in practice (see Section 5.4.2 for an example). The fully general application of $\textsc{Axiom}$ can be performed by a user command that provides the required matching substitution $\alpha$ instantiating $u' \mid \varphi'$.

**Proof Trees, Closed Goals, and Provability**. Given an initial set of $T$-consistent sequents $[\mathcal{L},\ \mathcal{C}] \vdash_T\ \mathcal{C}$, with $\mathcal{L}$ valid reachability formulas in the theory $\mathcal{R}$, a $T$-consistent sequent $[\mathcal{A},\ \mathcal{C}'] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ is called a *subgoal* of this initial set of sequents iff either: (i) it is one of the sequents in the initial set, or (ii) it is one of the hypothesis sequents obtained by repeated application of the above STEP$^{\forall}$ and AXIOM rules. Therefore, $[\mathcal{A}, \mathcal{C}']$ must be either $[\mathcal{A}, \mathcal{C}'] = [\mathcal{L}, \mathcal{C}]$, or $[\mathcal{A}, \mathcal{C}'] = [\mathcal{L} \cup \mathcal{C}, \varnothing]$. We call any such subgoal *closed* if a proof tree can be built with such a subgoal as its root such that each of its leaves can be closed by a SUBSUMPTION inference step. Finally, we say that $\mathcal{R}$ *proves* $[\mathcal{L},\ \mathcal{C}] \vdash_T \mathcal{C}$ if all the goals in the initial set of goals $[\mathcal{L},\ \mathcal{C}] \vdash_T \mathcal{C}$ have been closed. Recall that this is an all-or-nothing requirement: *all* the original goals $\mathcal{C}$ must be closed for them to be (collectively) proved. Note, finally, that a $T$-dubious goal can be used for building up a proof tree only if the additional proof obligations required to show that it is $T$-consistent have themselves been closed, i.e., if it has been actually shown to be a $T$-consistent goal. In summary, therefore, all subgoals of any closed goal and the closed goal itself are always, by definition, $T$-consistent.

The soundness of the SUBSUMPTION, STEP$^{\forall}$, and AXIOM inference rules is now the theorem:

**Theorem 5.4 (Soundness)** *Let $\mathcal{R}$ be a rewrite theory, and $\mathcal{C}$ a finite set of $T$-consistent reachability formulas. If $\mathcal{R}$ proves $[\mathcal{L},\ \mathcal{C}] \vdash_T \mathcal{C}$ and $\mathcal{R} \models_T^{\forall} \mathcal{L}$, then $\mathcal{R} \models_T^{\forall} \mathcal{C}$.*

### 5.4.1 The SPLIT, CASE ANALYSIS and SUBSTITUTION Auxiliary Rules

**Auxiliary Rules as Deduction Modulo**. All auxiliary rules presented in this section are rules of the form:

$$\frac{\mathcal{G}}{\mathcal{G}'} \tag{5.27}$$

where $\mathcal{G}$ and $\mathcal{G}'$ are *semantically equivalent* sets of goals, in the sense that $\mathcal{R} \models_T^{\forall} \mathcal{G} \;\Leftrightarrow\; \mathcal{R} \models_T^{\forall} \mathcal{G}'$. Since all auxiliary rules transform some goals into semantically equivalent ones, they do not affect the soundness of the inference system. Their specific role is to *facilitate the application* of the three inference rules of reachability logic, particularly of the SUBSUMPTION and AXIOM rules. They can be best understood as endowing reachability logic with *deduction modulo* [94] capabilities. That is, we can view the semantic equivalence $\mathcal{R} \models_T^{\forall} \mathcal{G} \;\Leftrightarrow\; \mathcal{R} \models_T^{\forall} \mathcal{G}'$ as an equivalence relation $\mathcal{G} \equiv \mathcal{G}'$, so that we can apply the three reachability logic rules *modulo* such goal equivalences. The classical analogue in first-order theorem proving is the application of inference rules, for example in a sequent calculus, *modulo* Boolean equivalences (see, e.g., [94, 95, 96]). The key point of deduction modulo is that *the original inference rules are not changed*, but deduction is rendered much more effective by allowing them to be applied *modulo* the given semantic equivalences.

A key reason why the auxiliary rules presented below are particularly useful is that the symbolic methods used in the application of the SUBSUMPTION, STEP$^{\forall}$, and AXIOM rules provide only *sufficient conditions* for verifying certain semantic requirements. For example, in the application of the SUBSUMPTION rule, the parameterized subsumption check $u \mid \varphi \sqsubseteq_Y \bigvee_{j \in J} v_j \mid \psi_j$ is a sufficient condition for proving the parametric semantic inclusion $[\![u \mid \varphi]\!] \subseteq_Y [\![\bigvee_{j \in J} v_j \mid \psi_j]\!]$. The point is that such a check may fail for a goal $\mathcal{G}'$ as given, but may succeed for a semantically equivalent goal (or set of goals) $\mathcal{G}$ thanks to an auxiliary rule.

The following SPLIT rule is an auxiliary proof rule that uses a $\Sigma$-formula equivalence $\varphi \Leftrightarrow \psi \vee \phi$ to split a goal into two. SPLIT is a *validity-preserving* rule transforming a set $\mathcal{G}$ of reachability logic goals to be proved (understood as a *conjunction*) into a *semantically equivalent* set of goals $\mathcal{G}'$, so that $\mathcal{R} \models_T^{\forall} \mathcal{G} \;\Leftrightarrow\; \mathcal{R} \models_T^{\forall} \mathcal{G}'$. This means that SPLIT *does not affect soundness.*

**Split**

$$\frac{[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \psi \to^{\circledast} A \qquad [\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \phi \to^{\circledast} A}{[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} A} \tag{5.28}$$

subject to the conditions: (i) $T_{\Sigma/E \cup B} \models \varphi \Leftrightarrow \psi \vee \phi$, and (ii) (parameter preservation) $vars(u \mid \varphi) \cap vars(A) = vars(u \mid \psi) \cap vars(A) = vars(u \mid \phi) \cap vars(A)$.

**Lemma 5.6** *In the above* Split *rule,* $\mathcal{R} \models^{\forall}_{T} \mathcal{G} \;\Leftrightarrow\; \mathcal{R} \models^{\forall}_{T} \mathcal{G}'$*, where* $\mathcal{G}$ *is the premise and* $\mathcal{G}'$ *the conclusion.*

A very common use of the Split rule in our examples is to use an always valid equivalence $\varphi \Leftrightarrow ((\varphi \wedge \phi) \vee (\varphi \wedge \neg\phi))$ to split the precondition $u \mid \varphi$ depending on whether an additional condition $\phi$ holds or not. This still leaves open the question of when it would be advantageous to use the Split rule and with what choice of $\phi$. One attractive possibility is to use Split to increase success in application attempts for the Axiom rule. Suppose that we have tried to apply Axiom with a substitution $\alpha$ such that $u =_{E_\Omega \cup B_\Omega} u'\alpha$, but the condition $T_{\Sigma/E \cup B} \models \varphi \Rightarrow (\varphi'\alpha)$ does not hold. Suppose, however, that $\varphi \wedge (\varphi'\alpha)$ is satisfiable in $T_{\Sigma/E \cup B}$, and that $vars(u \mid \varphi) \cap vars(\bigvee_i v_i \mid \psi_i) = vars(u \mid \varphi \wedge (\varphi'\alpha)) \cap vars(\bigvee_i v_i \mid \psi_i)$. In such a case, we can first apply Split to split $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ into $u \mid \varphi \wedge (\varphi'\alpha) \to^{\circledast} \bigvee_i v_i \mid \psi_i$ and $u \mid \varphi \wedge \neg(\varphi'\alpha) \to^{\circledast} \bigvee_i v_i \mid \psi_i$, and then apply Axiom (checking parameter preservation) to close the first of these two reachability goals.

Another very common use of the Split rule is to use a semantic QF formula equivalence $T_{\Sigma/E \cup B} \models \varphi \Leftrightarrow \psi$ to replace a goal $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \phi_i$ by the equivalent goal $u \mid \psi \to^{\circledast} \bigvee_i v_i \mid \phi_i$. This corresponds to the special case of splitting on the equivalence $T_{\Sigma/E \cup B} \models \varphi \Leftrightarrow (\psi \;\vee\; \bot)$, so that the second goal becomes vacuous and is automatically discharged. In this special case the parameter preservation condition can be relaxed to just requiring $vars(u \mid \varphi) \cap vars(A) = vars(u \mid \psi) \cap vars(A)$. In general we may not have $vars(u \mid \varphi) \cap vars(A) = vars(u \mid \bot) \cap vars(A)$, but this is immaterial: let $\{x_1, \ldots x_n\} = vars(u \mid \varphi) \cap vars(A) \backslash vars(u \mid \bot) \cap vars(A)$; if we care to do so, we can replace $\bot$ by the semantically equivalent formula $\bigwedge_{1 \leqslant i \leqslant n} x_i \neq x_i$.

A second, also validity-preserving, auxiliary rule is a Case Analysis rule. It allows us to reason by cases by decomposing a variable $x{:}s$ of sort $s$ into a complete covering of it by constructor patterns. Call $\{u_1, \ldots, u_k\} \subseteq T_\Omega(X)_s$ a *pattern set* for sort $s$ iff $T_{\Omega,s} = \bigcup_{1 \leqslant i \leqslant k} \{u_i\rho \mid \rho \in [X \to T_\Omega]\}$. We assume throughout that $i \neq i' \Rightarrow vars(u_i) \cap vars(u_{i'}) = \varnothing$, and that all variables in the pattern set are *fresh* variables not appearing in any current goal.

**Case Analysis**

$$\frac{\bigwedge_{1 \leqslant i \leqslant k} [\mathcal{A},\ \mathcal{C}] \vdash_T (u \mid \varphi)\{x{:}s \mapsto u_i\} \to^{\circledast} A\{x{:}s \mapsto u_i\}}{[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} A} \tag{5.29}$$

where $x{:}s \in vars(u)$ and $\{u_1, \ldots, u_k\}$ is a pattern set for $s$.

**Lemma 5.7** *In the above* Case Analysis *rule,* $\mathcal{R} \models^{\forall}_{T} \mathcal{G} \;\Leftrightarrow\; \mathcal{R} \models^{\forall}_{T} \mathcal{G}'$*, where* $\mathcal{G}$ *is the premise and* $\mathcal{G}'$ *the conclusion.*

A third auxiliary rule is the Substitution rule, which makes it possible to solve a conjunction of equalities $\bigwedge_i w_i = w'_i$ in a reachability formula's precondition $u \mid \bigwedge_i w_i = w'_i \wedge \varphi$ and apply the substitutions solving the conjunction to the formula's midcondition, provided a finitary unification algorithm can be used to solve them. This will be the case if a subtheory $(\Sigma_1, E_1 \cup B_1) \subseteq (\Sigma, E \cup B)$ can be found having a finitary $E_1 \cup B_1$-unification algorithm and such that $T_{\Sigma/E \cup B}|_{\Sigma_1} \cong T_{\Sigma_1/E_1 \cup B_1}$ and $\bigwedge_i w_i = w'_i$ is a conjunction of $\Sigma_1$-equations. Substitution is the conditional inference rule:

**Substitution**

$$\frac{\bigwedge_{\alpha \in Unif_{E_1 \cup B_1}(\bigwedge_i w_i = w'_i)} [\mathcal{A},\ \mathcal{C}] \vdash_T u\alpha \mid \varphi\alpha \wedge \widehat{\alpha} \rightarrow^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\alpha}{[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \bigwedge_i w_i = w'_i \wedge \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j} \tag{5.30}$$

subject to the above-mentioned conditions on $(\Sigma_1, E_1 \cup B_1)$ and $\bigwedge_i w_i = w'_i$, and where if $dom(\alpha) = \{x_1, \ldots, x_k\}$ then $\widehat{\alpha} \equiv x_1 = \alpha(x_1) \wedge \ldots \wedge x_k = \alpha(x_k)$, with the same freshness assumptions as in Footnote 12 for the unifiers $\alpha \in Unif_{E_1 \cup B_1}(\bigwedge_i w_i = w'_i)$.

**Lemma 5.8** *In the above* Substitution *rule,* $\mathcal{R} \models^{\forall}_{T} \mathcal{G} \;\Leftrightarrow\; \mathcal{R} \models^{\forall}_{T} \mathcal{G}'$, *where* $\mathcal{G}$ *is the premise and* $\mathcal{G}'$ *the conclusion.*

The proof of Lemma 5.8, given in Appendix D, uses the following lemma, which is of general interest and is also proved in Appendix D, as an auxiliary lemma:

**Lemma 5.9** *(Instance Lemma). Suppose* $\mathcal{R} \models^{\forall}_{T} u \mid \psi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ *with parameters* $Y$*, and let* $\beta$ *be a substitution whose domain* $V$ *is contained in* $vars(u \mid \psi)$ *and where the variables in* $ran(\beta)$ *are all fresh. Then* $\mathcal{R} \models^{\forall}_{T} (u \mid \psi)\beta \rightarrow^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\beta$

**Goal Subsumption Simplification**. The above Instance Lemma justifies the following, *validity-preserving, goal subsumption* simplification: whenever in an unclosed proof tree two subgoals of the form:

$$[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j \quad [\mathcal{A},\ \mathcal{C}] \vdash_T u\beta \mid \psi \rightarrow^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\beta \tag{5.31}$$

appear in two different *leaves* of the partial proof tree, with, say, $Y$ the parameters of the more general subgoal, $\beta$ satisfying the conditions in the Instance Lemma 5.9, $vars(u\beta \mid \psi) \cap vars((\bigvee_{j\in J} v_j \mid \phi_j)\beta) = vars((u \mid \varphi)\beta) \cap vars((\bigvee_{j\in J} v_j \mid \phi_j)\beta) = vars(\beta(Y))$, and $T_{\Sigma/E\cup B} \models \psi \Rightarrow (\varphi\beta)$. Then, in order to close the entire proof tree, only the more general goal $[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ as well as any other leaf nodes, but excluding the instance leaf subgoal $[\mathcal{A},\ \mathcal{C}] \vdash_T u\beta \mid \psi \rightarrow^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\beta$, need to be closed.

The correctness of this goal subsumption simplification then follows from the Instance Lemma plus the fact that the above requirements ensure a parameterized inclusion $[\![u\beta \mid \psi]\!] \subseteq_{vars(\beta(Y))} [\![(u \mid \varphi)\beta]\!]$, and therefore the implication $(\mathcal{R} \models^{\forall}_{T} (u \mid \varphi)\beta \rightarrow^{\circledast} \bigvee_{j\in J}(v_j \mid \phi_j)\beta) \Rightarrow (\mathcal{R} \models^{\forall}_{T} u\beta \mid \psi \rightarrow^{\circledast} \bigvee_{j\in J}(v_j \mid \phi_j)\beta)$.

### 5.4.2 A Simple Example

The following very simple example of a counter system illustrates the use of *goal subsumption* and of the Split, Case Analysis and Substitution rules. It also illustrates some possible pitfalls when applying the Axiom rule.

Recall from the Introduction the counter system whose states are of the form $\langle n \rangle$, with $n$ a natural number. We can specify this counter system as a rewrite theory $\mathcal{R}$ with three sorts, *Nat*, *Bool*, and *Counter*, a constructor signature $\Omega$ with constants $0, 1$ and binary operator $\_+\_$ of sort *Nat*, constants $\top, \bot$ of sort *Bool*, and a cell constructor $\langle\_\rangle : Nat \rightarrow Counter$. The signature $\Sigma$ extends $\Omega$ with defined functions $>, \geqslant$: *Nat Nat* $\rightarrow$ *Bool*. The axioms $B = B_\Omega$ are the associativity-commutativity of addition $\_+\_$ and the identity axiom $n + 0 = n$. The equations $E$ define the predicates $>$ and $\geqslant$ as follows: $n + m + 1 > n = \top$, $n > n + m = \bot$, $n + m \geqslant n = \top$, $n \geqslant n + m + 1 = \bot$. This equational theory is FVP. Furthermore, since its constructor subtheory $(\Omega, B_\Omega)$ is decidable by variant satisfiability, satisfiability of QF $\Sigma$-formulas in

$T_{\Sigma/E\cup B_\Omega}$ is also decidable [25]. The rewrite rules $R$ defining the semantics of this simple counter system are: $\langle n+1\rangle \to \langle n\rangle$ and $\langle n+1\rangle \to \langle n+1+1\rangle$. That is, a non-zero counter can be incremented or decremented by one unit. Its set of terminating states, of sort *Counter*, can be characterized by the pattern formula $T = \langle 0\rangle \mid \top$.

Note that this system is non-terminating. However, it satisfies the partial correctness reachability formula $\langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top$, which is actually in the Hoare logic fragment and can be proved as follows. We start with the sequent

$$[\varnothing,\ \{\langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top\}] \vdash_T \langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top \tag{5.32}$$

Note that $\langle 0\rangle$ cannot match $\langle n\rangle$, so we are unable to strengthen the constraint on our precondition by overapproximated difference performed as part of the Step$^\forall$ rule. Since precondition term $\langle n\rangle$ is the most general possible, the most general unifiers with our two rewrite rules is just the mapping $n \mapsto n+1$. Using these unifiers, by the Step$^\forall$ rule we get the sequents:

$$[\{\langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top\},\ \varnothing] \vdash_T \langle n'\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top \tag{5.33}$$

$$[\{\langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top\},\ \varnothing] \vdash_T \langle n''+1+1\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top \tag{5.34}$$

But, by *goal subsumption*, only the first, more general goal needs to be proved. Applying Axiom to the more general subgoal we get the subgoal

$$[\{\langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top\},\ \varnothing] \vdash_T \langle 0\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top \tag{5.35}$$

which can be immediately closed by the Subsumption rule, thus proving the partial correctness property $\mathcal{R} \models^{\forall}_{T} \langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top$.

Another general property of this counter system is that from a positive counter $\langle n+1\rangle$ any other counter holding a smaller number will be eventually reached along any terminating sequence. This can be specified by means of the reachability formula $\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top$, parametric on $m$, which can be proved as follows. We start with the sequent

$$[\varnothing,\ \{\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top\}] \vdash_T \langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.36}$$

Applying the Step$^\forall$ rule we get the sequents:

$$[\{\langle n{+}1\rangle \mid n{+}1 > m \to^{\circledast} \langle m\rangle \mid \top\}, \varnothing] \vdash_T \langle n'\rangle \mid n'+1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.37}$$

$$[\{\langle n{+}1\rangle \mid n{+}1 > m \to^{\circledast} \langle m\rangle \mid \top\}, \varnothing] \vdash_T \langle n''{+}1{+}1\rangle \mid n''{+}1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.38}$$

Note that for $\beta = \{n' \mapsto n''+1+1\}$ we get $T_{\Sigma/E\cup B} \models n''+1 > m \Rightarrow ((n'+1 > m)\beta)$, which can be automatically proved in Maude using variant satisfiability. Therefore, by *goal subsumption*, only the first goal needs to be closed. But since $\{0, k+1\}$ is a pattern set for the sort *Nat*, we can use the Case Analysis auxiliary rule to decompose the first goal into the subgoals:

$$[\{\langle n{+}1\rangle \mid n{+}1 > m \to^{\circledast} \langle m\rangle \mid \top\}, \varnothing] \vdash_T \langle 0\rangle \mid 0+1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.39}$$

$$[\{\langle n{+}1\rangle \mid n{+}1 > m \to^{\circledast} \langle m\rangle \mid \top\}, \varnothing] \vdash_T \langle k{+}1\rangle \mid k{+}1{+}1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.40}$$

Applying Split to the first subgoal with equivalence $T_{\Sigma/E\cup B} \models 0+1 > m \Leftrightarrow m = 0$, which can be automatically proved by variant satisfiability, we obtain:

$$[\{\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top\},\ \varnothing] \vdash_T \langle 0\rangle \mid m = 0 \to^{\circledast} \langle m\rangle \mid \top \tag{5.41}$$

Then Substitution lets us solve the constraint $m = 0$ with $\{m \mapsto 0\}$, giving:

$$[\{\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top\},\ \varnothing] \vdash_T \langle 0\rangle \mid m = 0 \to^{\circledast} \langle 0\rangle \mid \top \tag{5.42}$$

which is trivially subsumed by Subsumption. This closes the first subgoal.

Using the equivalence $T_{\Sigma/E\cup B} \models k+1+1 > m \Leftrightarrow (k+1 > m \vee m = k+1)$, which can also be automatically proved by variant satisfiability, we can use the Split auxiliary rule to split the second subgoal into the two subgoals:

$$[\{\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top\},\ \varnothing] \vdash_T \langle k+1\rangle \mid m = k+1 \to^{\circledast} \langle m\rangle \mid \top \tag{5.43}$$

$$[\{\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top\},\ \varnothing] \vdash_T \langle k+1\rangle \mid k+1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.44}$$

Then, the first of these subgoals can be closed by applying Substitution followed by Subsumption; and the second subgoal can be closed by applying Axiom followed by Subsumption. This finishes the proof of the desired property $\mathcal{R} \models^{\forall}_{T} \langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top$.

**Getting Nowhere with the Axiom Rule**. The Axiom rule is very powerful. But its power must be used wisely. Unwise applications of Axiom can produce *invalid* subgoals which can never be closed, so we get nowhere that way. The reason is easy to explain. Axiom is a very powerful "seven league boots" inference rule that, under appropriate parameter preservation conditions, can apply an axiom $A \to^{\circledast} B$ such that $[\![C]\!] \subseteq [\![A\alpha]\!]$ to a goal $C \to^{\circledast} D$ with $C \equiv u \mid \varphi$ to "fast forward" and reduce the proof of $C \to^{\circledast} D$ to that of[14] $(B\alpha) \wedge \varphi \to^{\circledast} D$. But, of course, the Axiom rule *implicitly assumes in its hypothesis* that the midcondition $(B\alpha) \wedge \varphi$ will happen *before* (or simultaneously with) the midcondition $D$. But this need not be the case in general and can, if Axiom is applied unwisely, produce invalid subgoals that can never be closed.

We can illustrate this undesirable phenomenon by an unwise application of Axiom. Suppose that we get into our heads the idea that, to prove the property $\mathcal{R} \models^{\forall}_{T} \langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top$, we will be better off using the already proved partial correctness property $\mathcal{R} \models^{\forall}_{T} \langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top$ as an axiom. That is, we start with the sequent:

$$[\{\langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top\}, \{\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top\}] \vdash_T \langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.45}$$

Then, one application of Axiom yields the sequent:

$$[\{\langle n\rangle \mid \top \to^{\circledast} \langle 0\rangle \mid \top\}, \{\langle n+1\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top\}] \vdash_T \langle 0\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top \tag{5.46}$$

But $\mathcal{R} \not\models^{\forall}_{T} \langle 0\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top$, since this formula is parameterized by $m$, and, therefore, if it were valid, we should in particular have for $\rho = \{m \mapsto 1\}$ that $\mathcal{R} \models^{\forall}_{T} \langle 0\rangle \mid n+1 > 1 \to^{\circledast} \langle 1\rangle \mid \top$. But of course it is impossible to rewrite the counter state $\langle 0\rangle$ to the counter state $\langle 1\rangle$. That is, this application of Axiom gets us nowhere. Note, furthermore, that in this example $[\![T]\!] = \{\langle 0\rangle\} = [\![\langle 0\rangle \mid n+1 > 1]\!]$. But of course $\langle 0\rangle \notin [\![\langle 1\rangle \mid \top]\!] = \{\langle 1\rangle\}$. Therefore, the derived goal $\langle 0\rangle \mid n+1 > m \to^{\circledast} \langle m\rangle \mid \top$ is $T$-*inconsistent.*

[14] We are abusing notation a little for the sake of conciseness. $(B\alpha)$ is really a disjunction of pattern predicates, and what the notation $(B\alpha) \wedge \varphi$ abbreviates is the conjunction of $\varphi$ with each formula in each of those pattern predicates.

Two simple guidelines for the application of Axiom can be drawn from this last frustrated proof attempt and the prior successful applications of Axiom:

1. Given a goal $A \rightarrow^{\circledast} B \in \mathcal{C}$, with $B \equiv v \mid \psi$, call a goal $C \rightarrow^{\circledast} D$ a *descendant* of $A \rightarrow^{\circledast} B$ if $C \rightarrow^{\circledast} D$ has been obtained from $A \rightarrow^{\circledast} B$ by successive applications of the Step$^{\forall}$ rule. Given $C \equiv u \mid \phi$, the fact that Axiom can be applied to the descendant $C \rightarrow^{\circledast} D$ using the "ancestor" axiom $A \rightarrow^{\circledast} B$ because $[\![C]\!] \subseteq [\![A\alpha]\!]$ will often be linked to the fact that in the resulting goal $(B\alpha) \wedge \varphi \rightarrow^{\circledast} D$, midcondition $D$ is just a substitution instance of $B$ by the same chain of substitutions that were used to obtain $C$. Thus, $D$ is likely to subsume the precondition $(B\alpha) \wedge \varphi$, resulting in a successful application of Axiom followed by Subsumption. This reasoning can be generalized to ancestors of the form $A \rightarrow^{\circledast} B$ with $B \equiv \bigvee_j v_j \mid \psi_j$.

2. It is always a *bad idea* to apply a formula $A \rightarrow^{\circledast} B$ as an axiom to another formula $C \rightarrow^{\circledast} D$ when $B$ is a *postcondition* but $D$ is not so: this is what got us into trouble in the above frustrated proof attempt. Axioms with postconditions should only be applied to formulas having also a postcondition.

### 5.4.3 Revisiting QLOCK

In the same vein as for the counter example, here we revisit QLOCK, bringing our entire example together to obtain a bird's eye view of the mathematical proof. We will not explicitly list all intermediate states, since the larger number of rules as well as list/multiset unification generate many tens of descendants. Instead, we will describe such states more abstractly by explaining which rules can narrow which goals, which vacuous goals are closed by Subsumption, and how the axioms are applied. Recall from the note in Section 5.4 that we needed to prove in the rewrite theory $\mathcal{R}$ of QLOCK that the following sequents hold: (a) $[\varnothing,\ \mathcal{C}] \vdash_T\ P_1 \rightarrow^{\circledast} \bigvee_{j \in I} P'_j$ and (b) $[\varnothing,\ \mathcal{C}] \vdash_T\ P_2 \rightarrow^{\circledast} \bigvee_{j \in I} P'_j$, where $\mathcal{C} = \{P_i \rightarrow^{\circledast} \bigvee_{j \in I} P'_j\}_{i \in I}$ and $I = \{1, 2\}$. As a convenience to the reader, we expand out the two formulas below:

$$< n \mid w \mid \varnothing \mid q > \mid \psi \rightarrow^{\circledast} [< n' \mid w' \mid i' \mid i';\ q' > \mid \varphi' \vee < n' \mid w' \mid \varnothing \mid q' > \mid \psi'] \qquad (5.19)$$

$$< n \mid w \mid i \mid i\ ;\ q > \mid \varphi \rightarrow^{\circledast} [< n' \mid w' \mid i' \mid i';\ q' > \mid \varphi' \vee < n' \mid w' \mid \varnothing \mid q' > \mid \psi'] \qquad (5.20)$$

where $\varphi = dupl(n\ w\ i) \neq tt$, $\psi = dupl(n\ w) \neq tt$, and $\varphi', \psi'$ are their obvious renamings. By abuse of notation, let $P_1$ and $P_2$ refer to both the goal preconditions as well as the entire formula/sequents to be proved.

To begin the proof, we first apply the Step$^{\forall}$ rule. The goal $P_1$ has 12 successors while $P_2$ has 14. The discrepancy lies in the fact that, since goal $P_2$ has a non-empty set of processes in its critical section, the rule *c2n* is enabled. For all other rules, the successors generated for $P_1$ and $P_2$ are entirely analogous. Note that most rules have multiple successors; this occurs due to the flexibility of *ACU* and *AU* unification, so that any rule that contains a multiset variable underneath a multiset union operator/list variable underneath a list concatenation operators generates two variants: one for the non-empty and empty multiset/list respectively.

After generating the successors of goals $P_1$ and $P_2$, one of two things will happen. A successor of goal $P_2$ generated by the *w2c* rule will immediately be closed by Subsumption because it is vacuous—adding an extra critical process to the critical process set violates the *dupl* predicate constraint. For all other successors, they are now an instance of one of our two original invariant patterns, allowing us to apply the Axiom rule. Since the original goals have no parameters, the structure of invariants of the form $P \rightarrow^{\circledast} [P\sigma]$

and the Axiom rule force all the successors to be immediately ready to be subsumed by the Subsumption rule. Recall that each axiom will generate two successors—one corresponding to the $P_1$ case and another to the $P_2$ case—doubling the amount of proof goals that are ultimately closed by Subsumption.

To give a flavor for how the proof process proceeds, we consider the successors of the $P_2$ goal by the *c2n* rule. For convenience, we recall the *c2n* rule below:

$$\mathit{c2n} : < n \mid w \mid c\ i \mid i\ ;\ q > \;\to\; < n\ i \mid w \mid c \mid q > \tag{5.47}$$

The precondition of goal $P_2$ is $< n \mid w \mid i \mid i\ ;\ q > \mid \mathit{dupl}(n\ w\ i) \neq \mathit{tt}$. By the Step$^\forall$ rule, unify the precondition term of $P_2$ and left-hand side of *c2n* via most general unifier $\alpha = \{c \mapsto \varnothing, q \mapsto q'\}$. By rewriting the precondition of $P_2\alpha$ by *c2n*, obtain $K = < n\ i \mid w \mid \varnothing \mid q' > \mid \mathit{dupl}(n\ w\ i) \neq \mathit{tt}$ (in our implementation, this unification proceeds slightly differently due to A/U unification; see Sec. 5.5.2).

At this point, our sequent will be of the form $[\mathcal{C},\ \varnothing] \vdash_T K \to^{\circledast} \bigvee_{j \in I} P'_j$, and we can apply the Axiom rule with $P_1 \in \mathcal{C}$. To see this, note that the precondition of $P_1$, $< n \mid w \mid \varnothing \mid q > \mid \mathit{dupl}(n\ w) \neq \mathit{tt}$ covers our goal by the substitution $n \mapsto n\ i, q \mapsto q'$, where we have to prove the validity of the implication $\mathit{dupl}(n\ w\ i) \neq \mathit{tt} \Rightarrow \mathit{dupl}(n\ w\ i) \neq \mathit{tt}$, which is a tautology.

## 5.5 PROTOTYPE IMPLEMENTATION AND EXPERIMENTS

We have implemented the reachability logic proof system in Maude [6]. We exploit the fact that rewriting logic is reflective, so that concepts such as terms, rewrite rules, signatures, and theories are directly expressible as data in the logic. This is supported by Maude's `META-LEVEL` library [6]. Our prototype tool takes as input (i) a reflected rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ and (ii) a set of reachability formulas $\mathcal{C} = \{A_i \to^{\circledast} B_i\}_{i \in I}$ to be simultaneously proved.

The state of a reachability proof is represented as a set of proof sequents with associative-commutative union, as defined in Section 5.4, plus some global state information (for example, the theory $\mathcal{R}$). Given goal set $\mathcal{C}$, the initial proof state will be $\left\{[\varnothing,\ \mathcal{C}] \vdash_T A_i \to^{\circledast} B_i\right\}_{i \in I}$, that is, one sequent for each goal in $\mathcal{C}$. Given the simplicity of the proof system, we need only perform a very simple proof search strategy: until there are no pending goals, we first apply Axiom as much as possible and then apply Step$^\forall$ if possible. Before every Step$^\forall$ and Axiom application, we greedily try to the apply the Substitution rule to simplify goals and the Subsumption rule to discharge them. At the same time, we also perform T-consistency checks so that errors can be reported to the user as early as possible. Currently, aside from Substitution, the other derived rules must be applied manually—in a future version of the tool we will investigate heuristics to further guide auxiliary proof rule application. Note that the simple strategy just outlined cannot distinguish between appropriate and inappropriate uses of the Axiom rule; instead, for any sequent, we allow the user to control which reachability formulas will be tried as axioms by selecting some subset $\mathcal{A}' \subset \mathcal{A}$ of possible axioms.

We of course need to mechanize the three proof rules, all the auxiliary rules, and the $T$-consistency checks. Internally, the *action* of each proof rule is specified as an equationally-defined function, while the *policy* is specified by a single non-deterministic rewrite rule, which arbitrarily selects an active goal to advance according to the strategy specified above. Proof rule application on a goal is controlled by a simple strategy language. Currently, there are only limited commands to interact with the strategy language—in particular, to select the set of axioms that applies to a particular goal. In a future version, there will be additional commands to modify the proof strategies for any particular goal.

Our implementation further requires a finitary $B_\Omega$-unification algorithm as well as an inductive validity backend that tries to answer inductive validity questions of the form $T_{\Sigma/E\cup B} \models \varphi$ for $\varphi$ a QF $\Sigma$-formula. Maude can perform unification modulo commutativity and associativity/commutativity with or without identity and in many cases associativity without commutativity. Our tool has infrastructure to support various user-selectable pluggable backends to try to check inductive validity. The application of the backends is syntax-driven in the sense that they are associated to some subtheory $(\Sigma_1, E_1 \cup B_1)$ of $(\Sigma, E \cup B)$ and are applied whenever the formula to be verified falls into the subsignature $\Sigma_1$. Any requirements on subtheory $(\Sigma_1, E_1 \cup B_1)$ for utilizing these backends are proof obligations for the user. Currently, we have implemented two such backends.

**Decidable Case**. $((\Sigma_1, E_1 \cup B_1) = (\Sigma, E \cup B))$. If the validity of QF $\Sigma$-formulas in $T_{\Sigma/E\cup B}$ is decidable by variant satisfiability, we use a variant satisfiability-based backend using techniques in [25, 88]. This allows us to handle any *suitable* rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ such that the equational theory $(\Sigma, E \cup B)$ has a convergent decomposition satisfying the finite variant property [23] and protects a constructor subtheory which we assume consists only of axioms $B_\Omega$ of the above-described form. Note that this means that both validity and satisfiability of QF formulas in the initial $(\Sigma, E \cup B)$-algebra $T_{\Sigma/E\cup B}$ are decidable [36]. The only exception is the case when $B_\Omega$ includes associativity without commutativity axioms for some operators unless (as is the case for the QLOCK example) such associative-only operators do not appear in constraints.

**Undecidable Case**. When $(\Sigma, E \cup B)$ does not have the finite variant property, but still protects a constructor subtheory consisting only of axioms $B_\Omega$ of the above-described form, $(\Sigma, E \cup B)$ will still protect an FVP subspecification $(\Sigma_1, E_1 \cup B_1)$ with decidable inductive validity (in the worse case, just $(\Omega, B_\Omega)$ itself). In this case, we provide a second backend that extends the variant-satisfiability one and therefore becomes a decision procedure for the inductive validity of QF $\Sigma_1$-formulas. Outside the $\Sigma_1$-formula case, it applies various heuristics based on clause simplification to try to answer inductive validity questions about QF $\Sigma$-formulas. Future versions of the tool will add other decision procedures and more powerful automated inductive theorem proving routines to further automate this kind of inductive reasoning.

In addition to the issue of proof representation, several other issues must be addressed. First, to ensure correct applications of unification, we uniquely rename all variables in rules in the theory $\mathcal{R}$ and in goals $\mathcal{C}$. Second, recall that we assume that the rewrite theory $\mathcal{R}$ has been $\Omega$-abstracted as $\hat{\mathcal{R}}$. Therefore, we have automated the $\Omega$-abstraction as well. Third, an important practical consideration during any tool development is a user interface that is flexible and usable enough to express real theories and problems that users may wish to reason about. To that end, we have developed a `FULL-MAUDE`-based user interface [97] in Maude that provides commands to input goals and invariants, solve pattern predicate subsumption/intersection queries, and specify theories plus the corresponding terminating state pattern predicates of interest. The full command grammar is given in Appendix A.

In the following subsections we illustrate how to use the tool by way of complete examples that are *executable* using our prototype Maude implementation. Recall that our implementation requires two main arguments: (i) a reflected rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ and (ii) a set of reachability formulas $\mathcal{C} = \{A_i \to^{\circledast} B_i\}_{i \in I}$ to be simultaneously proved. Generally, the user of the tool will spend some time thinking about the best way to specify a system design as a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ in Maude, since a well-specified problem can be both more readable and easier to verify. To that end, we show complete examples written in a style that we believe is both easy to read and to verify.

### 5.5.1 Counter Proof Example

Though mathematically and syntactically simple, the counter example shown in subsection 5.4.2 illustrates how each of the proof rules can be used. Below we present a Maude specification of a rewrite theory $\mathcal{R}$ specifying such a counter. Later we will see how to verify reachability formulas over this theory.

```
1  fmod PRES-NAT is
2    sort Bool .
3    op true : -> Bool [ctor] .
4    op false : -> Bool [ctor] .
5
6    sort NzNat Nat .
7    subsort NzNat < Nat .
8    op 0 : -> Nat [ctor] .
9    op 1 : -> NzNat [ctor] .
10   op _+_ : Nat Nat -> Nat [ctor assoc comm id: 0] .
11   op _+_ : NzNat Nat -> NzNat [ctor assoc comm id: 0] .
12
13   var J K : Nat . var P : NzNat .
14
15   op _<=_ : Nat Nat -> Bool .
16   op _<_  : Nat Nat -> Bool .
17
18   eq J     <= J + K = true  [variant] .
19   eq J + P <= J     = false [variant] .
20   eq J     <  J + P = true  [variant] .
21   eq J + K <  J     = false [variant] .
22 endfm
23
24 mod COUNTER is
25   protecting PRES-NAT .
26
27   var N : Nat .
28
29   sort Counter .
30   op {_} : Nat -> Counter [ctor] .
31
32   rl {N + 1} => {N + 1 + 1} .
33   rl {N + 1} => {N} .
34 endm
```

Figure 5.3: Theory specification for Counter.

Note the two rewrite rules on lines 32-33 define the state changes of the counter. Recall that this system is non-terminating, since the counter increment rule on line 32 can loop. For more details on Maude syntax, see the primer in Section 2.3.

Now that we have presented the rewrite theory to be analyzed as a Maude specification, we can use our tool to perform reachability logic verification over it. To do so, we first load the file that contains our theory specification, then load the file that contains our prototype implementation definition, and then type out the commands that make up the proof script. Since the loading commands are always relative to the filesystem layout, and are commands given directly to the Maude interpreter, not to our prototype tool, we do not display them below.

Partial Correctness Property of Counter.

As a first example, consider the proof script for proving the partial correctness property of the counter theory shown in Figure 5.4 with the associated output shown inline, as if the commands were typed directly into the terminal.

All proof scripts for our implementation are divided into two parts: a header, which initializes the proof state, followed by a body beginning with the command `start-proof` (here on line 14) that contains the actual commands that drive the prover engine to do something. Furthermore, there are three grammatical requirements that every command in our grammar obeys: (1) it is written inside parentheses; (2) it is terminated with a period (`.`) before the closing parenthesis; (3) any object theory term is written wrapped by another pair of parentheses; however, identifiers such as module, rule, and goal names need not be wrapped. All lines that are *not* wrapped in parentheses are tool output.

```
 1  (select COUNTER .)
 2  Set module to COUNTER
 3  (use tool varsat for validity on PRES-NAT .)
 4  Loaded function varsat for validity
 5  (def-term-set ({0}) | true .)
 6  Added terminating state:
 7  {0} | true
 8  (declare-vars (N:Nat) .)
 9  Declared variable(s):
10  { N:Nat }
11  (add-goal partial-correctness : ({N}) | true => ({0}) | true .)
12  Added goal(s):
13  [partial-correctness : {N:Nat} | true  => {0} | true]
14  (start-proof .)
15  Started proof:
16  [1     | {N:Nat} | true  => {0} | true]
17  (auto .)
18  Auto Results:
19  [13    | {&2:Nat} | true  => {0} | true]
20  [14    | {1 + 1 + &2:Nat} | true  => {0} | true]
21  (subsume 14 by 13 .)
22  Goal 14 subsumed by 13 via matching
23  (auto .)
24  Proof Completed.
```

Figure 5.4: The proof script for the partial correctness property of Counter.

In the proof header, the first command is always `select`; this specifies the object theory $\mathcal{R}$ that we will be reasoning about. The rest of the commands in the header can usually be supplied in any order. The `use tool` command on line 3 instructs the tool that the backend to be used when trying to solve validity problems for formulas in the `PRES-NAT` theory is the variant satisfiability backend (this works because `PRES-NAT` has a decidable satisfiability problem).

The `def-term-set` command specifies a set of patterns that define the terminating states for the given theory. Here, the only terminating state is the zero counter, since the decrement and increment rules assume that the counter is non-zero. Since we can directly express this pattern as the term `{0}` without constraints, we express the terminating state via the pattern `({0}) | true`, with `true` the constraint that always holds.

The `declare-vars` command specifies the sort of each declared variable. In this way, we need not later mention a variable's sort when it is used. This command is not strictly needed, but it makes complex goals easier to input.

The most important header command is `add-goal`, which adds a reachability formula to our set $\mathcal{C} = \{A_i \rightarrow^{\circledast} B_i\}_{i \in I}$ of reachability formulas to be proved. Recall that in the partial correctness example, the formula to be proved was $\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$. The tool's notation has only minor syntactic differences; the most important of which is all terms must be wrapped in parentheses.

Finally, we arrive at the proof body. Here, there are only two interesting commands so far: `auto` and `subsume`. The main proof command is `auto` which applies the default proof strategy to all goals. The `subsume` command must be invoked manually and uses one goal to subsume another, according to the goal subsumption simplification, which is justified by Lemma 5.9. Note that the structure of this tool-based proof follows logically the original high-level proof. The first application of `auto` applies the STEP$^{\forall}$ rule, since the AXIOM rule is not enabled yet, generating two successors. We then use the goal subsumption simplification to close one of these goals. The second invocation of the `auto` command applies the AXIOM rule and closes the second goal using the SUBSUMPTION rule.

Eventual Decrease of Counter.

As a second example, we prove that from any starting state, on all terminating paths (of which there are infinitely many), the counter will always pass through each natural number in the subsequence of numbers smaller than itself on its way to zero. The proof script with inline output is shown in Figure 5.5. This script is slightly more complicated than our first example; this is not surprising, since the high-level proof also required more steps to finish. Note that the proof header is identical to our previous example, except for an extra variable declaration and a different goal to be proved. Recall that any object theory terms that appear in any goal must be wrapped by parentheses, *including* any terms in the constraint.

Here, we see that the proof body exactly follows the high-level outline shown in Subsection 5.4.2. The `auto` and `subsume` commands on lines 17 and 23 apply the STEP$^{\forall}$ rule followed by a goal subsumption simplification in a way completely analogous to the previous proof. This is followed by the application of the CASE ANALYSIS auxiliary rule, two applications of two different variants of the SPLIT rule, concluding with a second invocation of `auto` that applies the SUBSTITUTION and SUBSUMPTION rules to close goals 17 and 18 and the AXIOM and SUBSUMPTION rules to close goal 19. Finally, as mentioned above, the tool automatically checks $T$-consistency and issues a warning on line 18 for a $T$-dubious goal, which here is no problem because the goal is immediately subsumed.

Let us look at these commands in more detail. The `case` command on line 25 corresponds to a CASE ANALYSIS application on a single variable in the named goal; currently, proving that the pattern covers the intended sort is a proof obligation that must be manually verified by the user. A future version of the tool will include automatic coverage checks when possible. The `replace` command on line 29 takes a goal identifier and a formula and replaces the original constraint of a goal precondition with the supplied formula; it exactly corresponds to the variant of the SPLIT rule that replaces a constraint using the pattern $\phi = \psi \vee \bot$. The `split by and` command on line 32 corresponds exactly to the fully general SPLIT rule that

```
(select COUNTER .)
Set module to COUNTER
(use tool varsat for validity on PRES-NAT .)
Loaded function varsat for validity
(def-term-set ({0}) | true .)
Added terminating state:
{0} | true
(declare-vars (N:Nat) U (M:Nat) U (K:Nat) .)
Declared variable(s):
{ K:Nat, M:Nat, N:Nat }
(add-goal count-red : ({N + 1}) | (M < N + 1) = (true) => ({M}) | true .)
Added goal(s):
[count-red : {1 + N:Nat} | true = M:Nat < 1 + N:Nat => {M:Nat} | true]
(start-proof .)
Started proof:
[1  | {1 + N:Nat} | true = M:Nat < 1 + N:Nat => {M:Nat} | true]
(auto .)
Warning: [9 | {&3:Nat} | true = M&4:Nat < 1 + &3:Nat => {M&4:Nat} | true]
may have terminated
Auto Results:
[13 | {&3:Nat} | true = M&4:Nat < 1 + &3:Nat => {M&4:Nat} | true]
[14 | {1 + 1 + &3:Nat} | true = M&4:Nat < 1 + &3:Nat => {M&4:Nat} | true]
(subsume 14 by 13 .)
Goal 14 could not be automatically subsumed; check subsumption manually
(case 13 on &3:Nat by (0) U (K + 1) .)
Case rule generated:
[15 | {0} | true = M&4:Nat < 1 + 0 => {M&4:Nat} | true]
[16 | {1 + K:Nat} | true = M&4:Nat < 1 + 1 + K:Nat => {M&4:Nat} | true]
(replace 15 by (M&4:Nat) = (0) .)
Split rule generated:
[17 | {0} | 0 = M&4:Nat => {M&4:Nat} | true]
(split 16 by (M&4:Nat < K:Nat + 1) = (true) and (K:Nat + 1) = (M&4:Nat).)
Split rule generated:
[18 | {1 + K:Nat} | M&4:Nat = 1 + K:Nat => {M&4:Nat} | true]
[19 | {1 + K:Nat} | true = M&4:Nat < 1 + K:Nat => {M&4:Nat} | true]
(auto .)
Proof Completed.
```

Figure 5.5: Eventual Decrease of Counter Proof Script.

performs a replacement of the form $\phi = \psi \vee \rho$. When attempting either kind of Split, the tool attempts a validity check to show that the formulas are semantically equivalent, printing a warning if it cannot verify the equivalence.

### 5.5.2 QLOCK Proof Example

Next let us use our tool to prove that mutual exclusion is an invariant for QLOCK. We describe a Maude specification of QLOCK in Figures 5.6 and 5.7 and then show the proof script (without inline output) in Figure 5.8. As before, we separate our example into two modules: an underlying functional module called `QLOCK-STATE` imported by the system module `QLOCK`.

Due to the fact that associative unification is, in general, *infinitary*, we make a few tweaks to the QLOCK signature $\Sigma$ and rewrite rules $R$ to ensure that Maude's associative unification algorithm will work.[15] Thus, the signature $\Sigma'$ of the Maude module `QLOCK` is identical to the signature $\Sigma$ *except* for the list constructor which has an extra sort *NeList* with subsorts $Nat < NeList < List$, two constructors $nil\ :\ \to List$, $\_;\_\ :\ NeList\ NeList \to NeList$, and where: (a) the original list concatenation operator $\_;\_\ :\ List\ List \to List$ is no longer a constructor; (b) the operator $\_;\_$ only uses built-in associativity axioms; and (c) the built-in identity axiom designation is replaced by a pair of equations explicitly defining the identity axiom. The set of rewrite rules $R'$ of the `QLOCK` module are slightly more verbose than the rules $R$ in the original specification. Since we are no longer matching modulo associativity and identity, but only modulo associativity, more patterns are needed. Specifically, the rules *n2w*, *w2c*, and *c2n* now have two versions, corresponding to *nil* and non-*nil* *List* substitutions.

Note that, unlike `COUNTER`, the module `QLOCK` has the conditional rule *join*; Maude requires conditional rewrite rules to be declared with the `crl` keyword. Finally, recall that in order to prove invariants, we extend our theory with a *stop* rule; this is done in the module `QLOCK-stop`, which is the specific instance for this example of the general theory construction $\mathcal{R}_{stop}$ presented earlier.

Let us now describe the proof script in Figure 5.8. After replacing associativity plus identity matching patterns by associativity-only ones, the proof script proceeds as we would expect. In fact, inputting the goals and applying `auto` is enough. Note the new command `use strat` on lines 31-33. This command selects which axioms to use when applying the Axiom rule to a goal and its descendants; it is necessary because each goal corresponding to one of the patterns in the disjunct $P'$ may reach any of the others via one of the rewrite rules.

Finally, we complete the proof script with two applications of `auto`. The first `auto` command applies the Step$^{\forall}$ rule, generating 30 successors from the three initial goals. The second `auto` closes each generated goal either by: (a) failing to satisfy the condition on the $w2c$ rule and being removed by the vacuousness check; or (b) applying the Axiom rule followed by the Subsumption rule.

[15] Specifically, we require that: (1) associative symbols do *not* also have identity axioms; (2) associative lists to be unified do *not* share variables of the list sort (see [98]).

```
fmod QLOCK-STATE is
  sorts Nat MSet NeList List Pred .
  subsort Nat < MSet .
  subsort Nat < NeList < List .

  op 0    : -> Nat [ctor] .
  op s_   : Nat -> Nat [ctor] .
  op __   : MSet MSet -> MSet [ctor assoc comm id: mt] .
  op mt   : -> MSet [ctor] .
  op nil  : -> List [ctor] .
  op _;_  : List   List   -> List   [assoc] .
  op _;_  : NeList NeList -> NeList [ctor assoc] .
  op tt   : -> Pred [ctor] .
  op dupl : MSet -> Pred [ctor] .

  var N : Nat . var S : MSet . var L : List .
  eq dupl(N N S) = tt [variant] .
  eq L ; nil = L [variant] .
  eq nil ; L = L [variant] .
endfm
```

Figure 5.6: Theory Specification for the QLOCK State

```
mod QLOCK is pr QLOCK-STATE .
  sort Conf State .
  op _|_|_|_ : MSet MSet MSet List -> Conf  [ctor] .
  op <_>     : Conf                -> State [ctor] .

  var I W C : MSet . var L : NeList . var M N : Nat . var CNF : Conf .
  --- n2w
  rl < I M | W   | C   | L     > => < I   | W M | C   | L ; M  > .
  rl < I M | W   | C   | nil   > => < I   | W M | C   | M      > .
  --- w2c
  rl < I   | W N | C   | N ; L > => < I   | W   | C N | N ; L > .
  rl < I   | W N | C   | N     > => < I   | W   | C N | N     > .
  --- c2n
  rl < I   | W   | C M | N ; L > => < I M | W   | C   | L      > .
  rl < I   | W   | C M | N     > => < I M | W   | C   | nil    > .
  --- exit/join
  rl < I M | W   | C   | L     > => < I   | W   | C   | L      > .
 crl < I   | W   | C M | L     > => < I M | W   | C   | L      >
  if dupl(I M) =/= tt .
endm

mod QLOCK-stop is pr QLOCK .
  op [_] : Conf -> State [ctor] .
  rl < CNF:Conf > => [ CNF:Conf ] .
endm
```

Figure 5.7: Theory Specification for the QLOCK Example

```
(select QLOCK-stop .)
(use tool varsat for validity on QLOCK-STATE .)
(def-term-set ([C:Conf]) | true .)
(declare-vars (I:MSet) U (I':MSet)   U (W:MSet)   U (W':MSet) U
(N:Nat)  U (M:Nat)     U (N':Nat)   U (M':Nat)  U
(Q:List) U (NQ:NeList) U (Q':List) .)
(def-term-set ([C:Conf]) | true .)

(add-goal mutex1 : (< I  | W  | mt | Q       >) |
(dupl(I W)) =/= (true)
=>
([ I' | W' | N' | M' ; NQ ]) | (N') = (M')      \/
([ I' | W' | N' | M'      ]) | (N') = (M')      \/
([ I' | W' | mt | Q'      ]) | true             .)

(add-goal mutex2a : (< I  | W  | N  | M      >) |
(dupl(I W N)) =/= (true) /\ (N) = (M)
=>
([ I' | W' | N' | M' ; NQ ]) | (N') = (M')      \/
([ I' | W' | N' | M'      ]) | (N') = (M')      \/
([ I' | W' | mt | Q'      ]) | true             .)

(add-goal mutex2b : (< I  | W  | N  | M ; Q  >) |
((dupl(I W N)) =/= (true) /\ (N) = (M))
=>
([ I' | W' | N' | M' ; NQ ]) | (N') = (M')      \/
([ I' | W' | N' | M'      ]) | (N') = (M')      \/
([ I' | W' | mt | Q'      ]) | true             .)

(start-proof .)
(on 1 use strat mutex1 mutex2a mutex2b .)
(on 2 use strat mutex1 mutex2a mutex2b .)
(on 3 use strat mutex1 mutex2a mutex2b .)
(auto .)
(auto .)
```

Figure 5.8: QLOCK Mutual Exclusion Proof Script

### 5.5.3 Other Examples

To validate the feasibility of our approach we also verified properties for a collection of examples. Table 5.1 summarizes these experiments; it is subdivided into sections based on the kind of example, i.e., communication protocols, mutual exclusion algorithms, programs written in a simple imperative programming language IMP, and examples that do not belong in any of the other categories. For complete details, refer to http://maude.cs.illinois.edu/tools/rltool/.

Table 5.1: Examples Verified by the Tool

| Example | Decidable | Goals | Leaves | Lemmas | Auto |
|---|---|---|---|---|---|
| Simple Comm. Protocol | No | 1 | 2 | 0 | Yes |
| Fault-Tolerant Comm. Protocol | Yes | 6 | 21 | N/A | Yes |
| Dijkstra's Mutex Algorithm | Yes | 3 | 218 | N/A | Yes |
| QLOCK | Yes | 4 | 71 | N/A | No |
| Unbounded Lamport's Bakery | Yes | 11 | 827 | N/A | No |
| Readers/Writers Problem | Yes | 2 | 5 | N/A | Yes |
| Fixed-Size Token Ring | Yes | 3 | 7 | N/A | Yes |
| IMP Factorial Function | No | 1 | 5 | 1 | Yes |
| IMP Fibonacci Function | No | 1 | 5 | 1 | Yes |
| IMP Multiplication Function | No | 1 | 5 | 1 | No |
| IMP Remainder Function | No | 1 | 10 | 0 | No |
| Bank Account | No | 1 | 7 | 2 | Yes |
| Non-Deterministic Choice | Yes | 1 | 5 | N/A | No |
| Counter Eventual Decrease | No | 1 | 4 | 0 | No |
| Counter Partial Correctness | No | 1 | 2 | 0 | No |
| List Sorting Element Preservation | No | 1 | 5 | 0 | Yes |

For each example the table shows whether satisfiability of quantifier-free formulas in the initial algebra defined by the equational part of the example theory is decidable or not *as far as the equational formulas involved in the property to be verified and in rule conditions are concerned.*[16] The table also shows the number of initial goals/invariants, the number of leaves in the proof tree, the number of inductive lemmas needed for verifying examples in undecidable theories, and whether the built-in strategy can find a proof (aside from finding necessary circularities), i.e. whether the user needed to manually apply a derived rule to complete the proof. Below, we briefly describe each example and property verified in the table above.

We defined two order-preserving communication protocols assuming a single sender, receiver, and channel: (1) a simple communication protocol that is not fault-tolerant with a unidirectional channel as well as (2) a fault-tolerant protocol with a bidirectional channel that uses acknowledgments to confirm that messages are received. For the simple protocol, we verified that the final state contains the message sent in the correct order, so that the sequence number corresponds correctly to that of messages sent; the undecidability comes from the fact that counting sequence numbers of a set of messages requires non-FVP recursive equations. For the fault-tolerant protocol, we verified that the message sent is received in the correct order. Note that in-order message reception reduces to an equality check over an associative list of naturals.

We also defined five classic mutual exclusion algorithms. In each case the reachability goal proved was always the invariant stating that the mutual exclusion algorithm never allows two or more processes to

[16] For example, in the QLOCK specification, due to the use of an associativity axiom for queues, decidable satisfiability of arbitrary quantifier-free formulas in the initial algebra of QLOCK cannot be ensured by variant satisfiability methods. But *is* ensured by variant satisfiability for equational formulas whose equations only involve terms of sorts either *MSet* or *Pred*.

enter the critical section at the same time. For each of these algorithms the system in question was never-terminating and thus required the techniques introduced in Section 5.3.1 to carry out the proof.

Additionally, we defined a rewrite theory that specifies the semantics of a simple programming language we call IMP. Its expressions range over two data types: booleans and natural numbers. The supported operations are addition, subtraction, multiplication, boolean negation, and boolean conjunction; expressions are side-effect free. Variables all have the natural number type, share a global namespace, and must be declared before the program body executes. Supported statements include assignment, if-statements, and while-loops. Note that, since IMP does not have dynamic memory allocation, heap-based reasoning is not needed.

We wrote four functions in IMP and verified their correctness: multiplication, factorial, remainder, and the function that returns the n-th element of the Fibonacci sequence. In each case, the property verified is that the IMP function implements the same function as one equationally defined in Maude. Since each of the functions verified is inherently recursive, verifying their correctness is generally undecidable. An interesting feature of the tool-based proofs for these IMP programs is that, even though there are few leaves in the proof tree, the depth of the proof trees is considerably longer than for other examples, because the program must be unfolded through many steps before reaching a state captured by one of our loop invariants.

Finally, we verified a few other examples that do not fit in any of the previous categories, including the two counter examples discussed in Subsection 5.4.2. We also gave a specification of a bank account that allows deposits and withdrawals to nondeterministically occur, where each withdrawal occurs in two steps: the withdrawal is initiated and, at some later time, the withdrawal is completed. For this bank account specification we proved the invariant that a bank account where the pending withdrawals are initially less than the balance will never overdraft later. We defined an algorithm that takes a multiset of natural numbers and nondeterministically throws numbers away and proved that this algorithm, when supplied with a non-empty multiset as its starting state, will always reach a singleton contained in the original set. Finally, we defined a list sorting specification and proved that the multiset of elements belonging to the partially sorted list remains invariant.

## 5.6 RELATED WORK AND CONCLUSIONS

### 5.6.1 Related Work

Reachability logic [90, 99, 26, 27] is a language-generic approach to program verification, parametric on the operational semantics of a programming language. Both Hoare logic and separation logic can be naturally mapped into reachability logic [90, 99]. This work, based on our earlier work in [100], extends reachability logic from a programming-language-generic logic of programs to a rewrite-theory-generic logic to reason about *both* distributed system designs and programs, based on their rewriting logic semantics. This extension is non-trivial and requires a number of new concepts and results, including: (i) relativization of terminating sequences to a chosen subset $[\![T]\!]$ of terminating states; (ii) solving the "invariant paradox," to reason about invariants and co-invariants of possibly non-terminating systems, and characterizing such invariants by means of reachability formulas through a theory transformation; and (iii) making it possible to achieve higher levels of automation by systematically basing the state predicates on positive Boolean combinations of constrained constructor patterns of the form $u \mid \varphi$ with $u$ a constructor term.

In contrast, standard reachability logic [26, 27] uses matching logic, which assumes a first-order model $\mathcal{M}$

and its satisfaction relation $\mathcal{M} \models \varphi$ as the basis of the reachability logic proof system, and further assumes a matching-logic-definable transition relation on $\mathcal{M}$. As discussed in Section 5.2, we choose $T_{\Sigma/E\cup B}$ as the model and $\rightarrow_{\mathcal{R}}$ for transitions, rather than some general $\mathcal{M}$ with definable transitions, and systematically exploit the isomorphism $T_{\Sigma/E\cup B}|_{\Omega} \cong T_{\Omega/E_{\Omega}\cup B_{\Omega}}$, allowing us to use unification, matching, narrowing, and satisfiability procedures based on the typically much simpler initial algebra of constructors $T_{\Omega/E_{\Omega}\cup B_{\Omega}}$. This has the advantage that we can explicitly give the complete details of our inference rules (e.g., how SUBSUMPTION checks for subsumption, or STEP$^{\forall}$ ensures that states have at least a successor), instead of relying on a general satisfaction relation $\models$ on some $\mathcal{M}$ with definable transitions. The result is a simpler inference system with only three rules (instead of the eight in reachability logic). On the practical side, reachability logic has been previously implemented as part of the $\mathbb{K}$ framework, and has only been instantiated with operational semantics of programming languages and used for the purpose of program verification. In particular, the implementation in $\mathbb{K}$ has several hand-crafted heuristics for reasoning about specific features of programming languages, such as dynamically allocated memory (the "heap"). In spite of the fact that similar heuristics have not yet been added to the current prototype described in Section 5.5, the potential for automation of the constructor-based reachability logic approach has been demonstrated by the tool's capacity to prove relevant properties for a representative suite of distributed system designs, including various distributed system designs and algorithms as well as programs in a simple imperative programming language. Of course, this is a proof of concept: improving the tool by adding reasoning heuristics, e.g., attempting to guess inductive axioms for loops, as well as more powerful inductive validity checking support will be crucial to scale up to bigger applications.

As mentioned in the Introduction, we have been inspired by the work in [29]. We agree on the common goal of making reachability logic rewrite-theory-generic, but differ on the methods used and their applicability. Main differences include: (1) The authors in [29] do not give an inference system but a verification algorithm manipulating goals, which makes it hard to compare both logics. (2) The theories to which the methods in [29] apply seem more restricted than the ones presented here. Roughly, (see their **Assumption 3**) [29] assumes restrictions akin to those imposed in [101] to allow "rewriting modulo SMT," which limits the equational theories $(\Sigma, E)$ that can be handled. (3) Matching is used throughout in [29] instead of unification. This means that, unless a formula has been sufficiently instantiated, no matching rule may exist, whereas unification with some rule is always possible in our case. (4) No method for proving invariants is given in [29]; our solving of the "invariant paradox" provides such a method.

Three recent further developments that add coinductive reasoning capabilities to reachability logic are also worth mentioning, namely: (1) Moore's Ph.D. dissertation [102]; (2) the coinductive approach by Lucanu et al. in [103]; and (3) the coinductive approach to reachability logic by Ciobâcă and Lucanu in [104]. The closest to our work are the approaches in [103] and, even more so, [104]. The approach in [103] adopts a semantic framework for models similar to the already-discussed work in [26, 27], i.e., state properties are specified using matching logic and assume a given first-order logic model. In this regard, the already discussed model theoretic differences between our work and that in [26, 27], as well as the associated advantages and disadvantages, seem to be essentially the same as for [103]. However, an important contribution of the work in [103] is its coinductive semantics and justification for circular co-inductive reasoning. The relationship to our work is that the circular coinduction inference rule in [103] roughly corresponds to our AXIOM rule, where formulas that have become available as circularities provide a kind of "seven league boots" to advance the proof process and eventually finish it. Perhaps the work closest to ours in the coinductive approach is that of Ciobâcă and Lucanu in [104]. At a very high level, it seems fair to say that regarding the models assumed, the kinds of reachability properties proved, and the state predicates and inference systems proposed, the approach in [104] and ours are quite close. Of course, one important difference is that, to achieve essentially the same objectives, their semantic approach uses coinductive reasoning, whereas ours, particularly in the proof of our Soundness Theorem 5.4, uses inductive or, more precisely, minimal counter-example reasoning. There are however other substantial differences:

(i) rewriting, as in our case, is based on conditional rules with formulas as constraints in conditions; but in [104] the topmost requirement is dropped, as is also the possibility of matching and rewriting modulo axioms $B$, except for the fact that, using "built-in constraints" in an (expected to be decidable) reduct model on built-in sorts, one could achieve the effect of rewriting modulo such a built-in decidable reduct model;

(ii) their reachability formulas are less general than ours: in our notation their formulas have the form $u \mid \varphi \to^{\circledast} v \mid \psi$, i.e., only atomic patterns predicates can be used, but as our QLOCK example shows, having disjunctions of atomic pattern predicates in midconditions is very useful in practice;

(iii) although the inference systems are relatively close, they are not in a one-to-one correspondence:

   (a) their [*axiom*] rule corresponds to the subcase of our Subsumption rule that discharges vacuous formulas,

   (b) their [*subs*] and [$der^{\forall}$] rules roughly correspond to the effect of our Step$^{\forall}$ and Subsumption rules (which in [100] were combined into a single Step$^{\forall}$ + Subsumption rule); but this comes with a slight twist: their [*subs*] rule does the job of our Subsumption rule *and*, roughly, that of the formula $\varphi'$ in our Step$^{\forall}$ rule, whose purpose is to restrict the (in their sense "derived") goals after one rewriting step to states not already subsumed by the precondition,

   (c) their [*circ*] rule and our Axiom rule basically agree with each other.

Another area of related work is that of *deductive proofs of safety properties*, particularly of *invariants*, for systems specified in rewriting-based languages such as CafeOBJ [13] and Maude [6]. The two main approaches that have been developed in this area are:

1. The CafeOBJ approach to the specification of transition systems and the verification of their invariants using either so-called *proof scores* (which can use CafeOBJ itself or Maude directly as a theorem prover in the spirit of, say, [105]) as in, e.g., [106], or by direct use of an inductive theorem prover or a combination of standard inductive theorem proving and score-based theorem proving as in, e.g., [107, 108]. An important feature of this approach, illustrated in the proof-score case but also applicable to the standard inductive theorem proving or mixed approaches is that *both* proofs of *invariants* and of purely equational inductive verification conditions associated to both invariants and inductive properties of algebraic data types can be carried out.

2. The *Invariant Analyzer* (InvA) Maude-based approach to the deductive verification of invariants and other safety properties of a concurrent system specified as a topmost rewrite theory [109, 110, 111]. The key ideas in the InvA approach include: (i) the proof of an invariant is inductively reduced to proving that all one-step transitions with the rules $R$ preserve the invariant; (ii) using unification and narrowing symbolic techniques, the proof that each system transition preserves the invariant is *reduced* to proving purely equational inductive verification conditions in the underlying algebraic data type of

states (that is, in $T_{\Sigma/E\cup B}$ if the rewrite theory has the form $\mathcal{R} = (\Sigma, E \cup B, R)$); and (iii) an inductive theorem prover (in this case Maude's ITP) is used to discharge the generated verification conditions.

Generally speaking, although technically the CafeOBJ-based and Maude-based approaches are different and not directly comparable, their main goals, namely, the deductive verification of transition systems with emphasis on their safety properties, are quite close and these two approaches have stimulated each other. In comparison with constructor-based reachability logic, the following remarks can be made: (i) the verification of *invariants* in constructor-based reachability logic is closely related to both invariant verification in InvA and in the CafeOBJ-based tools (very roughly speaking, in reachability logic, the equational verification conditions in these two other approaches are replaced by the symbolic methods and side conditions involved in applying the reachability logic inference rules); (ii) of course, the more general reachability properties, including in particular the Hoare logic partial correctness properties, do not have a counterpart in the CafeOBJ-based and InvA approaches; (iii) the use of *pattern predicates* in reachability logic seems to be new: in the other two approaches state predicates are typically specified by Boolean predicates; we conjecture that pattern predicates and their associated symbolic techniques should also be quite useful in future developments of the InvA and Cafe-OBJ approaches; and (iv) last but not least, in [106], besides proving invariants of transition systems, a new method for proving *leads to* properties using proof scores is also presented. Such properties are *liveness* temporal logic properties beyond the usual safety properties. This is quite intriguing, and suggests investigating whether leads-to properties could also be verified in a future version of reachability logic.

Finally, there is also a close connection between this work and other rewriting-based symbolic methods, including: (i) unification modulo FVP theories [24]; (ii) decidable satisfiability (and validity) of quantifier-free formulas in initial algebras [14, 16, 31, 79, 18, 80, 81, 82, 83, 84, 36]; (iii) narrowing-based reachability analysis [78, 87]; (iv) narrowing-based proof of safety properties [109, 111]; (v) patterns and constrained patterns [80, 87]; and (vi) rewriting modulo SMT [101]. Exploiting such connections, particularly with [24, 36, 47, 87], has been essential to achieve the goals of this chapter.

### 5.6.2 Conclusions

In conclusion, this chapter advances the goal of making reachability logic available as a rewrite-theory-generic verification logic. The goals of wide applicability, invariant verification, simplicity, and mechanization of inference rules have been substantially advanced, but much work remains ahead. The feasibility of the approach has been validated with our prototype implementation using a suite of representative examples. They show that, both for reasoning about distributed protocols and algorithms and for proving properties of programs in conventional languages, the verification approach presented here seems promising and feasible in practice. However, the examples are relatively small, and the prototype tool implementation should be further improved, automated, and endowed with more powerful backends for inductive validity checking. Hand in hand with this, both the proof methods and the tool capabilities should be stressed by means of more substantial case studies, both of distributed system designs and of programming language verification.

At the foundational level, several problems deserve further research, including: (i) a *relative completeness* proof for a suitable future version of constructor-based reachability logic; (ii) investigation of additional temporal logic properties that could be expressed in reachability logic; a case in point is that of the leads-to properties already discussed in connection with the work in [106]; and (iii) how to exploit modularity and parameterization at the level of rewriting logic specifications (in the sense of parameterized rewrite theories) to make reachability logic proofs as modular, generic and reusable as possible.

All this, together with reaching a mature tool implementation, are among our current goals for the near future.

# CHAPTER 6 IBOS CASE STUDY

## 6.1 INTRODUCTION

In the previous chapter, we saw in Sections 5.2 and 5.3 how several techniques, such as exploiting the good properties of the canonical term algebra and constrained constructor pattern predicates support our proof system's algorithmic description and automatability. However, these claims do not hold much weight practically unless they can be substantiated by a suite of relevant case studies demonstrating the practical effectiveness of the logic. Though Chapter 5 presented many examples, they are not substantial enough to be anything more than toys. In this chapter, we rectify this situation by presenting a significantly more substantial case study involving the verification of several security properties for the Illinois Browser Operating System (IBOS).

**Rationale and Origins**. Web browsers have in fact become operating systems for a myriad of web-based applications. Given the enormous user base and the massive increase in web-based application areas, browsers have for a long time been a prime target for security attacks, with a seemingly unending sequence of browser security violations. One key reason for this problematic state of affairs is the enormous size (millions of lines of code) and sheer complexity of conventional browsers, which make their formal verification a daunting task. An early effort to substantially improve browser security by formal methods was jointly carried out by researchers at Microsoft Research and the University of Illinois at Urbana-Champaign (UIUC), who formally specified Internet Explorer (IE) in Maude [6], and model checked that formalization finding 13 new types of unknown address bar or status bar spoofing attacks in it [112]. Those vulnerabilities were all corrected in IE *before* [112] was published. But the research in [112] just uncovered *some* kinds of possible attacks, and the sheer size and complexity of IE made full verification unfeasible. This stimulated a team of systems and formal methods researchers at UIUC to ask the following question: *could formal methods be used from the very beginning in the design of a secure browser with a very small trusted code base (TCB) whose design could be verified?* The answer given to this question was the Maude-based design, model checking verification, and implementation of the IBOS Browser cum operating system, [113, 114, 115, 116], with a 42K line trusted code base (TCB), several orders of magnitude smaller than the TCBs of commodity browsers.

**Why this Work**. As further explained in Section 6.4, only a model checking verification of the IBOS security properties relying on a hand-proof abstraction argument for its full applicability was possible at the time IBOS was developed [115, 116, 115, 110]. A subsequent attempt at a full deductive verification of IBOS in [110] had to be abandoned due to the generation of thousands of proof obligations. In retrospect, this is not surprising for two reasons: (1) Many of the symbolic techniques needed to scale up the IBOS deductive verification effort, including variant unification and narrowing [24], order-sorted congruence closure module axioms [117], and variant-based satisfiability [25, 88], did not exist at the time. In the meantime, those symbolic techniques have been developed and implemented in Maude. (2) Also missing was a *program logic* generalizing Hoare logic for Maude specifications in which properties of concurrent systems specified in Maude could be specified and verified. This has been recently addressed with the development of a *constructor-based reachability logic* for rewrite theories in [100], which extends prior reachability logic research on verification of conventional programs using K in [90, 99, 26, 27]. In fact, what has made possible the deductive proof of the IBOS security properties presented in this chapter is precisely the combination of the strengths from (1) and (2) within the reachability logic theorem prover that we have developed for carrying out such a proof. Of course, implicit in both (1) and (2) are two important proof obligations: (a) Both our symbolic reasoning and reachability logic engines take as input a rewrite theory $\mathcal{R}$. However, like most efficiently scalable verification methods, these methods are *not sound* for any arbitrary $\mathcal{R}$. Thus, we require methods to validate that the input theory $\mathcal{R}$ is *suitable* for symbolic reachability analysis, i.e., ground convergent and sufficiently complete. (b) [115, 116, 115] proved that the IBOS formalization satisfies certain security properties which are specifiable an invariant $I_0$. Our analysis uses a slightly modified invariant $I$ that is also *inductive* (as explained in Section 6.1.2). Thus, we require that $I$ is *at least* as strong as or stronger than $I_0$ to ensure that our security property specification does not miss any cases covered by prior work. Unlike prior work, both requirements (a) and (b) are now fully checked as explained in Appendix E. Last but not least, as we further explain in Section 6.4, the IBOS browser security goals remain as relevant and promising today as when IBOS was first developed, and this work bring us closer to achieving those goals.

**Main Contributions** include: (A) The first full *deductive verification of the IBOS browser* as explained above. (B) A general *modular proof methodology* for scaling up reachability logic proofs of object-based distributed systems that has been invaluable for verifying IBOS, but has a much wider applicability to general distributed system verification. (C) A substantial and useful *case study* that can be of help to other researchers interested in both browser verification and distributed system verification. (D) A *reachability logic prover*, which in the course of this research has evolved from the original prototype reported in [100] to a first prover version to be released in the near future.

**Chapter Overview**.

The notion of object-based rewrite theories and a recap of invariant verification in reachability logic are presented in the preliminaries. IBOS, its rewriting logic Maude specification, and the specification of its security properties are explained in Section 6.2. The deductive proof of those IBOS properties and the modular proof methodology used are described in Section 6.3. Section 6.4 discusses related work and concludes the chapter.

### 6.1.1 Object-Based Rewrite Theories

Most distributed systems, including the IBOS browser, can be naturally modeled by *object-based rewrite theories*. We give here a brief introduction and refer to [118, 6] for more details. The *distributed state* of an object-based system, called a *configuration*, is modeled as a *multiset* or "soup" of objects and messages built up by an *associative-commutative* binary multiset union operator (with juxtaposition syntax) $\_\ \_ :$ $\mathit{Conf}\ \mathit{Conf} \to \mathit{Conf}$ with identity *null*. The sort *Conf* has two subsorts: a sort *Object* of *objects* and a sort *Msg* of *messages* "traveling" in the configuration from a sender object to a receiver object. The syntax for messages is user-definable, but it is convenient to adopt a conventional syntax for objects as record-like structures of the form: $\langle\, o \mid a_1(v_1), \ldots, a_n(v_n)\rangle$, where $o$ is the object's name or *object identifier*, belonging to a subsort of a general sort *Oid*, and $a_1(v_1), \ldots, a_n(v_n)$ is a *set* of object *atributes* of sort *Att* built with an associative-commutative union operator $\_,\_ : \mathit{Atts}\ \mathit{Atts} \to \mathit{Atts}$, with *null* as identity element and with $\mathit{Att} < \mathit{Atts}$. Each $a_i$ is a constructor operator $a_i : s_i \to \mathit{Att}$ so that the *data value* $v_i$ has sort $s_i$. Objects can be classified in *object classes*, so that a class $C$ has an associated subsort $C.\mathit{Oid} < \mathit{Oid}$ for its object identifiers and associated attribute constructors $a_i : s_i \to \mathit{Att}$, $1 \leqslant i \leqslant n$. Usually, a configuration may have many objects of the same class, each with a different object identifier; but some classes (e.g., the *Kernel* class in IBOS) are *singleton classes*, so that only one object of that class, with a fixed name, will apear in a configuration. Another example in the IBOS specification is the singleton class *Display*. The single display object represents the rendering of the web page shown to the user and has the form: $< display \mid displayContent(D), activeTab(WA) >$, where the *activeTab* attribute constructor contains a reference to the web process that the user has selected (each tab corresponds to a different web process) and the *displayContent* constructor encapsulates the web page content currently shown on the display. Not all configurations of objects and messages are sensible. A configuration is *well-formed* iff it satisfies the following two requirements: (i) *unique object identifiers*: each object has a unique name different from all other object names; and (ii) *uniqueness of object attributes*: within an object, each object attribute *appears only once*; for example, an object like $< display \mid displayContent(D), activeTab(WA), activeTab(WA') >$ is nonsensical.

The *rewrite rules* $R$ of an object-based rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ have the general form $l \to r$ *if* $\phi$, where $l$ and $r$ are terms of sort *Conf*. Intuitively, $l$ is a pattern describing a *local fragment* of the overall configuration, so that a substitution instance $l\sigma$ describing a concrete such fragment (e.g., two objects, or an object and a message) can be rewritten to a new subfragment $r\sigma$, provided the rule's condition $\phi\sigma$ holds (see Section 6.2.1 for an example rule). Classes can be structured in *multiple inheritance hierarchies*, where a subclass $C'$ of class $C$, denoted $C' < C$, may have additional attributes and has a subsort $C'.Oid < C.Oid$ for its object identifiers. By using extra variables $Attrs_j$ of sort *Atts* for "any extra attributes" that may appear in a subclass of any object $o_j$ in the left-hand side patterns $l$ of a rule, rewrite rules can be *automatically inherited by subclasses* [118]. Furthermore, a subclass $C' < C$ may have *extra rules*, which may modify both its superclass attributes and its additional attributes.

An object-based rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ *can be easily be made topmost* as follows: (i) we add a fresh now sort *State* and an "encapsulation operator," say, $\{\_\} : Conf \to State$, and (ii) we trasform each rule $l \to r$ *if* $\phi$ into the rule $\{l\ C\} \to \{r\ C\}$ *if* $\phi$, where $C$ is a fresh variable of sort *Conf* modeling "the rest of the configuration," which could be empty by the identity axiom for *null*.

### 6.1.2 Proving Inductive Invariants [100, 119].

For a transition system $\mathcal{Q} = (Q, \to_{\mathcal{Q}})$ and a subset $Q_0 \subseteq Q$ of *initial states*, a subset $I \subseteq Q$ is called an *invariant* for $\mathcal{Q}$ from $Q_0$ iff for each $a \in Q_0$ and $b \in Q$, $a \to^*_{\mathcal{Q}} b$ implies $b \in I$, where $\to^*_{\mathcal{Q}}$ denotes the reflexive-transitive closure of $\to_{\mathcal{Q}}$. A subset $A \subseteq Q$ is called *stable* in $\mathcal{Q}$ iff for each $a \in A$ and $b \in Q$, $a \to_{\mathcal{Q}} b$ implies $b \in A$. An invariant $I$ for $\mathcal{Q}$ from $Q_0$ is called *inductive* iff $I$ is stable. We instantiate this generic framework to prove invariants over a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $(\Sigma, B, \vec{E})$ convergent and sufficiently complete with respect to $(\Omega, B_\Omega)$ and consider the transition system induced by $\mathcal{R}$ over sort *State*, i.e., $(C_{\Sigma/E \cup B, State}, \to_{\mathcal{R}})$.

To prove an invariant $I$ from $Q_0$ over $\mathcal{R}$, we use a simple theory transformation mapping topmost theory $\mathcal{R}$ to a theory $\mathcal{R}_{stop}$ having a fresh operator $[\_]$ and a rule $stop : c(\vec{x}) \to [c(\vec{x})]$ for each constructor $c$ of sort *State*. Then, by Corollary 1 in [100], to prove $I$ is an invariant from $Q_0$ over $\mathcal{R}$ we prove $Q_0 \subseteq I$ and that the reachability formula $I\sigma \to^{\circledast} [I]$ holds over $\mathcal{R}$, where: (i) $I\sigma$ is a renaming of $I$ with $vars(I) \cap vars(I\sigma) = \varnothing$ and (ii) if $I = (u \mid \varphi)$ then $[I] = ([u] \mid \varphi)$. If $I$ is inductive and $I = u \mid \varphi$, the proof of $I\sigma \to^{\circledast} [I]$ proceeds as follows:

1. The initial sequent is $[\varnothing, I\sigma \to^{\circledast} [I]] \vdash_{[]} I\sigma \to^{\circledast} [I]$.

2. Apply the Step rule; based on which rule $\zeta : l \to r$ if $\phi$ was used, obtain:

   (a) if $\zeta$ is *stop*, $[I\sigma \rightarrow^{\circledast} [I], \varnothing] \vdash_{[]} [I\sigma] \rightarrow^{\circledast} [I]$;
   (b) otherwise, $\bigwedge_{\alpha \in Unif_{B_\Omega}(u,l)} [I\sigma \rightarrow^{\circledast} [I], \varnothing] \vdash_{[]} (r \mid \varphi \wedge \phi)\alpha \rightarrow^{\circledast} [I]$.

3. For case (2a), apply Subsumption. For case (2b), apply zero or more derived rules to obtain sequents of the form: $[I\sigma \rightarrow^{\circledast} [I], \varnothing] \vdash_{[]} A \rightarrow^{\circledast} [I]$.

4. Since $I$ is assumed inductive, we have $[\![A]\!] \subseteq [\![I\sigma]\!]$. Thus, apply Axiom to derive $[I\sigma \rightarrow^{\circledast} [I], \varnothing] \vdash_{[]} [I] \rightarrow^{\circledast} [I]$.

5. Finally, apply Subsumption.

Since all the IBOS security properties *are invariants*, all our proofs follow steps (1)–(5) above.

## 6.2 IBOS AND ITS SECURITY PROPERTIES

One important security principle adopted by all modern browsers is the same-origin policy (SOP). SOP isolates web apps from different origins, where an origin is represented as a tuple of a protocol, a domain name, and a port number. For example, if a user loads a web page from origin (`https`,`mybank.com`,`80`) in one tab and in a separate tab loads a web page from origin (`https`,`mybnk.com`,`80`), i.e., a spoofed domain that omits the 'a' in 'mybank,' any code originating from the spoofed domain in the latter tab will not be able to interact with the former tab. SOP also ensures that asynchronous JavaScript and XML (AJAX) network requests from a web app are routed to its origin. Unfortunately, browser vendors often fail to correctly implement the SOP policy [120, 121].

The Illinois Browser Operating System (IBOS) is an operating system and web browser that was designed and modeled in Maude with security and compatibility as primary goals [113, 114, 116]. Unlike commodity browsers, where security checks are implemented across millions of lines of code, in IBOS a small trusted computing base of only 42K code lines is established in the *kernel* through which all network interaction is filtered. What this means in practice is that, even if highly complex HTML rendering or JavaScript interpretation code is compromised, the browser *still* cannot violate SOP (and several other security properties besides). The threat model considered here allows for much of the browser code itself to be compromised while still upholding the security invariants we describe below.

In the following subsections we provide a general overview of the IBOS browser, how is formally specified as a rewriting logic theory in Maude, how the SOP can be concretely understood for this system, and how the SOP and other IBOS security properties can be formally specified as invariants.

### 6.2.1 IBOS System Specification

**IBOS System Design.** The Illinois Browser Operating System is an operating system and web browser designed to be highly secure as a browser while maintaining compatibility with modern web apps. It was built on top of the micro-kernel L4Ka::Pistachio [122, 123], which embraces the principles of least privilege and privilege separation by separating operating subsystems into separate daemons that communicate through the kernel via checked inter-process communication (IPC). IBOS directly piggybacks on top of this micro-kernel design by implementing various browser abstractions, such as the browser chrome and network connections, as separate components that communicate using L4ka kernel message passing infrastructure. Figure 6.1 gives an overview of the IBOS architecture; as an explanatory aid we highlight a few key objects:

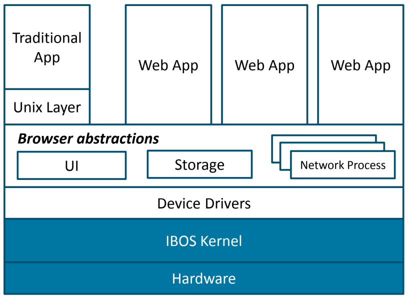


Figure 6.1: IBOS System Architecture [116].

- **Kernel.** The IBOS kernel is built on top of the L4Ka::Pistachio micro-kernel which, as noted previously, can check IPC messages against security policies.
- **Network Process.** A network process is responsible for managing a network connection (e.g., HTTP connections) to a specific origin. It understands how to encode and decode TCP datagrams and Ethernet frames and can send and receive frames from the network interface card (NIC).
- **Web Application.** A web application represents a specific instance of a web page loaded in a particular browser tab (e.g., when a link is clicked or a URL is entered into the address bar). Web applications know how to render HTML documents. As per SOP, each web page is labeled by its origin.
- **Browser UI.** The browser user inferface (UI) minimally includes the address bar and the mouse pointer and extends to any input mechanism.
- **Display.** The display represents the rendered web app shown to the user; it is blank when no web app has loaded. For security, it cannot modify the UI.

**IBOS System Specification in Maude.** We present an overview of the IBOS formal executable specification as a Maude rewrite theory, which closely follows previous work [116, 115, 110]. We model IBOS as an object-based rewrite theory (see Section 6.1.1). We use *italics* to write Maude rewrite rules and *CamelCase* for variable or sort names. Some objects of interest include the singleton objects *kernel*, *ui*, and *display* (in classes *Kernel*, *UI*, and *Display*, respectively). We also have non-singleton classes *WebApp* and *NetProc* representing web apps and network processes in IBOS, respectively.

As a further aid to the reader and a complement to the graphical overview of IBOS in Figure 6.1, we rewrite this graphical figure using our formal specification. Said another way, we provide in Figure 6.2 a representative state (e.g., a ground term) of the transition system $(C_{\Sigma/E\cup B,State}, \rightarrow_{\mathcal{R}})$ where $\mathcal{R}$ is the Maude rewrite theory specifying IBOS. To improve readability, we write each object attribute on a separate line.

Let us make a few high-level remarks about the figure. In our specification, urls are encoded as numbers wrapped by constructor *url*(...). In IBOS, to enforce SOP as well as other security policies, both browser frames and network connections must be tracked. Each browser frame is represented by an object of class *WebApp*, while each network connection is represented by an object of class *NetProc*. The *kernel* manages process state by internal metadata tables *webAppInfo* where each web app is tagged by its origin and *netProcInfo* where each network connection is tagged by its origin web app and destination server. When a new web app is created, the kernel automatically creates a corresponding network process between the webapp and its origin server. Since the *kernel* is responsible for creating network connections, it records the next fresh network process identifier in *nextNetProc*.

The *kernel*'s *secPolicy* attribute stores its security policy. Each policy consists of a sender, a receiver, and a message type. Any message not explicitly allowed by the policy is dropped. In the policy, the special *Oid*s *network* and *webapp* represent a policy allowing any network process or webapp to send a particular kind of message to its corresponding webapp or network process, respectively. In the figure, the *kernel* is preparing to forward a message from *webapp*(*0*) to its corresponding network process *network*(*0*) asking to fetch an item from the web app's origin *url*(*15*).

Aside from the *kernel*, the system has a few other distinguished objects. The *display* object tracks the content currently shown on the screen in *displayContent*; to do that, it should know which web app is the *activeTab*. The *ui* (user interface) contains the list of commands given by the user during some usage session in its *toKernel* attribute. The *webappmgr* (web app manager) is responsible for spawning new web apps; in our model, it just records the next fresh web app identifier in attribute *nextWebApp*. Finally, the *nic* (network interface card) has two attributes for ingoing and outgoing data. In our model, we identify urls and their loaded content. To model network latency, outgoing messages in *nic-out* are queued up in *nic-in* in a random order.

Of course, we also have objects representing web apps and network connections. In the figure, *webapp*(*0*)'s *rendered* content is blank. However, it is currently *loading* its content from its origin *url(15)*; its request to fetch data from its corresponding network process is currently being handled by the kernel. Currently, its *toKernel* and *fromKernel* IPC message queues are blank because it is not performing any other communication at this time. There is also a corresponding network process *network(0)*; it has two direct-memory access (DMA) buffers *mem-in* and *mem-out* that are used as I/O channels between itself and the network card driver. Like web apps, network processes also have two IPC message queues. Finally, a network process also stores which web app its received data should be sent back to.

As a second example, let us consider the specification of the *change-display* rewrite rule shown in Figure 6.3. This rule involves the *display* object and the *WebApp* designated as the display's *activeTab*. In our model, the *rendered* attribute of a web app represents its current rendering of the HTML document located at its origin. When the web app is first created, its *rendered* attribute has the value *about-blank*, i.e., nothing has yet been rendered. Thus, this rule essentially states that, at any time, the displayed content can be replaced by currently rendered HTML document of the active tab, *only if* it is different from the currently displayed content.

Our IBOS browser specification contains 23 rewrite rules and is about 850 lines of Maude code; it is available at http://maude.cs.illinois.edu/tools/ibos.

### 6.2.2 IBOS Security Properties Specification

We first describe at a high level the security properties that we will formally specify and verify. The key property that we verify is the *same-origin policy* (SOP), but we also specify and verify the *address bar correctness* (ABC) property. Our discussion follows that of [116, 115], based on invariants $P_1$-$P_{11}$:

$\mathrm{P}_1$ The kernel must route network requests from web page instances to the proper network process.

$\mathrm{P}_2$ The kernel must route Ethernet frames from the network interface card (NIC) to the proper network process.

```
< kernel | addrBar(url(15)),
           handling(msg(webapp(0), network(0), FETCH-URL, url(15)))
           nextNetProc(1),
           webAppInfo(pi(webapp(0), url(15))),
           netProcInfo(pi(network(0), url(15), url(15))),
           secPolicy(policy(webapp, network, FETCH-URL),
              policy(network, webapp, RETURN-URL),
              policy(ui, webapp, NEW-URL),
              policy(ui, webapp, SWITCH-TAB)) >
< display | displayContent(about-blank),
            activeTab(webapp(0)) >
< ui | toKernel(msg(ui, webapp, NEW-URL, url(25))) >
< webappmgr | nextWebApp(1) >
< nic | nic-in(emtpy),
        nic-out(empty) >
< webapp(0) | URL(url(15)),
              rendered(about-blank),
              loading(true),
              toKernel(empty),
              fromKernel(empty) >
< network(0) | mem-in(empty),
               mem-out(empty),
               returnTo(webapp(0)),
               toKernel(empty),
               fromKernel(empty) >
```

Figure 6.2: Representative IBOS System State

$$
\begin{aligned}
&\big\{\langle \mathit{display} \mid \mathit{displayContent}(D), \mathit{activeTab}(\mathit{WA}), \mathit{Atts}\rangle\, \langle \mathit{WA} \mid \mathit{rendered}(D'), \mathit{Atts}'\rangle\ \mathit{Conf}\,\big\}\\
\rightarrow\ &\big\{\langle \mathit{display} \mid \mathit{displayContent}(D'), \mathit{activeTab}(\mathit{WA}), \mathit{Atts}\rangle\, \langle \mathit{WA} \mid \mathit{rendered}(D'), \mathit{Atts}'\rangle\ \mathit{Conf}\,\big\}\\
\text{if}\ \ &D \neq D'
\end{aligned}
$$

Figure 6.3: *change-display* rule

$\mathrm{P}_3$ Ethernet frames from network processes to the NIC must have an IP address and TCP port that matches the origin of the network process.

$\mathrm{P}_4$ HTTP data from network processes to web page instances must be from acceptable origins.

$\mathrm{P}_5$ Network processes for different web page instances must remain isolated.

$\mathrm{P}_6$ The browser chrome and web page content displays are isolated.

$\mathrm{P}_7$ Only the current tab can access the screen, mouse, and keyboard.

$\mathrm{P}_8$ All components can only perform their designated functions.

$\mathrm{P}_9$ The URL of the active tab is displayed to the user.

$\mathrm{P}_{10}$ The displayed web page content is from the URL shown in the address bar.

$\mathrm{P}_{11}$ All configurations are well-formed, i.e., non-duplication of *Oid*s and *Att*s.

We define same-origin policy as SOP $= \bigwedge_{1\leqslant i \leqslant 7} P_i$; address-bar correctness is specified as ABC $= P_{10}$. Note that $P_9 \wedge P_{10} \Rightarrow P_7$. Since $P_5$, $P_6$, and $P_8$ follow directly from the model design, it is sufficient to prove $\bigwedge_{i\in I} P_i$ for $I = \{1, 2, 3, 4, 9, 10\}$. Invariant $P_{11}$ is new to our current formalization, but is implicitly used in prior work; it forbids absurd configurations, e.g., having two *kernel*s or a *WebApp* that has two *URL*s (see Section 6.1.1). Due to its fundamental relation to how object-based rewrite theories are defined, we need $P_{11}$ in the proof of *all* other invariants. As an example of how these invariant properties can be formalized in our model as constrained pattern predicates, we show how the ABC invariant can be specified in our system below:

$$\begin{aligned} &\{\langle \mathit{kernel} \mid \mathit{addrBar}(U), \mathit{Atts}\rangle \\ &\qquad\qquad\qquad\qquad \langle \mathit{display} \mid \mathit{displayContent}(U'), \mathit{Atts}'\rangle\ \mathit{Conf}\} \mid U' \trianglelefteq U \end{aligned} \tag{6.1}$$

where $U' \trianglelefteq U \Leftrightarrow_{\text{def}} (U' \neq \mathit{about\text{-}blank} \Rightarrow U = U')$, i.e., the *display* is either blank *or* its contents' origin matches the address bar. Note that, for simplicity, in our model, we identify displayed content with its origin URL. As a second example, consider how to formalize invariant $P_9$:

$$\begin{aligned} &\big\{\langle \mathit{kernel} \mid \mathit{addrBar}(U), \mathit{Atts}\rangle \\ &\qquad\qquad\qquad \langle \mathit{display} \mid \mathit{activeTab}(\mathit{WA}), \mathit{Atts}'\rangle \\ &\qquad\qquad\qquad\qquad\qquad \langle\, \mathit{WA} \mid \mathit{URL}(U''), \mathit{Atts}''\rangle\ \mathit{Conf}\,\big\} \mid U \trianglelefteq U'' \end{aligned} \tag{6.2}$$

i.e., the address bar must match the URL of the active tab, unless the address shown is *about-blank*, i.e., nothing is shown. Finally, $P_{11}$ has a trivial encoding: $\{\mathit{Conf}\} \mid \mathit{WF}(\mathit{Conf})$, where $\mathit{WF} : \mathit{Conf} \to \mathit{Bool}$ is the well-formedness predicate. Our specification has 200 lines of Maude code to specify the pattern predicates used in our invariants and another 900 lines of code specifying all of the auxiliary functions and predicates. As stated in the Introduction, we additionally must prove that (a) the IBOS system specification extended by our security property specification is *suitable* for symbolic reachability analysis; (b) our ABC and SOP invariants are at least as strong as the corresponding invariants in prior work [116, 115] $\mathrm{ABC}_0$ and

$\mathrm{SOP}_0$. Proof obligation (a) can be met by using techniques for proving *ground convergence* and *sufficient completeness* of conditional equational theories while proof obligation (b) is typicially demonstrated by proving an implication, e.g., SOP $\Rightarrow$ $\mathrm{SOP}_0$. We have carried out full proofs of (a) and (b); see more details in Appendix E. Note that, given (a) holds, our theory trivially meets all of the suitability requirements for reachability analysis given in Definition 5.2. Finally, see http://maude.cs.illinois.edu/tools/ibos/ for a complete listing of the invariant patterns SOP and ABC.

## 6.3 PROOF OF IBOS SECURITY PROPERTIES

In this section, due to space limitations, we give a high-level overview of our proof methodology for verifying the SOP and ABC properties for IBOS, and show a subproof used in deductively verifying ABC. Each proof script has roughly 200 lines of code and another 20 to specify the reachability logic sequent being proved. The full script is available at http://maude.cs.illinois.edu/tools/ibos/.

**Modular Proof Methodology.** In this subsection we survey our modular proof methodology for proving invariants using reachability logic and comment on how we exploit modularity in three key ways. By "proof methodology", we mean the common principles underlying how to structure and efficiently carry out formal proofs. By "modularity", we mean the ability to compose complex specifications from simple and independent reusable pieces. Therefore, modularity is a key principle to scale up formal proofs beyond toy examples.

Most of the IBOS proof effort was spent strengthening an invariant $I$ into an inductive invariant $I_{ind}$, where $I$ is either SOP or ABC. Typically, $I_{ind}$ is obtained by iteratively applying the proof strategy given in Section 6.1.2. In each round, assume candidate $I'$ is inductive and attempt to complete the proof. If, after applying the Step rule (and possibly some derived rules), an application of Axiom is impossible, examine the proof of failed pattern subsumption $[\![A]\!] \subseteq [\![I']\!]$ in the side-condition of Axiom. If pattern formula $C$ (possibly using new functions and predicates defined in theory $\Delta$) can be found that might enable the subsumption proof to succeed, try again with candidate $I' \wedge C$. In parallel, our system specification $\mathcal{R}$ is enriched by extending the underlying *convergent* equational theory $\mathcal{E}$ to $\mathcal{E} \cup \Delta$ to obtain the enriched rewrite theory $\mathcal{R}_{\Delta}$.

The first kind of modularity we exploit is *rule modularity.* Recall that any reachability logic proof begins with an application of the Step rule. Since Step must consider the result of symbolically rewriting the initial sequent with all possible rewrite rules, we can equivalently construct our proof on a "rule-by-rule" basis, i.e., if $\mathcal{R} = (\Sigma, E \cup B, \{l_j \to r_j \text{ if } \phi_j\}_{j \in J})$, we can consider $|J|$ separate reachability proofs using respective theories $\mathcal{R}_j = (\Sigma, E \cup B, \{l_j \to r_j \text{ if } \phi_j\})$ for $j \in J$. Thus, we can focus on strengthening invariant $I'$ for each rule $j \in J$ separately.

Another kind of modularity that we can exploit is *subclass modularity.* Because we are reasoning in an object-based rewrite theory, each rule mentions one or more objects in one or more classes, and describes how they evolve. Recall from Sect. 6.1.1 that subclasses must contain all of the attributes of their superclass, but may define additional attributes and have additional rewrite rules which affect them. The upshot of all this is that if we *refine* our specification by instantiating objects in one class into some subclass, any invariants proved for the original rules immediately hold for the same rules in the refined specification. Because of rule modularity, we need only prove our invariants hold for the newly defined rules. Furthermore, among those newly defined rules, all *non-interfering* rules trivially satisfy any already proved invariants, where non-interfering rules do not directly or indirectly influence the state of previously defined attributes.

Lastly, we exploit what we call *structural modularity.* Since our logic is constructor-based and we assume that $B_\Omega$-matching is decidable, by pattern matching we can easily specify a set $S$ of sequents to which we can apply the *same* combination of derived proof rules. This is based on the intuition that syntactically similar goals typically can be proved in a similar way. More concretely, given a set of reachability formulas $S$ and pattern $p \in T_\Omega(X)$, we can define the subset of formulas $S_p = \{(u \mid \varphi) \to^{\circledast} B \in S \mid \exists \alpha \in [X \to T_\Omega(X)]\, p\alpha =_{B_\Omega} u\}$. Although the simplified example below does not illustrate structural modularity, we have heavily exploited this principle in our formal verification of IBOS.

**Address Bar Correctness Proof Example.** Here we show a snippet of the ABC invariant verification, namely, we prove that the invariant holds for the *change-display* rule by exploiting rule modularity as noted above. As mentioned in our description of invariant $P_{11}$, well-formedness is required for all invariants. Therefore, we begin with ABC $\wedge$ $P_{11}$ as our candidate inductive invariant, which, as mentioned in Sect. 2, normalizes by disjoint $B_\Omega$-unification to the invariant:

$$\big\{\langle \mathit{kernel} \mid \mathit{addrBar}(U), \mathit{Atts}\rangle \\ \langle \mathit{display} \mid \mathit{displayContent}(U'), \mathit{Atts}'\rangle\ \mathit{Conf}\,\big\} \mid U' \trianglelefteq U \wedge \mathit{WF}(\ldots) \quad (6.3)$$

where for brevity ... expands to the entire term of sort *Conf* wrapped inside operator $\{\_\}$, i.e., the entire configuration is *well-formed.* Recall the definition of the *change-display* rule in Section 6.2.1. We can see that our invariant only mentions the *kernel* and *display* processes, whereas in rule *change-display* the value of *displayContent* depends on the *rendered* attribute of a *WebApp*, i.e., the one selected as the *activeTab*. Clearly, our invariant seems too weak. How can we strengthen it? The reader may recall $P_9$, which states "the URL of the active tab is displayed to the user." Thus, by further disjoint $B_\Omega$-unification, the strengthened invariant ABC $\wedge$ $P_{11}$ $\wedge$ $P_9$ normalizes to:

$$\big\{\langle \mathit{kernel} \mid \mathit{addrBar}(U), \mathit{Atts}\rangle \langle \mathit{display} \mid \mathit{displayContent}(U'), \mathit{activeTab}(\mathit{WA}), \mathit{Atts}'\rangle \\ \langle \mathit{WA} \mid \mathit{URL}(U''), \mathit{Atts}''\rangle\ \mathit{Conf}\,\big\} \mid U' \trianglelefteq U \wedge U \trianglelefteq U'' \wedge \mathit{WF}(\ldots) \quad (6.4)$$

This new invariant is closer to what we need, since the pattern now mentions the particular web app we want. Unfortunately, since our invariant still knows nothing about the *rendered* attribute, at least one further strengthening is needed.

At this point, we can enrich our theory with a new predicate stating that the *rendered* and *URL* attributes of any *WebApp* always agree[1]. Let us declare it as $R : \mathit{Conf} \to \mathit{Bool}.$ We can define it inductively over configurations by:

$$R(\langle \mathit{WA} \mid \mathit{rendered}(U), \mathit{URL}(U'), \mathit{Atts}\rangle\ \mathit{Conf}) = U \trianglelefteq U' \wedge R(\mathit{Conf}) \quad (6.5)$$

$$R(\langle P \mid \mathit{Atts}\rangle\ \mathit{Conf}) = R(\mathit{Conf})\ \mathit{if}\ \neg \mathit{WA}(P) \quad (6.6)$$

$$R(\mathit{none}) = \top \quad (6.7)$$

[1] Note that, in a very real sense, this requirement is at the heart of the SOP, since it means that any *WebApp* has indeed obtained content from its claimed origin.

where $WA : Oid \rightarrow Bool$ ambiguously denotes a predicate that holds iff an *Oid* refers to a web app. Intuitively, it says that whatever a *WebApp* has *rendered* is either blank or has been loaded from its *URL*. The strengthened invariant becomes:

$$\begin{aligned}\{\langle \mathit{kernel} \mid \mathit{addrBar}(U), \mathit{Atts}\rangle\, \langle \mathit{display} \mid \mathit{displayContent}(U'), \mathit{activeTab}(WA), \mathit{Atts}'\rangle \\ \langle WA \mid \mathit{URL}(U''), \mathit{Atts}''\rangle\ \mathit{Conf}\,\} \mid \\ U' \trianglelefteq U \wedge U \trianglelefteq U'' \wedge WF(\ldots) \wedge R(\ldots)\end{aligned} \tag{6.8}$$

where, as before, the ... represents the entire term of sort *Conf* enclosed in $\{\_\}$. Now, all of the required relationships between variables in the rewrite rule seem to be accounted for. Uneventfully, with the strengthened invariant $ABC \wedge P_{11} \wedge P_9 \wedge R$, the subproof for the *change-display* rule now succeeds.

## 6.4 RELATED WORK AND CONCLUSIONS

**Related Work on IBOS Verification**. In this chapter, we have presented the first fully formalized deductive verification of SOP and ABC for IBOS. Note that in [116, 115], SOP and ABC were also verified. Their approach consisted of a hand-written proof that any invariant counter-example must appear on some trace of maximum length $n$ plus bounded model checking showing that such counterexamples are unreachable on all traces of length $n$. In [110], an attempt was made to deductively verify these same invariants via the Maude invariant analyzer. In that work, a few basic invariants were proved, but unfortunately, due to the generation of thousands of proof obligations, not including any property listed in Sect. 6.2.2. Compared to that previous work, this chapter presents the first full deductive verification of the IBOS security properties.

**Related Work on Browser Security**. In terms of computer technology, the same-origin policy is quite old: it was first referenced in Netscape Navigator 2 released in 1996 [124]. As [121] points out, different browser vendors have different notions of SOP. Here, we situate IBOS and our work into this larger context. Many papers have been written on policies for enforcing SOP with regards to frames [125], third-party scripts [126, 127], cached content [128], CSS [129], and mobile OSes [130]. Typically, these discussions assume that browser code is working as intended and then show existing policies are insufficient. Instead, IBOS attacks the problem taking a different tack: even if the browser itself is compromised, can SOP still be ensured? What the IBOS verification demonstrates is that —by using a minimal trusted computing base in the kernel, implementing separate web frames as separate processes, and requiring all IPC to be kernel-checked— one can in fact enforce the standard SOP notions, *even if* the complex browser code for rendering HTML or executing JavaScript is *compromised*. Although our model does not treat JavaScript, HTML, or cookies, explicitly, since it models system calls which are used for process creation, network access, and inter-process communication, code execution and resource references can be abstracted away into the communication primitives they ultimately cause to be invoked, allowing us to perform strong verification in a high-level fashion. [131] surveys many promising lines of research in the formal methods web security landscape. Prior work on formal and declarative models of web browsers includes [132] as well as the *executable* models [133, 134]. Our work complements the Quark browser design and implementation of [135]: Quark, like IBOS, has a small trusted kernel (specified in Coq). In addition to proving tab non-interference and address bar correctness theorems similar to our own, the authors use Coq code extraction to produce a verified, functional browser. Unlike Quark, whose TCB includes the underlying OS (e.g., the linux kernel) and Coq code extraction and compilation tools, the TCB of the IBOS browser consists of only 42K lines of C/C++ code.

**Related Work on Reachability Logic**. Our work on *constructor-based* reachability logic [100] builds upon previous work on reachability logic [90, 99, 26, 27] as a language-generic approach to program verification, parametric on the operational semantics of a programming language. Our work extends in a non-trivial manner reachability logic from a programming-language-generic logic of programs to a rewrite-theory-generic logic to reason about *both* distributed system designs in Maude, and imperative programs based on their rewriting logic semantics. Our work in [100] was also inspired by the work in [29], which for the first time consider reachability logic for rewrite theories, but went beyond [29] in several ways, including more expressive input theories and state predicates, and a simple inference system as opposed to an algorithm. Also related to our work in [100], but focusing on *coinductive reasoning*, we have the recent work in [102, 103, 104], of which, in spite of various substantial differences, the closest to our work regarding the models assumed, the kinds of reachability properties proved, and the state predicates and inference systems proposed is the work in [104].

**Conclusions and Future Work**. We have presented a full deductive proof of the SOP and ABC properties of the IBOS browser design, as well as a prover and a modular reachability logic verification methodology making proofs scalable to substantial proof efforts like that of IBOS. Besides offering a case study that can help other distributed system verification efforts, this work should also be seen as a useful step towards incorporating the IBOS design ideas into future fully verified browsers. The web is alive and evolving; HTML and JavaScript standards as well as web browser designs must adapt to meet business and consumer requirements. These evolutions necessitate that formal approaches adapt and scale to increasingly rich applications and connected environments. Looking towards the future of IBOS, two goals stand out: (i) extending the design of IBOS to handle some recent extensions of the SOP, e.g., cross-origin resource sharing (CORS) to analyze potential cross-site scripting (XSS) and cross-site request forgery attacks (XSRF) [136], and to check for incompatible content security policies (CSP) [137] in relation to SOP; by exploiting subclass and rule modularity, the verification of an IBOS extension with such new functionality could reuse most of the current IBOS proofs, since extra proofs would only be needed for the new, functionality-adding rules; and (ii) exploiting the intrinsic concurrency of Maude rewrite theories to transform them into correct-by-construction, deployable Maude-based distributed system implementations, thus closing the gap between verified designs and correct implementations. Our work on IBOS takes one more step towards demonstrating that a formally secure web is possible in a connected world where security is needed more than ever before.

# CHAPTER 7 CONCLUSIONS AND FUTURE WORK

From pattern operations to extensible satisfiability methods to reachability logic, we have seen increasingly powerful and sophisticated rewriting logic-based symbolic methods and tools for reasoning about distributed system specifications in rewriting logic. We believe that these developments detailed above provide further evidence that rewriting logic is particularly well-suited for specifying and analyzing distributed systems. The skeptic may claim that the methods and tools described above are mere theoretical constructs without practical backing. To that objection, we note that all of the (meta-)algorithms sketched in the pages above have already been implemented using the Maude rewrite engine and have already been used in various case studies [80, 35, 36, 25, 88, 100], in addition to the as-of-yet unpublished IBOS case study in this thesis. Given the *reflective* nature of rewriting logic, these (meta-)algorithms can all be implemented in rewriting logic itself (this also has the advantage of allowing quite simple correctness proofs!).

As we stated in the Introduction, we further believe that combination of rewriting logic and Maude as an executable specification logic and interpreter is a good candidate to address the various challenges faced by the practitioner designing real-world distributed systems. In particular, the techniques described in this thesis have several important qualities that let us scale up to complex problems:

1. the methods presented provide *theory-generic* and *rewrite-theory-generic* reasoning capabilities;

2. in particular, our variant-satisfiability framework makes SMT solving *decidable* for a *broad* and *parametric* class of theories;

3. constructor patterns and constrained constructor pattern predicates provide a powerful method to capture state predicates where techniques like *pattern operations*, *B-matching*, *B-unification*, and *rewriting modulo B* can be aggressively exploited for state simplification;

4. constructor-based reachability logic parameterized over a rewrite theory provides an elegant solution to deductively verify *reachability claims* as well as *inductive invariants* in an *initial reachability model*;

5. for complex reachability logic proofs, we can easily exploit modularity at every level of specification—at the system specification level when designing rewrite theories, at the property specification level when defining equational theories, and at the proof level when constructing proofs of reachability claims.

This thesis is the culmination of several years worth of effort. Since the various chapters build on each other, it has the pleasing effect that different ideas can "cross-pollinate" and provides an easy path to stress test our algorithms. This cross-pollination has resulted in several improvements to our ideas:

1. our variant satisfiability algorithm has been optimized for the common case where the set of defined symbols has no subsort-overloadings that are constructors;

2. we have developed powerful *theory-generic* heuristics to extend our satisfiability techniques to *undecidable* theories;

3. our capstone project, the verification of security properties of the the Illinois Browser Operating System [116] led to many improvements in the user-interface of the reachability logic tool as well as new proof methodologies.

Nevertheless, there is of course more work to be done. The future work can be subdivided into two categories: tooling improvements and theory extensions. Let us discuss each in turn.

In terms of tooling improvements, several attractive possibilities include:

1. discharging more trivial proof obligations automatically, e.g. that a set of terms covers a sort for the Case rule;
2. deeper *integration* of the various tools in the Maude ecosystem, enabling workflows where, e.g. the reachability logic tool can invoke an inductive theorem prover if needed to discharge proof obligations that it could not automatically prove;
3. extending our reachability logic tool so that it can generate *proof objects*, providing a much easier path to both proof visualization and certification.

In terms of theoretical advancements, we would like to explore:

1. constructor-based reachability logic meta-theorems that describe the class of *liveness properties* that can be expressed and verified as well as expressing the required *fairness assumptions*;
2. advanced techniques for reasoning about concurrent, imperative programming languages in our reachability logic framework;
3. additional heuristics for handling undecidable theories that integrate well with our existing techniques;
4. lifting our reachability logic framework to reason over parameterized theories.

To summarize, what has been achieved in this thesis is:

1. the development of several new symbolic reasoning techniques such as:
   (a) order-sorted pattern operations,
   (b) meta-level algorithms for variant satisfiability, and
   (c) constructor-based reachability logic; and
2. the first mechanized, general, deductive framework for verifying reachability claims in rewriting logic, including:
   (a) the tool implementation;
   (b) its modular proof methodology, and
   (c) case studies demonstrating the tool's effectiveness culminating in the IBOS case study.

The implications and potential applications are as varied as the kinds of rewrite theories one can imagine. Aside from obvious system modeling applications for both academic and industrial researchers, we also envision wide-ranging uses for education. Since rewriting logic embeds many other semantic frameworks and logics of interest, solving reachability claims over generic rewrite theories provides a powerful and general framework to explore these systems as well as designing new ones.

# APPENDIX A REACHABILITY LOGIC TOOL COMMAND GRAMMAR

Here we provide a BNF grammar of the commands which can be given as inputs to our prototype Maude tool. In the grammar, **boldface words** represent themselves (i.e. terminals) while ⟨words in angle brackets⟩ represent non-terminals. A nonterminal surrounded by square brackets, e.g., [⟨number⟩], represents an optional argument. BNF grammar alternatives are separated by vertical bars (|). Whenever we use a reserved symbol as a terminal, we surround it in double quotes, e.g. "|". The horizontal lines delimit the three basic categories of commands: (i) proof setup, (ii) adding invariants and adding goals, and (iii) applying proof steps or simple proof strategies.

| | | |
|---|---|---|
| ⟨outer-cmd⟩ | ::= | **(** ⟨inner-cmd⟩ **. )** |
| ⟨inner-cmd⟩ | ::= | **select** ⟨module-name⟩ |
| | \| | **declare-vars** ⟨var-set⟩ |
| | \| | **def-term-set** ⟨pattern-form⟩ |
| | \| | **use tool** ⟨tool-name⟩ **for validity on** ⟨module-name⟩ |
| | \| | **start-proof** |
| | \| | **inv** ⟨goalname⟩ **to** ⟨op-id⟩ [**with** ⟨var-set⟩] **on** ⟨pattern-form⟩ |
| | \| | **add-goal** ⟨goalname⟩ **:** ⟨reach-form⟩ |
| | \| | **auto** [⟨number⟩] |
| | \| | **auto*** |
| | \| | **list-goals** |
| | \| | **focus** ⟨goal-id⟩ |
| | \| | **case** ⟨goal-id⟩ **on** ⟨var-name⟩ **using** ⟨term-set⟩ |
| | \| | **split** ⟨goal-id⟩ **by** ⟨eqform⟩ [**and** ⟨eqform⟩] |
| | \| | **replace** ⟨goal-id⟩ **by** ⟨eqform⟩ |
| | \| | **subsume** ⟨goal-id⟩ **by** ⟨goal-id⟩ |
| | \| | **on** ⟨goal-id⟩ **use strat** ⟨goal-name-set⟩ |

Figure A.1: Reachability Tool Command Grammar

Category (i) commands let the user select a module defining a rewrite relation we wish to reason over, initialize a tool backend, to declare variables which can be used in commands of type (i) and (iii), and to start/stop proofs. The commands in category (ii) are also straightforward: **inv** takes a bracket operator-id ([]), a set of shared variables $V$, and a pattern form $P$ and adds a goal to be solved of the form $P\sigma \rightarrow^{\circledast} [P]$ where $\sigma(v) = v \Leftrightarrow v \in V$; we can also use the lower-level **add-goal** command to add goals directly, i.e. reachability formulas. Finally, type (iii) commands let the user apply the default strategy using the **auto** command for one or more steps as well as applying the case analysis derived rule using **case** and split derived rules using **split** and **replace**. Focusing on a goal eliminates all other goals from the proof state; obviously, this is unsound. The intent, however, is not to continue the proof process, but to *restart it* after such focusing. The **focus** command enables the user to focus attention on some proof goals that seem to lead to looping so that, for example, the proof can be restarted with some additional lemmas (e.g., some strengthened invariants) to help its completion, or some bug in the original set of goals may be detected. The **use strat** command allows the user to select the set of axioms that will be tried when applying the Axiom rule to the specified goal as well as any of its descendants.

The grammar below defines the syntactic categories used by tool commands. Some non-terminals are marked as special. These non-terminals are handled by built-in parsers as part of the Maude runtime.

```
⟨reach-form⟩        ::=   ⟨pattern-form⟩ => ⟨pattern-form⟩
⟨pattern-form⟩      ::=   ⟨pattern-form⟩ \/ ⟨pattern-form⟩
                      |   ⟨term⟩ “|” ⟨eqform⟩
⟨eqform⟩            ::=   ⟨eqform⟩ \/ ⟨eqform⟩ | ⟨eqform⟩ /\ ⟨eqform⟩
                      |   ⟨term⟩ = ⟨term⟩ | ⟨term⟩ =/= ⟨term⟩
⟨term-set⟩          ::=   ( ⟨term⟩ ) ⟨term-set⟩ | ( ⟨term⟩ )
⟨var-set⟩           ::=   ( ⟨var-name⟩ ) ⟨var-set⟩ | ( ⟨var-name⟩ )
⟨goal-name-set⟩     ::=   ⟨goal-name⟩ ⟨goal-name-set⟩ | ⟨goal-name⟩
⟨goal-id⟩           ::=   ⟨nat⟩
⟨goal-name⟩         ::=   special
⟨op-id⟩             ::=   special
⟨module-name⟩       ::=   special
⟨tool-name⟩         ::=   special
⟨var-name⟩          ::=   special
⟨term⟩              ::=   special
⟨nat⟩               ::=   special
```

Figure A.2: Reachability Logic Tool Command Grammar Syntactic Categories

# APPENDIX B AUXILIARY PROOF TECHNIQUES

## B.1 INTRODUCTION

Throughout this thesis, we have been building a theoretical framework for reasoning about constrained terms and reachability between them. Recall that reachability logic reasoning requires reasoning at two levels simultaneously: (i) reasoning about reachable states using narrowing over some rewrite theory $\mathcal{R}$; and (ii) reasoning about satisfiablity and validity of accumulated constraints in $\mathcal{R}$'s underlying equational theory $\mathcal{E}$. In Chapters 4 and 5, we present systems of type (ii) and (i) respectively. This led to our capstone project in Chapter 6, specifying the same-origin policy invariant for IBOS and then verifying that it holds using reachability logic. The point that we wish to emphasize here is that systems of type (i) crucially depend on ever more powerful and efficient systems of type (ii) in order to scale up to more complex theories. Our IBOS case study is no exception to this rule. In fact, due to the complexity of this case study, we will need additional type (ii) equational reasoning techniques not previously discussed. The purpose of this appendix is to enrich our theoretical toolkit developed thus far with all these additional techniques, enabling us to tie up all loose ends and nail down all proofs, especially those in Appendix E.

**Rewriting and Induction**. When it comes to equational reasoning, before we jump to advanced techniques, we have to start with the basics. In particular, reasoning in an arbitrary conditional equational theory $\mathcal{E} = (\Sigma, E \cup B)$ is typically hopeless whenever the equivalence classes involved are large. Instead, we can mechanize equational deduction via rewriting using a rewrite theory $\mathcal{R} = (\Sigma, B, \vec{E})$, where the equations $E$ are *oriented* as rewrite rules. That such a theory is *convergent* (which intuitively can be understood as the combination of both confluence and termination modulo $B$) implies rewriting with $\vec{E}$ modulo $B$ is a sound and complete method for checking $E \cup B$-equality. Since mature rewrite engines (e.g. Maude, CafeOBJ, etc.) exist, what is needed is a method for checking whether a theory $\mathcal{R}$ really is convergent, or at least ground convergent, i.e., convergent for ground terms.

On top of that, to reasonably support induction, we additionally require that our theory $(\Sigma, E \cup B)$ is *sufficiently complete* with respect to (or *protects*) a constructor subtheory $(\Omega, B_\Omega) \subseteq (\Sigma, E \cup B)$. This essentially amounts to three facts: that $\Sigma$ and $\Omega$ have the same sort poset, $B_\Omega \subseteq B$, and $T_{\Sigma/E \cup B}|_\Omega \cong T_{\Omega/B_\Omega}$. Sufficient completeness allows us to perform induction by doing case analysis on constructors alone. Typically, the language of constructors is *much simpler* than the full theory, leading to huge gains in both efficiency and clarity. As is the case above, doing case analysis by constructors is a relatively simple affair; the difficulty lies in finding methods for checking sufficient completeness of a theory with respect to a constructor subtheory.

Given that rewriting-based techniques and constructor induction have been widely known and used for decades, one might assume that these matters of checking convergence (or at least ground convergence) and sufficient completeness are totally solved. In actuality, scaling up existing techniques to complex equational theories, especially when some equations in $E$ are conditional, is a highly non-trivial matter. In practice, it seems people either *give up* on proving convergence or sufficient completeness at all or else rely on *one-shot, ad-hoc* proof techniques. Ideally, we would like a principled approach to proving convergence and sufficient completeness, that is, both (a) sufficiently general, so that it can be applied to complex theories of interest; and (b) respects modularity, so that the proof effort scales linearly with theory size. In this appendix we provide a partial answer to this question by: (i) recalling existing techniques for proving termination using recursive path orders (RPO); (ii) presenting a proof system for proving ground convergence and sufficient completeness *hierarchically.*

**Variant Unification**. A powerful technique for equational reasoning involves exploiting the finite variant property (or FVP, see Definition 2.13). When a theory is FVP, all well-formed terms in that theory have a finite number of most general variants.When a theory is FVP and sufficiently complete with respect to a subtheory $(\Omega, B_\Omega)$ with decidable unification, we can immediately decide unifiability by the *variant unification* algorithm. In addition to theories, by abuse of notation, we say that a term is FVP whenever it has a finite number of most general variants. In a similar fashion, we say a conjunction of equalities $\phi$ is *variant solvable* whenever a complete set of unifiers for $\phi$ may be computed by variant unification. Clearly, a conjunction of equalities $\phi$ in an FVP theory that protects a subtheory with decidable unification is variant solvable. A surprising and interesting fact is that $\phi$ *may* be variant solvable even when it *only* has an FVP subtheory. In this appendix, we describe conditions (which can often be discharged automatically) under which variant unification of an equality in an FVP subtheory is complete with respect to the entire theory, so that such conjunctions of such equalities are *variant solvable in an FVP subtheory.*

**Contextual Rewriting**. The final technique that we will address in this appendix is *contextual rewriting*, i.e. contextual rewriting in the intial model of an equational theory $(\Sigma, E \cup B)$. This technique applies mainly when doing validity checking in the quantifier-free fragment of equational logic. In that case, we can always reduce formulas to conjunctive normal form (CNF) so that our formula has the form:

$$(\Sigma, E \cup B) \models \bigwedge_i \left[ \left( \bigvee_k u_{i,k} \neq v_{i,k} \right) \vee \left( \bigvee_j u_{i,j} = v_{i,j} \right) \right] \tag{B.1}$$

or equivalently:

$$(\Sigma, E \cup B) \models \bigwedge_i \left[ \left( \bigwedge_k u_{i,k} = v_{i,k} \right) \Rightarrow \left( \bigvee_j u_{i,j} = v_{i,j} \right) \right] \tag{B.2}$$

This second form suggests a possible reasoning technique, that is, to apply the theorem of constants and deduction theorem to transform our problem into the equivalent problem:

$$\bigwedge_i \left[ (\Sigma \uplus V_i, E \cup B \cup \{\overline{u}_{i,k} = \overline{v}_{i,k}\}_k) \models \left( \bigvee_j \overline{u}_{i,j} = \overline{v}_{i,j} \right) \right] \tag{B.3}$$

where $V_i = \mathit{vars}(\{\overline{u}_{i,k} = \overline{v}_{i,k}\}_k)$ and $\overline{t}_i$ is identical to $t_i$ except with all variables in $V_i$ are replaced by corresponding fresh constants. This is widely known to be sound for equational reasoning [32]. The complication arises when we want to mechanize reasoning in the combined theory $(\Sigma \uplus V_i, E \cup \{\overline{u}_{i,k} = \overline{v}_{i,k}\}_k)$ by *rewriting*. In particular, the combined theory is often *non-confluent* and *non-terminating*! This leaves us with two options: we either give up and try to make do without confluence and termination, or more ideally, we *recover* analogues of these properties under additional assumptions. In this appendix we describe how equipping our input theory $(\Sigma, E \cup B)$ with a $B$-compatible and total recursive path ordering (RPO) enables us to at least recover termination of the combined theory.

**Auxiliary Proof Techniques Overview**. In summary, in this appendix we will cover the following techniques:

1. proving termination via axiom-compatible RPOs (recursive path orderings);

2. *hierarchically* proving ground convergence and sufficient-completeness;

3. giving conditions for provable completeness of variant-unification in an FVP *subtheory*;
4. using *contextual rewriting* for validity checking.

Since these techniques represent very recent research and are not central to our core advances in the thesis, a full proof of soundness of the techniques is left for papers in preparation.

Finally, before we survey these additional techniques, we first limit the scope of the kinds of theories we consider. As we have seen in the preliminaries, a fully general rewrite theory is incredibly flexible. In practice, a simpler subset of rewrite theory features is already quite enough for specifying many theories of interest. Here we codify this simpler subset in the notion of a *standard rewrite theory.*

**Definition B.1** *An order-sorted conditional equational theory* $\mathcal{E} = (\Sigma, E \cup B)$ *is called a* standard equational theory *over constructor signature* $\Omega$ *whenever: (1)* $\Omega \subseteq \Sigma$*; (2) the axioms* $B$ *are a combination of associativity, commutativity, and identity; (3) each equation* $f(\vec{t}) = v$ *if* $\phi \in E$*, when oriented as a rule* $f(\vec{t}) \to v$ *if* $\phi \in \vec{E}$*, is* standard*, i.e., it satisfies* $(\vec{t} \in T_\Omega(X)^*) \wedge (f(\vec{t}) \in T_\Omega(X) \Rightarrow v \in T_\Omega(X)) \wedge$ $\phi$ *is a* $\Sigma$*-Conjunction* $\wedge$ $vars(v) \cup vars(\phi) \subseteq vars(\vec{t})$*. A rewrite theory of the form* $\mathcal{R} = (\Sigma, B, \vec{E})$ *is a* standard rewrite theory *when it is derived by orientating the equations in a standard equational theory* $(\Sigma, E \cup B)$*, but where the conditions remain conjunctions of equalities. Let* $\mathcal{E}_\mathcal{R}$ *be the standard equational theory corresponding to* $\mathcal{R}$ *whenever* $\mathcal{R}$ *is ground convergent, i.e., sort-decreasing, ground confluent, and ground operationally terminating.*

## B.2 PROVING TERMINATION VIA AXIOM-COMPATIBLE RPOS

It is well-known that conditional rewrite theories can be proved terminating via recursive path orderings (RPOs). What is less well-known is that RPOs can be automatically lifted to symbols with any combination of associativity or commutativity axioms, due to a result by A. Rubio [138]. This result, combined with a semantics-preserving transformation that allows unit axioms to be entirely removed from a theory from [139], leads to a general procedure for proving termination in theories with associative, commutative, and unit symbols. These ideas have been mechanized in the Maude Termination Assistant [140], which we used to great effect in our IBOS case study.

## B.3 PROVING SUFFICIENT COMPLETENESS/GROUND CONVERGENCE HIERARCHICALLY

### B.3.1 Introduction to Sufficient-Completeness and Ground Confluence Checking

A recurring theme in this thesis is the usage of pattern predicates (possibly with constraints) to reason about terms in initial models of equational theories. In this section, we continue in our tradition of utilizing *constrained pattern predicates* to reason about proving sufficient completeness and ground convergence in a *hierarchical* fashion. This description leads to two obvious questions: (i) in what sense are we reasoning hierarchically; and (ii) how do constrained terms relate to proof obligations for sufficient completeness or ground convergence? To answer question (i), we refer back to similar hierarchical reasoning that was already used in Chapter 3 in the proof of correctness of the sort-sharpening $\Sigma \mapsto \Sigma^\#$ signature transformation. There we redefined signature $\Sigma$ as a "telescope" of signatures $\Sigma_0 \subset \Sigma_1 \subset \cdots \subset \Sigma$ and then inductively built the new signature $\Sigma_0^\# \subset \Sigma_1^\# \subset \cdots \subset \Sigma^\#$. In a similar fashion, here we will break down a standard rewrite theory

$\mathcal{R} = (\Sigma, B, R)$ into an inductive telescope of rewrite theories $\mathcal{R}_0 \subset \mathcal{R}_1 \subset \cdots \subset \mathcal{R}$ and inductively verify sufficient completeness and ground convergence of theory $\mathcal{R}_{i+1}$ *assuming* that $\mathcal{R}_i$ is sufficiently complete and ground convergent. The paper [141] is an important precursor to our work, which used hierarchical techniques for proving confluence.

Given that we have already presented RPO methods for proving termination, we will assume here that $\mathcal{R}$ has already been proved ground operationally termianting. This means in our inductive step, we need only to verify that $R_{i+1}$ is *sufficiently complete* and *confluent* assuming that $R_i$ is sufficiently complete and ground convergent. Thus answering question (ii) requires reviewing standard techniques to prove sufficient completeness and ground confluence.

### Standard Techniques for Proving Sufficient Completeness [59]

To prove sufficient completeness of a theory $\mathcal{R} = (\Sigma, B, \vec{E})$, we first subdivide the symbols of our signature $\Sigma$ into disjoint sets of *constructor* and *defined* symbols, which we will respectively denote $\Omega$ and $\Delta$. The sufficient completeness problem asks whether the set $Irr_{\mathcal{R}}$ of ground irreducible terms modulo axioms $B$ is contained in the constructor term algebra $T_\Omega$. To check sufficient completeness for such a rewrite theory $\mathcal{R}$ with respect to a constructor subsignature $\Omega$, we must show that for each $f : s_1 \cdots s_n \to s \in \Delta$ and for each ground instance $f(t_1, \cdots, t_n)$, there exists a constructor term $u \in T_\Omega$ such that $\mathcal{R} \vdash f(t_1, \cdots, t_n) \to^+ u$. However, assuming we have already proved operational termination of $\mathcal{R}$, it is sufficient to prove that for each defined symbol $f : s_1 \cdots s_n \to s \in \Delta$ and all possible substitutions $\delta \in [\{x_1 : s_1, \cdots, x_n : s_n\} \to T_{\Omega,B}]$, there exists a term $u \in T_\Sigma$ such that $\mathcal{R} \vdash f(x_1 : s_1, \cdots, x_n : s_n)\delta \to^1 u$, i.e. applying each ground constructor substitution enables the defined symbol take *at least one* rewrite step. For each defined symbol, there are a number of *defining* rules, i.e. given $f : s_1 \cdots s_n \to s \in \Delta$, the defining rules for $f$ is the set of rules of the form $R_f = \{f(t_1 \cdots t_n) = v \text{ if } \phi \in R\}$. By definition, an instance $f(x_1 : s_1, \cdots, x_n : s_n)\delta$ will be able to take one rewrite step iff there exists a defining rule $l \to r$ if $\phi \in R_f$ such that $l\alpha = f(x_1 : s_1, \cdots, x_n : s_n)\delta$ and $T_{\Sigma/E \cup B} \models \phi\alpha$.

### Standard Techniques for Proving Ground Confluence [45]

To prove that a theory $\mathcal{R} = (\Sigma, B, R)$ where $\Sigma = ((S, <), F)$ is ground confluent, we typically prove the theory is both *terminating* as well as *ground locally confluent* and *sort-decreasing*. Since we have already covered proving termination with RPOs, we focus here on how to prove ground local confluence and sort-decreasingness. Let us first define some notation. Given a term $t$, let $Pos(t)$ denote the set of positions in the term. Further let $ls(t)$ denote the least sort of $t$ in signature $\Sigma$. Given terms $t$ and $t'$, let $t \downarrow t'$ be the *joinability* relation for $\mathcal{R}$, i.e. $t \downarrow t'$ iff there exists a term $w$ such that $t \to^*_{R,B} w \; {}_{R,B}\!\overset{*}{\leftarrow} t'$. Let us redefine the set of possibly conditional rules $R$ as the $I$-indexed set $R = \{l_i = r_i \text{ if } \phi_i\}_{i \in I}$. For simplicity, assume the variables used in each rule are pairwise disjoint, which can always be assured by renaming. To prove ground local confluence, we must show that each conditional critical pair is ground joinable.

**Definition B.2** *Given $i, j \in I$, $p \in Pos(l_i)$, and $\sigma \in Unif_B(l_{i_p}, l_j)$, a conditional critical pair is a tuple: $(\phi_i \wedge \phi_j)\sigma \Rightarrow (l_i[r_j]_p)\sigma \downarrow r_i\sigma$. A conditional critial pair $\phi \Rightarrow t \downarrow t'$ is called* ground joinable *iff for each ground solution $\tau$ of the condition $\phi$, we have $t\tau \downarrow t'\tau$.*

Using known contextual rewriting techniques (which despite the same name, are somewhat different from the techniques we present below), we can use assumptions $\phi$ in addition to theory $\mathcal{R}$ to prove joinability of $t \downarrow t'$. Unfortunately, complex, conditional critical pairs may still fail to become *provably* ground joinable *even* using the additional assumptions $\phi$.

To prove ground sort-decreasingness, it is sufficient to prove that for each rule and sort specialization of that rule, the rule is sort-decreasing. Given a rule $l \to r$ if $\phi$ with $X = vars(l)$, a *sort specialization* is a substitution $\sigma \in [X \to Y]$ such that for each $x\!:\!s \in X$, $x\!:\!s \mapsto y\!:\!s'$ where $s' < s$. A sort specialization of a rule is sort-decreasing if and only if for all sort specializations $\sigma$, $ls(l\sigma) \geqslant ls(r\sigma)$. Since for this thesis, we will not require new techiques for proving sort-decreasingness, we do not pursue this matter further here.

### Sufficient Completeness and Local Confluence Problems as Constrained Terms

We now return to question (ii) posed at the beginning of this section: in what sense can ground local confluence or sufficient completeness proof obligations be understood as operations on constrained terms? Let us first consider the method of proving sufficient completeness that we presented above. The attentive reader will recall that this is equivalent to the constrained pattern predicate subsumption problem:

$$[\![f(x_1\!:\!s_1,\cdots,x_n\!:\!s_n)]\!] \subseteq \bigcup_{l \to r \text{ if } \phi \in R_f} [\![l \mid \phi]\!] \tag{B.4}$$

Of course, this is just a generalization of the same result we saw in Section 3.7, i.e. if the defining rules for $f$ are all unconditional, the theory has no axioms, and each left-hand side is linear, we can use pattern operations to prove sufficient completeness! Unfortunately, often these assumptions fail to hold. The question becomes: how can we generalize our methods for verifying sufficient completeness when these assumptions fail to hold?

Now, consider the method presented above to prove ground local confluence, i.e. that all conditional critical pairs are ground joinable. By enriching our signature $\Sigma$ with a fresh symbol $(\_ \downarrow \_)$, a conditional critical pair $\phi \Rightarrow t \downarrow t'$ can be viewed as a constrained term or pattern predicate in the signature $\Sigma \cup \{\_ \downarrow \_\}$ of the form $(t \downarrow t') \mid \phi$. Though this observation alone does not seem to provide much benefit, as we will see, by viewing all of our proof obligations as contrained pattern predicates, we gain the ability to transform and simplify these different kinds of proof obligations in a simple, intuitive, and uniform fashion.

### B.3.2 Hierarchical Proof System for Ground Convergence and Sufficient Completeness

We now are ready to present our hierarchical proof system for proving ground convergence and sufficient completeness where, as stated earlier, we have already used standard, non-hierarchical methods to prove termination and sort-decreasingness. There are two key ideas at work here which correspond to the two key questions that we posed at the beginning of this section. The first key idea is that, we are actually proving ground *convergence* and *sufficient completeness* of the entire theory $\mathcal{R}$ via an inductive telescope of theories, where we prove ground convergence and sufficient completeness at each level by assuming the previous level is ground convergent and sufficiently complete. The upshot of this is that we have the full power of inductive equational reasoning for any fragment of our theory below. The second key idea is that, if our proof obligations look like constrained pattern predicates, we can, as a matter of course, use sound constrained pattern predicate transformation techniques to transform a constrained pattern predicate into a set of simpler predicates. Even if the original proof obligation was impossible to prove, the transformed version may often be proved trivially. As before, we must solve two specific problems here: (i) how can we usefully redefine theory $\mathcal{R}$ as an inductive telescope of theories $\mathcal{R}_0 \subseteq R_1 \subseteq \cdots \mathcal{R}_{k-1} \subseteq \mathcal{R}$; (ii) how can we utilize (a) the assumption that $\mathcal{R}_i$ is ground convergent and sufficiently complete and (b) our constrained term representation in proving that $\mathcal{R}_{i+1}$ is ground convergent and sufficiently complete? We answer these questions in turn.

Call-Graph Stratification

We answer question (i) via a definition, that is, the call-graph stratification of a theory $\mathcal{R} = (\Sigma, E \cup B, R)$ over a constructor subtheory $(\Omega, B_\Omega, R_\Omega)$. However, to present this definition, we first must survey a few related concepts.

**Definition B.3** *Given a set $\Sigma$, the set of strings over $\Sigma$ is defined as the monoid $\Sigma^*$. The empty string is the identity element denoted by $\epsilon$. String $x$ is called a substring of $s$, written $x \sqsubseteq s$, whenever $s = wxy$ and $w, y \in \Sigma^*$. The letters in a string is the set $let(s) = \{x \in \Sigma \mid x \sqsubseteq s\}$. Note that a total order $(<) \in \mathcal{P}(\Sigma^2)$ can be lifted to a total order on $\Sigma^*$ by sorting on length and then using a lexicographic construction.*

**Definition B.4** *A* directed graph *over a set of vertices $V$ is a pair $(V, E) \subseteq (V, V \times V)$. Let $G = (V, E)$ be a graph. Then we let $vertices(G) = V$ and $edges(G) = E$. $G$ has a* path *$p \in V^*$ whenever, assuming $u, v \in V$, $uv \sqsubseteq p \Rightarrow (u, v) \in E$. A path is a* cycle *iff it is of the form $vpv \in V^*$ where $v \in V$. A directed graph is* acyclic *(also called a directed acyclic graph or DAG) iff it has no cycles. Observe any DAG has at least one strict and total* topological ordering *$(<_{top})$ on $V$ where $(u, v) \in E \Rightarrow u <_{top} v$.*

Every graph can be transformed by a collapse operation into a DAG. For the definition of the collapse operation, we first define a mapping operation for sets of pairs.

**Definition B.5** *Given a universe $U$ with $V \subseteq U$, we define: (a) the map operation for singletons $\_[V \mapsto \_] \in [U \times U \to U]$ where $u[V \mapsto \alpha] = \underline{if}\ u \in V\ \underline{then}\ \alpha\ \underline{else}\ u\ \underline{fi}$; (b) the set-of-pairs-lifting of the map operation where if $E \subseteq U^2$ and $(u, v) \in E$, then the resulting set is defined by $(u[V \mapsto \alpha], v[V \mapsto \alpha]) \in E[V \mapsto \alpha]$.*

We now define how to collapse a cycle and the DAG generated by a directed graph.

**Definition B.6** *Given a graph $G = (V, E)$ where $V$ has total order $(<)$, let $cycle(G)$ be the set of all cycles in $G$. Let $c$ be the maximum element in $cycle(G)$ and further let $L = let(c)$ and $\overline{L} = \{L\}$. The* maximum cycle-collapsed graph *$collapse(G) = (\overline{L} \uplus (V - L), E[L \mapsto \overline{L}])$.*

Note that, when collapsing a cycle in a graph $G = (V, E)$, a new vertex is added which itself is a *set of deleted vertices*. This means that, after collapsing a cycle, we still know which vertices were in the cycle; only their relative ordering information is discarded.

**Definition B.7** *The DAG generated by a finite graph $G$ can be characterized by the solution to the recursive equation:*

$$dag(G) = \begin{cases} G & \text{if } cycle(G) = \varnothing \\ dag(collapse(G)) & \text{otherwise.} \end{cases} \tag{B.5}$$

*The function terminates because the value of the function $f$ mapping graphs to $\mathbb{N}$ defined by $f(G) \mapsto |cycle(G)|$ strictly decreases on each recursive call.*

At this point, we have finished defining the necessary mathematical framework. We now are apply these concepts to define the non-constructor call graph for a theory.

**Definition B.8** *Given an order-sorted signature $\Delta$, let $\Delta^{ms}$ denote the corresponding many-sorted signature which erases subsort distinctions, and given $\Gamma \subseteq \Delta^{ms}$, let $\Gamma^{\#} \subseteq \Delta$ denote the expansion of many-sorted operators into corresponding subsort-overloaded operator families in $\Delta$.*

**Definition B.9** *Given a signature $\Sigma$ with a constructor subsignature $\Omega$ and defined symbols $\Delta = \Sigma - \Omega$, the defined symbols in a term $t \in T_\Sigma$, written $def(t)$, is the set of symbols in $\Delta$ that occur at any position in $t$. Similarly, given a conjunction of $\Sigma$-equalities $G = \bigwedge_i u_i = v_i$, the defined symbols in $G$, written $def(G)$, is the set of symbols $\bigcup_i \big(def(u_i) \cup def(v_i)\big)$.*

**Definition B.10** *Given a standard rewrite theory $\mathcal{R} = (\Sigma, B, R)$ over $\Omega$ where $\Delta = \Sigma - \Omega$, the* non-constructor call graph *$call(\mathcal{R})$ is a directed graph $(\Delta^{ms}, E)$ defined by the following rule $(g, f) \in E \Leftrightarrow \big[f(\vec{t}) \to u \text{ if } \phi \in R \wedge f(\vec{t}) \notin T_\Omega \wedge g \in def(u) \cup def(\phi)\big]$.*

In fact, the graph we are primarily interested in is $dag(call(\mathcal{R}))$. Since this graph is a DAG, it has at least one associated topological ordering. Note that the vertices in this graph are non-empty sets of many-sorted operators, i.e. elements of the set $\mathcal{P}(\Delta^{ms}) - \varnothing$, that *represent* subsort-overloaded families of defined function symbols.

**Definition B.11** *Given a standard rewrite theory $\mathcal{R}$, call any topological order corresponding to $dag(call(\mathcal{R}))$ a* definitional order *($<_{def}$). Since the definitional order is total, we may write its elements as an increasing sequence, e.g. $\Delta_1^{ms} <_{def} \Delta_2^{ms} \cdots <_{def} \Delta_k^{ms}$. For each $1 \leqslant i \leqslant k$, let $\Delta_i = (\Delta_i^{ms})^{\#}$, i.e. the sequence mapped onto its order-sorted lifting.*

The definitional order has several interesting properties. It exactly describes when a subsort-overloaded family depends on another set, and every non-singleton element corresponds to a mutually defining set of non-constructor operator families. However, it is clear from practice that the $\Delta_i$ sequence may actually be *too coarse*, in that proving sufficient completeness or ground convergence for more general operators *may depend* on proving those same properties for less general operators. Finding the right heuristics for soundly and automatically splitting operator-overloaded families is left for future work.

In any case, the definitional order provides the backbone for our inductive telescope.

**Definition B.12** *A call-graph stratification of a standard rewrite theory $\mathcal{R}$ over a constructor theory $\mathcal{R}_\Omega$ is an inductive telescope corresponding to a definitional order:*

$$\begin{gathered}\Omega < \Delta_1 < \Delta_2 \cdots < \Delta_{k-1} < \Delta_k \\ \mathcal{R}_\Omega = \mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \mathcal{R}_2 \cdots \subseteq \mathcal{R}_{k-1} \subseteq \mathcal{R}_k = \mathcal{R}\end{gathered} \tag{B.6}$$

*where if $\mathcal{R}_i = (\Pi_i, B_i, R_i)$, we let $\mathcal{R}_{i+1} = (\Pi_i \uplus \Delta_{i+1}, B_i \uplus B_{\Delta_{i+1}}, R_i \uplus R_{\Delta_{i+1}})$ such that $B_{\Delta_i}$ (resp. $R_{\Delta_i}$) is the set of axioms (resp. defining rules) for all symbols in $\Delta_i$.*

Hierarchical Proof Strategy and Object Theory

Using this telescope of theories, we will inductively prove that $\mathcal{R}$ is both ground convergent and sufficiently complete. At this point we are ready to tackle question (ii) posed above: how can we utilize (a) the assumption that $\mathcal{R}_i$ is ground convergent and sufficiently complete and (b) our constrained term representation in the proving that $\mathcal{R}_{i+1}$ is ground convergent and sufficiently complete (equivalently, ground locally confluent and sufficiently complete when termination and sort-decreasingness have already been proved)?

**Definition B.13** *A* hierarchical proof strategy for sufficient completeness and ground convergence *is defined as an application of the strategy below:*

1. *Prove operational termination and sort-decreasingness of $\mathcal{R}$ using standard methods.*
2. *Prove $\mathcal{R}_0$, i.e. $\mathcal{R}_\Omega$, is ground locally confluent if needed.*
3. *For each $1 \leqslant i \leqslant k$, assuming $\mathcal{R}_{i-1}$ is sufficiently complete and ground convergent:*
   - *(a) first prove sufficient completeness of symbols $\Delta_i$ in $\mathcal{R}_i$,*
   - *(b) then prove ground local confluence of symbols $\Delta_i$ in $\mathcal{R}_i$.*

Given the proof strategy outlined above, the proof system we have in mind actually breaks down into two separate proof systems, one for sufficient completeness and another for ground local confluence. Due to their common representation, they share several structural rules. In fact, these structural rules are analogues of the auxiliary rules presented in Section 5.4.1.

Of course, as is the case in all of our proof systems presented upto this point, we define our proof system at the meta-level of rewriting logic, i.e. our proof system is takes a rewrite theory as input $\mathcal{R}$ and then determines whether it is ground convergent and sufficiently complete. An important question that we need to address in the construction of our hierarchical system is, when proving the ground convergence/sufficient completeness of $\mathcal{R}_i$, in which object theory should we reason about $\mathcal{R}_i$'s ground convergence/sufficient completeness proof obligations? The choice is more subtle than it it first appears for the simple reason that, because we have not yet verified $\mathcal{R}_i$ is sufficiently complete, our sorts in signature $\Pi_i$ are now "polluted" by the defined symbols $\Delta_i$. We cannot perform case analysis or induction on these polluted sorts in the standard sense, since they both require sufficient completeness for $\Delta_i$, the very property we are trying to prove! The way out of this mess is to not reason in $\mathcal{R}_i$ directly, but rather on its $\Delta$-*kind-lifting*.

**Definition B.14** *Given a call-graph stratification for $\mathcal{R}$ and index $i+1$, $\mathcal{R}_{i+1}$'s* $\Delta$-kind-lifting*, $\mathcal{R}_{i+1}^{\Delta}$, is equivalent to $\mathcal{R}_{i+1}$ except that defined symbols $\Delta_{i+1}$ are only added at the* kind-level*.*

Hierarchical Proof System Initial Goals and Rules

As we apply the hierarchical proof strategy in Def. B.13 at each level in step 3(a)-(b), we first generate an initial set of proof obligations. This set of proof obligations corresponds to a *proof forest*, such that, if each proof tree in the forest is closed, the corresponding property (sufficient completeness or local confluence) is proved for theory $\mathcal{R}_i$. We define the initial proof obligation set below in the obvious way.

**Definition B.15** *For theory level $\mathcal{R}_i$ with $i \neq 0$, we have an initial sufficient completeness (resp. ground local confluence) proof goal set $SC(i)$ (resp. $GLC(i)$) defined by:*

$$(f(x_1{:}s_1, \cdots x_n{:}s_n) \mid \top) \in SC(i) \Leftrightarrow f : s_1 \cdots s_n \to s \in \Delta_i \wedge f \textit{ maximal in } \Delta_i \tag{B.7}$$

$$GLC(i) = MCP(R_{\Delta_i}, R_{i-1}) \cup MCP(R_{\Delta_i}, R_{\Delta_i}) \tag{B.8}$$

*where $MCP(R_1, R_2)$ is a set of constrained terms corresponding to the most general set of critical pairs between any rule in $R_1$ and $R_2$. As an optimization, note that if (a) no subsort-overloaded symbol is shared at the top of a left-hand side of both $R_1$ and $R_2$ and (b) no axiom can modify the root of a term, then the set $MCP(R_1, R_2)$ is immediately empty.*

At long last, in Table B.1 we present our proof system rules based on terms constrained by conjunctions of equalities. The rules should be understood to apply *modulo* (a) the commutativity axiom for equality $\_ = \_$ and (b) the associativity, commutativity, and unit axioms for conjunction $\_ \wedge \_$ with unit $\top$. Note that applying our rules modulo axioms greatly simplifies the presentation of the rules, e.g., the SUBSTIUTION rule has particularly elegant representation. Also note that two rules specific to the ground local confluence proof system are directly pulled from [45]; a complete explanation and justification of those rules can be found there. Recall that we are viewing ground confluence proof obligations as *terms* in the enriched signature $\Sigma \uplus (\_ \downarrow \_)$; in this theory, the shared rules become applicable. Finally, as mentioned above, we leave a complete soundness proof as future work.

## B.4 PROVING COMPLETENESS OF VARIANT UNIFICATION IN AN FVP SUBTHEORY

Given (a) a standard rewrite theory $\mathcal{R}$ that is not FVP with an FVP subtheory $\mathcal{Q}$, i.e. $\mathcal{Q} = (\Gamma, B, Q) \subseteq (\Sigma, B, R) = \mathcal{R}$, and (b) a conjunction of $\Sigma$-equalities $\phi$, under what conditions is $\phi$ *variant solvable in FVP subtheory* $\mathcal{Q}$, i.e., has a *complete* set of variant unifiers in $\mathcal{E}_{\mathcal{Q}}$, so that $\mathit{Unif}_{\mathcal{E}_{\mathcal{R}}}(\phi) = \mathit{Unif}_{\mathcal{E}_{\mathcal{Q}}}(\phi)$? Note that we do *not* require that $\mathcal{R}$ *protects* $\mathcal{Q}$, but even if that were so, it would only guarantee completeness of variant unification if $\phi$ were a conjunction of $\Gamma$-equalities.

To simplify the problem, we assume that $\mathcal{R}$ already satisfies our executability conditions, i.e., it is sufficiently complete and ground convergent. $\mathcal{Q}$ must be convergent, as it is FVP, but we do not require it to be sufficiently complete. Furthermore, here we will only seek conditions which are *sufficient* to prove completeness of the variant unification.

The intuition we wish to apply here is, given some defined symbol $f \in \Delta$ and its set of defining rules $R_f = \{f(\vec{t}) \to u \text{ if } \phi \in R\}$, a subset of these rules $Q_f \subseteq R_f$, when considered alone, may *obviously* generate a finite set of variants.

**Example B.1** *Suppose we are specifying a theory $\mathcal{R}$ of finite binary relations on natural numbers, i.e. the finite relations which are a subset of $\mathbb{N} \times \mathbb{N}$. Assume our theory has sorts Bool, Nat, Pair, NeRel, and Rel with subsorts Pair < NeRel < Rel where sort Bool has constants $\top$ and $\bot$, sort Nat has constant $0$ and the successor function $s$, there is a boolean-valued natural number equality predicate $(\sim)$, the sort Pair has the natural number pairing operator $(\_,\_)$, the sort NeRel has juxtaposition $(\_\_)$ as an AC relation union operator, and the sort Rel has constant $\varnothing$. Also assume the FVP idempotency equations $A, A = A$ and $A, A, A' = A, A'$ where $A, A'$ are variables with sort NeRel. Then, by abuse of notation using the standard numeric syntax, we may write relation terms such as:*

$$(0,1)\ (1,2)\ (0,2) \tag{B.9}$$

$$(0,1)\ (1,0)\ (0,0) \tag{B.10}$$

*An operation we may wish to define is an irreflexivity test, i.e., $irr(A)$ iff $\nexists (n,n) \in A$. This operation is*

Table B.1: Proof System for Sufficient Completeness/Ground Convergence of $\mathcal{R}_{i\neq 0}$

| Name | Rule | Condition |
|---|---|---|
| | Shared Rules | |
| Case Analysis | $\frac{\bigwedge_{\vec{a}\in M}(u \mid \phi)[\vec{x}/\vec{a}]}{(u \mid \phi)}$ | $\vec{x} = x_1 : s_1 \cdots x_n : s_n$<br>$[\![M]\!] = T_{\Omega_{s_1}} \times \cdots \times T_{\Omega_{s_n}}$<br>$vars(M) \cap vars(u \mid \phi) = \varnothing$ |
| Generalization | $\frac{(v \mid \psi)}{(u \mid \phi)}$ | $\exists \alpha \,.\, v\alpha =_{B_i} u$<br>$\mathcal{E}_{\mathcal{R}_{i-1}} \models \phi \Rightarrow \psi\alpha$ |
| Simplification | $\frac{(u \mid \phi \wedge t!_{\mathcal{R}_{i-1}} = t')}{(u \mid \phi \wedge t = t')}$ | $t \in T_{\Pi_{i-1}(X)}$ |
| Split | $\frac{\bigwedge_{\varphi\in\Phi}(u \mid \phi \wedge \varphi)}{(u \mid \phi)}$ | $\Phi$ is set of $\Pi_i$-Formulas<br>$T_{\mathcal{R}_i} \models \bigvee \Phi \Leftrightarrow \top$ |
| Substitution | $\frac{\bigwedge_{\theta\in\Theta}(u \mid \phi)\theta}{(u \mid \phi \wedge \psi)}$ | $\mathit{Unif}_{\mathcal{E}_{\mathcal{R}_{i-1}}}(\psi) = \Theta$ |
| | Sufficient Completeness Rules | |
| Closure | $\frac{}{(u \mid \phi)}$ | $u \to v$ if $\phi \in R_{\Delta_i}$ |
| | Ground Local Confluence Rules | |
| Trivial Joinability | $\frac{}{(t \downarrow t \mid \phi)}$ | |
| Context Joinability | $\frac{}{(s \downarrow t \mid \phi)}$ | $\exists u \,.\, \bar{s} \to^{*}_{R_i \cup \bar{\phi}, B_i} u$<br>$\bar{t} \to^{*}_{R_i \cup \bar{\phi}, B_i} u$ |
| Unfeasibility | $\frac{}{(s \downarrow t \mid \phi)}$ | $\exists u = v \in \phi \,.\, \exists t_1, t_2 \,.$<br>$\bar{u} \to^{*}_{R_i \cup \bar{\phi}, B_i} t_1$<br>$\bar{u} \to^{*}_{R_i \cup \bar{\phi}, B_i} t_2$<br>$t_1 \neq_{B_i} t_2$<br>$(t_1, t_2) = (t_1, t_2)!_{\mathcal{R}_i}$ |

*definable with our given syntax by the recursive conditional equations:*

$$\mathit{irr}(\ (n,n)\ A\ ) = \bot \tag{B.11}$$

$$\mathit{irr}(\ (n,n)\quad\ ) = \bot \tag{B.12}$$

$$\mathit{irr}(\ (n,m)\ A\ ) = \mathit{irr}(A)\ \mathit{if}\ n \sim m = \bot \tag{B.13}$$

$$\mathit{irr}(\ (n,m)\quad\ ) = \top\ \mathit{if}\ n \sim m = \bot \tag{B.14}$$

$$\mathit{irr}(\ \varnothing\ ) = \top \tag{B.15}$$

*The rules $R_{irr}$ clearly are not FVP; they have unbounded recursion. However, if we consider the first two equations as the set $Q_{irr}$, then for those rules, the irreflexivity test is FVP.*

The problem with Ex. B.1 is, of course, that the information that $Q_{irr}$ is FVP alone does *nothing* for us. Completeness of variant unification clearly requires a sufficiently complete theory, but any guarantee of sufficient completeness is lost when we decide to arbitrarily throw away defining rules.

The key point that we present in this section is that, even though $\mathcal{R}$ is not FVP, certain conjunctions of equalities may be *variant solvable in an FVP subtheory.* For example, as we later prove, the variant unification of *any instance* of equality $\mathit{irr}(A) = \bot$ is *complete* in FVP subtheory $\mathcal{Q}$, so that the *undecidable* variant unification problem $\mathit{Unif}_{\mathcal{E}_\mathcal{R}}(\mathit{irr}(A) = \bot)$ can be solved by merely considering the *decidable* variant unification problem $\mathit{Unif}_{\mathcal{E}_\mathcal{Q}}(\mathit{irr}(A) = \bot)$.

We can state our desired result formally.

**Definition B.16** *Suppose we are considering a convergent and sufficiently complete standard rewrite theory $\mathcal{R} = (\Sigma, B, R)$ that protects an FVP subtheory with signature $\Gamma$. A conjunction of $\Sigma$-equalities is variant solvable iff each equality is variant solvable. A $\Sigma$-equality $\phi \equiv (u = v)$ is variant solvable if (a) $\phi$ is also a $\Gamma$-equality (trivially true when $\mathcal{R}$ itself is FVP); (b) $u = f(\vec{t})$, $\vec{t} \in T_\Gamma(X)^*$, $\exists p \in \mathit{fvp\text{-}constraints}(f, \mathcal{R})$ and substitution $\alpha$ such that $v = p\alpha$.*

In Ex. B.1, unsurprisingly, the function *fvp-constraints*$(\mathit{irr}, \mathcal{R}) = \{\bot\}$. Intuitively, the set of *fvp constraints* for a symbol $f$ consists of a set of constructor patterns $\{p_1, \cdots, p_k\}$, such that, given $\mathcal{Q}$ is an FVP subtheory of $\mathcal{R}$ generated by deleting badly behaved rules from $R_f$, the unifiers for $\mathit{Unif}_{\mathcal{E}_\mathcal{Q}}(f(\vec{t}) = p_i) = \mathit{Unif}_{\mathcal{E}_\mathcal{R}}(f(\vec{t}) = p_i)$ for $1 \leqslant i \leqslant k$.

Of course, the question then is, how can we compute a set of *fvp-constraints* and an FVP subtheory with complete variant unification for a given function symbol $f$? Armed with a clearer intuition, we now proceed to provide a set of conditions which enable this somewhat surprising result. Along the way, we will define a number of related concepts.

Firstly, we state some defintions related to *call graphs* (see Def. B.4 and B.10).

**Definition B.17** *Given a rewrite theory $\mathcal{R} = (\Sigma, B, R)$, we can* unconditionalize *$\mathcal{R}$ and obtain $\mathit{uncond}(\mathcal{R}) = (\Sigma, B, \{l \to r \mid (l \to r\ \mathit{if}\ \phi) \in R\})$.*

**Definition B.18** *Suppose $\mathcal{R} = (\Sigma, B, R)$ is standard and sufficiently complete with respect to a constructor subsignature $\Omega$ and where $\Delta = \Sigma - \Omega$. Let $E_U = \mathit{edges}(\mathit{call}(\mathit{uncond}(\mathcal{R}))) \subseteq \mathit{edges}(\mathit{call}(\mathcal{R})) = E$. Let $f \in \Delta$. Then we say that $f$ is: (a)* self-recursive *whenever $\mathit{call}(\mathcal{R})$ has a cycle $fwf \in (\Delta^m)^*$; (b)* immediately self-recursive *whenever $(f, f) \in E$ and* tail-recursive *if additionally $f(\vec{t}) \to f(\vec{u})\ \mathit{if}\ \phi \in R_f$; (c)* essentially self-recursive *whenever* $(g, f) \in E_U \Rightarrow g = f$ *and* essentially tail-recursive *if additionally* $f(\vec{t}) \to v$ *if* $\phi \in R_f \Rightarrow [(v = f(\vec{u}) \wedge \vec{u} \in T_\Omega^*) \vee v \in T_\Omega]$*; and (d)* totally self-recursive *whenever* $(g, f) \in E \Rightarrow g = f$ *and* totally tail-recursive *if additionally all occurences of* $f$ *in* $R_f$ *are at a topmost position.*

A key notion we will exploit is that of an *essentially tail-recursive* function $f$; such functions ensure the rewrite relation satisfies a particular invariant that we will examine below.

**Definition B.19** *Suppose rewrite theory* $\mathcal{R} = (\Sigma, B, R)$ *is ground convergent and sufficiently complete with respect to a constructor subsignature* $\Omega$ *and protects a FVP subtheory on signature* $\Gamma$*. Then (a) term* $t \in T_\Sigma(X)$ *is* FVP *iff* $t \in T_\Gamma(X)$*; (b) term* $f(\vec{t}) \in T_\Sigma(X)$ *is* FVP below *iff* $\vec{t} \in T_\Gamma(X)^*$*; (c) term* $f(\vec{t}) \in T_\Sigma(X)$ *has* constructors below *iff* $\vec{t} \in T_\Omega(X)^*$*.*

We now present the promised invariant: suppose as above that $f$ is essentially tail-recursive and additionally assume $f(\vec{t}) \in T_\Sigma(X)$ has constructors below. Then it must be the case that if $f(\vec{t}) \to_{R,B}^* t'$, then $t' = f(\vec{u})$ with constructors below or else $t'$ is a constructor. As we will see, this invariant greatly simplifies reasoning about the evaluation of $f$.

**Definition B.20** *Suppose* $\mathcal{R} = (\Sigma, B, R)$ *is a standard rewrite theory that is ground convergent and sufficiently complete with respect to subsignature* $(\Omega, B_\Omega)$*. Suppose* $f \in \Sigma$ *is essentially tail-recursive. The defining rules for* $f$ *can be partitioned into* $R_f = R_{\mathit{fin},f} \uplus R_{\mathit{rec},f}$ *where:*

1. *final rules* $R_{\mathit{fin},f}$ *have the form* $f(\vec{t}) \to u$ *if* $\phi$ *where* $u \in T_\Omega(X)$*; and*
2. *recursive rules* $R_{\mathit{rec},f}$ *have the form* $f(\vec{t}) \to f(\vec{u})$ *if* $\phi$ *with* $\vec{u} \in T_\Omega(X)^*$*.*

*Note that narrowing by unconditional final rules always terminates (assuming we are performing unification modulo a theory with a finitary unification algorithm).*

In Ex. B.1, we observe that *irr* is an essentially tail-recursive function with uncondtional final rules. Note that functions or predicates that select or project an element out of a set or list often have this structure.

**Lemma B.1** *Suppose* $\mathcal{R} = (\Sigma, B, R)$ *is standard and sufficiently complete with respect to* $(\Omega, B_\Omega)$ *and protects an FVP subtheory* $\mathcal{Q} = (\Gamma, B, Q)$*. Let* $f$ *be essentially tail-recursive, and let* $R_{\mathit{fin},f} = R_f^+ \uplus R_f^-$*, with rules in* $R_f^+$ *unconditional, such that:*

1. *if* $f(\vec{t}) \to p \in R_f^+$ *and* $f(\vec{u}) \to w$ *if* $\phi \in R_f^-$*, then for* $\alpha \in \mathit{Unif}_{B_\Omega}(p = w)$*,* $\mathcal{R} \not\vdash \phi\alpha$*;*
2. *if* $f(\vec{t}) \to f(\vec{u})$ *if* $\phi \in R_{\mathit{rec},f}$ *and* $f(\vec{w}) \to p \in R_f^+$ *and* $\alpha \in \mathit{Unif}_{B_\Omega}(f(\vec{u}) = f(\vec{w}))$*,*
   *then* $f(\vec{t})\alpha$ *must be rewritable by a positive rule assuming* $\phi\alpha$*.*

*Then fvp-constraints*$(f, \mathcal{R}) \subseteq T_\Omega(X)$ *is given by* $p \in$ *fvp-constraints*$(f, \mathcal{R}) \Leftrightarrow f(\vec{t}) \to p \in R_f^+$*, and* $\mathcal{R}$ *has an FVP subtheory* $\mathcal{Q}^+ = (\Gamma \uplus \{f\}, B, Q \uplus R_f^+)$ *where if* $\vec{t} \in T_\Gamma(X)^*$ *then:*

$$\mathit{Unif}_{\mathcal{E}_{\mathcal{Q}^+}}(f(\vec{t}) = p) = \mathit{Unif}_{\mathcal{E}_\mathcal{R}}(f(\vec{t}) = p) \tag{B.16}$$

*Proof.* To complete the proof, we must show that $\mathit{Unif}_{\mathcal{E}_{\mathcal{Q}^+}}(f(\vec{t}) = p) = \mathit{Unif}_{\mathcal{E}_\mathcal{R}}(f(\vec{t}) = p)$ for $p \in$ *fvp-constraints*$(f, \mathcal{R})$ and $\vec{t} \in T_\Gamma(X)^*$. Since $\vec{t} \in T_\Gamma(X)^*$, apply folding variant narrowing to generate a set of most general constructor variants. Thus, without loss of generality, let $\vec{t} \in T_\Omega(X)^*$. Now consider any unifier $\alpha \in \mathit{Unif}_{\mathcal{E}_{\mathcal{R}}}(f(\vec{t}) = p)$. Since $p \in T_\Omega(X)$ is in normal form and $f(\vec{t}) \notin T_\Omega(X)$, any unifier $\alpha$ must result from a rewrite $f(\vec{t})\alpha \to^{!}_{R,B} u$ with $u \in T_\Omega(X)$ with $u =_{B_\Omega} p\alpha$. Proceed by induction on the length of the rewrite path $f(\vec{t})\alpha \to^{!}_{R,B} u$.

**Base case**: Let $f(\vec{t})\alpha \to^{1}_{R,B} u$ with $u \in T_\Omega(X)$. By definition, we must use some rule $R_{fin,f}$. By assumption, if $f(\vec{t})\alpha \to^{1}_{R^-_f,B} u$ then $\mathit{Unif}_{B_\Omega}(u = p) = \varnothing$ which is a contradiction. Thus, we must have $f(\vec{t})\alpha \to^{1}_{R^+_f,B} u$, validating $\mathit{Unif}_{\mathcal{E}_{\mathcal{Q}^+}}(f(\vec{t}) = p) = \mathit{Unif}_{\mathcal{E}_{\mathcal{R}}}(f(\vec{t}) = p)$.

**Inductive case**: Suppose if $f(\vec{t}) \to^{n}_{R,B} u$ with $u \in T_\Omega(X)$, then ($B$.16) holds. By definition of unification modulo equations, ($B$.16) implies $f(\vec{t}) \to^{1}_{R^+_f,B} u$. Now we must prove that ($B$.16) holds when $f(\vec{t})\alpha \to^{n+1}_{R,B} u$ with $u \in T_\Omega(X)$. By transitivity of rewriting and definition of essentially tail-recursive, this means $f(\vec{t})\alpha \to^{1}_{R_{\text{rec,f}},B} f(\vec{v}) \to^{n}_{R,B} u$ with $\vec{v} \in T_\Omega(X)$. But note $f(\vec{v}) \to^{n}_{R,B} u$ is an instance of our inductive hypothesis, and thus satisfies ($B$.16). But this implies $f(\vec{v}) \to^{1}_{R^+_f,B} u$ by the observation above. Then by assumption on rewrite $f(\vec{t})\alpha \to^{1}_{R_{\text{rec,f}},B} f(\vec{v})$ with a rule in $R_{\text{rec,f}}$, we must have a rule $f(\vec{s}) \to p' \in R^+_f$ that can rewrite $f(\vec{t})\alpha$, so that $f(\vec{t})\alpha \to^{1}_{R^+_f,B} u'$ with $u' \in T_\Omega(X)$. Furthermore, by confluence of $\mathcal{R}$, a rewrite with any rule $R^-_f$ is impossible. □

Note that, for Ex. B.1, the required conditions of Lemma B.1 are easily checkable.

## B.5 PROVING VALIDITY USING CONTEXTUAL REWRITING

We now explore contextual rewriting, our final auxiliary proof technique explored in this appendix. Recall from the Introduction that contextual rewriting is a technique for proving validity of clauses, i.e., by using the CNF transformation, we consider problems of the form:

$$(\Sigma, E \cup B) \models \bigwedge\nolimits_i \left[ \left( \bigwedge\nolimits_k u_{i,k} = v_{i,k} \right) \Rightarrow \left( \bigvee\nolimits_j u_{i,j} = v_{i,j} \right) \right] \tag{B.2}$$

This form suggests a possible reasoning technique, that is, to apply the theorem of constants and deduction theorem to transform our problem into the equivalent problem:

$$\bigwedge\nolimits_i \left[ (\Sigma \uplus V_i, E \cup B \cup \{\overline{u}_{i,k} = \overline{v}_{i,k}\}_k) \models \left( \bigvee\nolimits_j \overline{u}_{i,j} = \overline{v}_{i,j} \right) \right] \tag{B.3}$$

As we mentioned earlier, the problem is, if $(\Sigma, E \cup B)$ is a standard equational theory and $(\Sigma, B, \vec{E})$ is a standard rewrite theory that is ground convergent, when we consider the new standard rewrite theory, $(\Sigma \uplus V_i, B, \vec{E} \cup \{\overline{u}_{i,k} \to \overline{v}_{i,k}\}_k)$, any guarantees of convergence are immediately lost. Thus, even though the equational reasoning is sound, it is likely hopelessly inefficient. The question becomes: how can we recover good executability properties for this theory so that we can mechanize execution via efficient rewriting-based methods? Here we essentially combine two existing techniques: (a) recursive path orders (RPOs) and (b) a congruence closure algorithm.

As a starting point, assume that $\mathcal{R} = (\Sigma, B, \vec{E})$ is a standard rewrite theory that is ground convergent. Let us additionally add the requirement that we have labelled each subsort-overloaded operator family $f : [s_1] \cdots [s_n] \to [s] \in \Sigma$ with a unique number which defines an RPO, which we denote by $(\succ)$. Finally, we require that this RPO is compatible with the rewrite relation induced by equations $E$, i.e., $\to_{\vec{E},B}$.

Now, let us denote the ground congruence closure modulo $B$ of a set of ground equations $G$ using ordering $(\succ)$ by $CC_B(G, \succ)$. By definition of congruence closure, $(\Sigma \uplus V_i, B \cup CC_B(\{\overline{u}_{i,k} = \overline{v}_{i,k}\}_k, \succ))$ must be a ground convergent theory. Furthermore, $(\Sigma \uplus V_i, B, \vec{E} \cup CC_B(\{\overline{u}_{i,k} \rightarrow \overline{v}_{i,k}\}_k, \succ))$, while possibly non-confluent, is at least a *terminating* theory, since all rules in $\vec{E}$ and in $CC_B(\{\overline{u}_{i,k} \rightarrow \overline{v}_{i,k}\}_k, \succ)$ are compatible with well-founded order $(\succ)$. This means that we can consider the problem:

$$\bigwedge\nolimits_i \left[(\Sigma \uplus V_i, B, \vec{E} \cup CC_B(\{\overline{u}_{i,k} = \overline{v}_{i,k}\}_k, \succ)) \models \left(\bigvee\nolimits_j \overline{u}_{i,j} = \overline{v}_{i,j}\right)\right] \tag{B.17}$$

where any equality $\overline{u}_{i,j} = \overline{v}_{i,j}$, though not necessarily checkable by arbitrary rewriting, is checkable via: (i) a computable search of normal forms of both $\overline{u}_{i,j}$ and $\overline{v}_{i,j}$ using the rewrite relation $\rightarrow^!_{\vec{E} \cup CC_B(\{\overline{u}_{i,k} = \overline{v}_{i,k}\}_k, \succ), B}$; (ii) a cheap $B$-equality check between normal forms. This gives a computable and sound method to check the validity of each clause.

Of course, in this dicussion, one important detail has been omitted; since we are rewriting modulo $B$, we must compute congruence closure modulo $B$ as well. Fortunately, due to recent work in [117], this is now possible.

# APPENDIX C OMITTED PROOFS FROM CHAPTER 4

## C.1 AUXILIARY LEMMAS FOR SECTION 4.3.1

### C.1.1 Sort Emptiness

In the first part of this appendix we develop an algorithm that checks if a sort $s \in S$ satisfies $T_{\Omega,s} = \varnothing$ by performing rewriting in the theory $\mathcal{R}_M$ over the set $\mathcal{P}(S)$. The initial state is the sort we wish to check for non-emptiness. We then trace the operator declarations in reverse to see which sorts are needed to build operators inhabiting the argument sort. The end result of this subsection is the proof of Lemma 4.1.

**Definition C.1** *Let $\mathcal{R}_M(\Omega) = (\Omega_M, ACI, R_M)$ where:*

*(1) $\Omega_M = S \uplus \{\boldsymbol{*}\} \uplus \{\_\boldsymbol{,}\_\}$ (an unsorted signature)*

*(2) $ACI = \{\boldsymbol{x}\boldsymbol{,}\boldsymbol{y} = \boldsymbol{y}\boldsymbol{,}\boldsymbol{x}\} \cup \{(\boldsymbol{x}\boldsymbol{,}\boldsymbol{y})\boldsymbol{,}\boldsymbol{z} = \boldsymbol{x}\boldsymbol{,}(\boldsymbol{y}\boldsymbol{,}\boldsymbol{z})\} \cup \{\boldsymbol{x}\boldsymbol{,}\boldsymbol{x} = \boldsymbol{x}\}$*

*(3) $R_M$ is the smallest rewrite relation such that:*

*(a) $(s, s') \in (<) \Rightarrow s' \rightarrow s \in R_M$*

*(b) $c : \rightarrow s \in F \Rightarrow s \rightarrow \boldsymbol{*} \in R_M$*

*(c) $f : s_1 \cdots s_k \rightarrow s \in F \wedge k \geqslant 1 \Rightarrow s \rightarrow s_1 \boldsymbol{,} \cdots \boldsymbol{,} s_k \in R_M$*

In this subsection, let $(\rightarrow) \subseteq T_{\Omega_M} \times T_{\Omega_M}$ abbreviate $(=_{ACI}; \rightarrow_{R_M}; =_{ACI})$. We further let $(\rightarrow^0) = (=_{ACI})$, $(\rightarrow^{n+1}) = (\rightarrow); (\rightarrow^n)$, $(\rightarrow^*) = \bigcup_{n \geqslant 0} (\rightarrow^n)$, and also $(\rightarrow^+) = \bigcup_{n > 0} (\rightarrow^n)$.

**Lemma C.1** *Let $a_1, \ldots, a_k$, $k \geqslant 1$ be a ground $\Omega_M$-term, so that $a_i \in S \uplus \{\boldsymbol{*}\}$, i.e., $a_1, \ldots, a_k$ is a* multiset*. If $a_1, \ldots, a_k \rightarrow^n \boldsymbol{*}$, then for each nonempty submultiset $B \subseteq a_1, \ldots, a_k$ there is an $m \leqslant n$ such that $B \rightarrow^m \boldsymbol{*}$.*

**Proof C.1** *By induction on $n$.*
**Base Case**. *If $n = 0$ we must have $a_i = \boldsymbol{*}$, $1 \leqslant i \leqslant k$, and the result follows trivially.*
**Induction Step**. *Suppose the result true for n and let $a_1, \ldots, a_k \rightarrow^{n+1} \boldsymbol{*}$. Since rewriting takes place modulo $ACI$ we may assume without loss of generality that $i \neq j \Rightarrow a_i \neq a_j$. Then we must have some $a_i \in S$, a rule $a_i \rightarrow D$ in $R_M$, and rewrites*

$$a_1, \ldots, a_k \rightarrow a_1, \ldots, a_{i-1}, D, a_{i+1}, \ldots, a_n \rightarrow^n \boldsymbol{*}. \tag{C.1}$$

*Note that $a_1, \ldots, a_{i-1}, D, a_{i+1}, \ldots, a_n$ may have repeated elements. We now reason by cases on $B \subseteq a_1, \ldots, a_k$. If $a_i \notin B$, then $B \subseteq a_1, \ldots, a_{i-1}, D, a_{i+1}, \ldots, a_n$ and the result follows trivially by the induction hypothesis. If $B = a_i, B'$ (where by convention $B'$ could be empty), then $B \rightarrow D, B'$ and we have an inclusion $D, B' \subseteq a_1, \ldots, a_{i-1}, D, a_{i+1}, \ldots, a_n$ so the result follows again trivially by the induction hypothesis.* □

**Lemma C.2** $\forall s \in S\ [T_{\Omega,s} \neq \varnothing \Leftrightarrow s \rightarrow^+ \boldsymbol{*}]$

**Proof C.2** *($\Rightarrow$). Let $s \in S$ with $T_{\Omega,s} \neq \varnothing$. Pick any $t \in T_{\Omega,s}$ and proceed by structural induction on $t$.*
**Base case**. *[$t = c$]: Suppose $c : \rightarrow s'' \in F$ is a constant. Since $c \in T_{\Omega,s}$, we know $s'' \leqslant s$. If $s'' = s$, then*

*directly apply rule $s \to *$ generated by declaration $c : \to s'' \in F$. If $s'' < s$, we will have an additional rule $s \to s''$, which we can apply followed by $s \to *$. In either case, obtain $s \to^{+} *$.*

**Induction Step**. *$[t = f(t_1, \cdots, t_n)]$: Since $t = f(t_1, \cdots, t_n) \in T_{\Omega,s}$, we have $\exists f : s_1 \cdots s_k \to s'' \in F$ with $s'' \leqslant s$ where $t_i \in T_{\Omega,s_i}$ for $i \in k$. If $s'' = s$, then directly apply rule $s \to s_1 \boldsymbol{,} \cdots \boldsymbol{,} s_k$ generated by declaration $f : s_1 \cdots s_k \to s'' \in F$. Since $t_i \in T_{\Omega,s_i}$ for $i \in k$, we know that $T_{\Omega,s_i} \neq \varnothing$. Thus, by inductive hypothesis, obtain that $s_i \to^{+} *$ for $i \in k$. By transitivity, we have $s'' \to^{+} * \boldsymbol{,} \cdots \boldsymbol{,} *$. By idempotency, obtain $s'' \to^{+} *$. If $s'' < s$, we will have an additional rule $s \to s''$ we can apply followed by $s'' \to^{+} *$. In either case, obtain $s \to^{+} *$.*

*$(\Leftarrow)$. Suppose towards a contradiction the set $S' = \{s \in S \mid T_{\Omega,s} = \varnothing \wedge s \to^{+} *\}$ is non-empty. For each $s \in S'$ these is an $m(s) \in \mathbb{N}$ with $s \to^{m(s)} *$ and $m(s)$ smallest possible with that property. Pick $s_0 \in S'$ with $m(s_0)$ smallest among such $m(s)$. We now have two cases to consider: $m(s_0) = 1$ or $m(s_0) > 1$. Suppose $m(s_0) = 1$. Then $s_0 \to *$. But this can only happen if there is a $c : \to s_0 \in F$. But then $c \in T_{\Omega,s_0}$ and $T_{\Omega,s_0} \neq \varnothing$, a contradiction. Thus, assume $m(s_0) > 1$. Again, there are two possibilities: $s_0 \to s' \to^{m(s_0)-1} *$ or $s_0 \to s_1 \boldsymbol{,} \cdots \boldsymbol{,} s_k \to^{m(s_0)-1} *$. If $s_0 \to s' \to^{m(s_0)-1} *$, since $m(s_0)$ is smallest possible in $S'$, we must have $s' \notin S'$ and therefore $T_{\Omega,s'} \neq \varnothing$. But this rewrite can only occur if $s' < s_0$. Thus, $T_{\Omega,s'} \subseteq T_{\Omega,s_0}$, so that $T_{\Omega,s_0} \neq \varnothing$, a contradiction. If $s_0 \to s_1 \boldsymbol{,} \cdots \boldsymbol{,} s_k \to^{m(s_0)-1} *$, by Lemma C.1 for each $1 \leqslant i \leqslant k$ we have $s_i \to^{m_i} *$ for some $m_i \leqslant m(s_0) - 1$. Therefore, $T_{\Omega,s_i} \neq \varnothing$, $1 \leqslant i \leqslant k$. But the rewrite $s_0 \to s_1 \boldsymbol{,} \cdots \boldsymbol{,} s_k$ can only occur if there is an $f : s_1 \cdots s_k \to s_0 \in F$. But given any $t_i \in T_{\Omega,s_i}$, $1 \leqslant i \leqslant k$, we can construct $f(t_1, \cdots, t_k) \in T_{\Omega,s_0}$. Thus, $T_{\Omega,s_0} \neq \varnothing$, a contradiction.* □

The main result of this subsection is essentially an application of Lemma C.2.

**Lemma 4.1 (Non-emptiness Checking, p. 60)** *Given signature $\Omega = ((S, \leqslant), F)$ with $|S| + |F| \leqslant \aleph_0$, we define $\Omega_M = S \uplus \{*\} \uplus \{\_\boldsymbol{,}\_\}$ an unsorted signature. Then (a) we can deterministically construct a rewrite theory $\mathcal{R}_M$ over signature $\Omega_M$ such that $(\forall s \in S)\ T_{\Omega,s} \neq \varnothing \Leftrightarrow \mathcal{R}_M \vdash s \to^{+} *$ (b) checking $R_M \vdash s \to^{+} *$ is* decidable *(c) given axiom set $B$, $T_{\Omega/B,s} \neq \varnothing$ iff $T_{\Omega,s} \neq \varnothing$.*

**Proof C.3** *To prove (a), apply Lemma C.2. To prove (b), note that whenever the condition $|S| + |F| < \aleph_0$ holds, then $|\mathcal{P}(S)| + |R_M| < \aleph_0$ by construction. Thus, we have a finite number of states and rules in a* ground *rewrite theory, rendering the problem decidable by exhaustive state search. Finally, to prove (c), note that, $T_{\Omega/B,s}$ is just an equivalence relation over $T_{\Omega,s}$. Thus, $T_{\Omega/B,s} = \varnothing$ iff $T_{\Omega,s} = \varnothing$.*

As a result of this section, note that the set of sorts $S_{\supset\varnothing} \subseteq S$ is computable; thus, we obtain that $F_{\supset\varnothing}$ and $\Omega_{\supset\varnothing}$ are computable as well.

### C.1.2 Sort Finiteness

**Lemma C.3** *If $|S| + |F| < \aleph_0$ then $(\mathcal{R}_G, s)$ is non-terminating iff $|T_{\Omega,s}| = \aleph_0$*

**Proof C.4** *By construction of $R_G$, $|R_G| = |(<)| + |F| < |S|^2 + |F| < \aleph_0$. Viewing possible rewrite paths starting from $s$ as forming a tree, observe that the tree branches finitely, since each term has finite positions and possible rewrites. Suppose $(\mathcal{R}_G, s)$ is terminating. Then, by König's Lemma, the tree of rewrites must be finite and therefore there is a finite number of final states, so that $|T_{\Omega,s}| < \aleph_0$. Otherwise, if $(\mathcal{R}_G, s)$ is non-terminating, we have an infinite path $s \to_{R_G} t_1 \to_{R_G} t_2 \to_{R_G} \cdots\ t_n \to_{R_G} \cdots$. Since $|R_G| < \aleph_0$, $\exists R \subseteq R_G$ that repeats infinitely often. Since $R_G = R_{G,S} \uplus R_{G,C} \uplus R_{G,NC}$ and $R_{G,S} \uplus R_{G,C}$ terminates (because* *acyclicity/finiteness of* $(<)$ *and because only* $S$*-terms can be rewritten), we must have* $R \cap R_{G,NC} \neq \varnothing$*. But note that, if* $|t|$ *is the size of* $t$ *viewed as a tree, then if* $t \rightarrow_{R_{G,S} \uplus R_{G,C}} t'$*, we must have* $|t| = |t'|$*, whereas if* $t \rightarrow_{R_{G,NC}} t'$*, we must have* $|t| < |t'|$*, so that* $\{|t_i|\}_{i \in \mathbb{N}}$ *is a sequence such that* $|t_i| \rightarrow \infty$*. Also note that by the definition of* $R_G$*, all sorts* $s'$ *occurring as a subterm of* $t_i$ *belong to* $S_{\supset\varnothing} = \{s_1, \cdots, s_m\}$*, so that we can choose terms* $u_1 \in T_{\Omega,s_1}, \cdots, u_m \in T_{\Omega,s_m}$*. We can then regard* $S_{\supset\varnothing}$ *as a set of variables and view* $\sigma = \{s_1 \mapsto u_1, \cdots, s_m \mapsto u_n\}$ *as a substitution. But, by definition of* $R_G$*, this gives us an infinite sequence* $\{t_i\sigma\}_{i \in \mathbb{N}}$ *of terms where for each* $i \in \mathbb{N}$*,* $t_i\sigma \in T_{\Omega,s}$ *and* $|t_i\sigma| \geqslant |t_i|$*. Therefore,* $|t_i\sigma| \rightarrow \infty$*, and since* $T_{\Omega,s}$ *contains terms of unbounded size, we have* $|T_{\Omega,s}| = \aleph_0$*.* □

**Lemma 4.4 (p. 63)** $\forall n \in \mathbb{N}\, [[s \rightarrow^{n}_{R_G} t] \Leftrightarrow [\exists i, j \in \mathbb{N}\, [s \rightarrow^{i}_{R_{G,\bigstar}} t' \rightarrow^{j}_{R_{G,C}} t \wedge n = i + j]]]$

**Proof C.5** *To begin, recall* $R_G = R_{G,\bigstar} \uplus R_{G,C}$ *and note the following equivalence for* $s \in S_{\supset\varnothing}, n \in \mathbb{N}$*, and* $t \in T_\Omega$*:*

$$\begin{gathered} s \rightarrow^{n}_{R_G} t \\ \Leftrightarrow \end{gathered}$$

$$\begin{aligned} &\exists l_1, l_2, m_1, m_2 \in \mathbb{N}\ \exists t', t'', t''', t^{iv} \in T_\Omega \\ &\quad \{[(s \rightarrow^{l_1}_{R_{G,\bigstar}} t' \rightarrow^{l_2}_{R_{G,C}} t) \vee (s \rightarrow^{m_1}_{R_{G,\bigstar}} t'' \rightarrow_{R_{G,C}} t''' \rightarrow_{R_{G,\bigstar}} t^{iv} \rightarrow^{m_2}_{R_G} t)] \wedge \\ &\qquad l_1 + l_2 = m_1 + m_2 + 2 = n\} \end{aligned} \tag{C.2}$$

*That is, either all the applications of rules in* $R_{G,C}$ *occur at the end, or there is at least one such application* before *a rule in* $R_{G,\star}$*. Since the first case already fits the desired form, we need only consider the second case. Note all rules in* $R_G$ *have the form* $S \ni s \rightarrow t \in T_{\Omega \uplus S_\Omega}$*.* $R_{G,C}$ *rules in particular have the form* $s \rightarrow c$ *for* $c \in F$*. Thus, if a* $R_{G,C}$ *rule is applied to* $t[s]_p$ *at position* $p$*, a* $R_{G,\bigstar}$ *rule cannot later also be applied at* $p$*. Now suppose* $s \rightarrow^{m_1}_{R_{G,\bigstar}} t'' \rightarrow_{R_{G,C}} t''' \rightarrow_{R_{G,\bigstar}} t^{iv} \rightarrow^{m_2}_{R_G} t$*. Then,* $t'' = t''[s', s'']_{p,q}$ *with* $p, q$ *disjoint positions and:*

$$\begin{tikzcd} s \arrow[r, "R_{G,\bigstar}"', "*" near end] & t''[s', s''] \arrow[r, "R_{G,C}"'] \arrow[d, "R_{G,\bigstar}"'] & t''[c, s''] \arrow[d, "R_{G,\bigstar}"] \\ & t''[s', u] \arrow[r, "R_{G,C}"] & t''[c, u] \end{tikzcd} \tag{C.3}$$

*for any* $c \in C$ *and* $u \in T_{\Omega \uplus S_\Omega}$*, the diagram above commutes. We complete the proof by induction on* $m_2$*, the number of rewrites occurring after the first* $R_{G,C}$ *rule followed by a* $R_{G,\bigstar}$ *rule. Suppose* $m_2 = 0$*. Then we can commute the* $R_{G,\bigstar}$ *and* $R_{G,C}$ *arrows as above, to obtain a rewrite chain of the form* $s \rightarrow^{m_1+1}_{R_{G,\bigstar}} v \rightarrow_{R_{G,C}} t$*, for some* $v \in T_{\Omega \uplus S_\Omega}$*, as required. Now suppose* $m_2 > 0$*. Again, we commute the two arrows to obtain* $s \rightarrow^{m_1+1}_{R_{G,\bigstar}} v_1 \rightarrow_{R_{G,C}} v_2 \rightarrow^{m_2}_{R_G} t$*. We apply our induction hypothesis to obtain* $s \rightarrow^{m_1+1}_{R_{G,\bigstar}} v_1 \rightarrow^{k_1}_{R_{G,\bigstar}} v_3 \rightarrow^{k_2}_{R_{G,C}} t$ *with* $k_1 + k_2 = m_2$ *which is equivalent to* $s \rightarrow^{m_1+k_1+1}_{R_{G,\bigstar}} v_3 \rightarrow^{k_2}_{R_{G,C}} t$ *and* $m_1 + k_1 + k_2 + 1 = m_1 + m_2 + 1 = n$*, as required.* □

## C.2 AUXILIARY LEMMAS FOR SECTION 4.3.2

In these proofs, we always assume $(S^c, <^c)$ is a constructor sort refinement of $(S, <)$. In Lemma 4.7, we require two simple lemmas which are left as an exercise to the reader. Let $\Sigma$ be an arbitrary signature. Then (1) if $\Sigma$ is preregular and $f(t_1, \cdots, t_n) \in T_\Sigma$ then $ty_\Sigma(f(t_1, \cdots, t_n)) = ty_\Sigma(f, ls_\Sigma(t_1) \cdots ls_\Sigma(t_n))$ with $n \geqslant 0$ and (2) $t \in T_\Sigma \Leftrightarrow ty_\Sigma(t) \neq \varnothing$.

**Lemma 4.7 (Preregular Below, Semantic Version, p. 68)** *Suppose that $\Omega \prec \Sigma$. Then $\Omega$ and $\Sigma$ are preregular and $(\forall t \in T_\Sigma)\ t \in T_\Omega \Rightarrow ls_\Omega(t) = ls_\Sigma(t)$.*

**Proof C.6** *Since $\Omega$ is preregular, let $t = f(t_1, \cdots, t_n) \in T_{\Omega,s}$ with $s_i = ls_\Omega(t_i)$ for $1 \leqslant i \leqslant n$ and let $w = s_1 \cdots s_n$ (by abuse of notation, when $n = 0$, let $f \in T_{\Omega,s}$ be a constant and $w = nil$). By definition, since the types of each $t_i$ is minimal, any typing of $f$ must satisfy $f : w' \to s' \in ty_\Omega(f, w)$ with $s' \leqslant s$. Since $\Omega$ is preregular, $\exists s'' \in S$ with $s'' = ls_\Omega(t)$. This can happen iff $\exists f : w'' \to s'' \in ty_\Omega(f, w)$ such that $f : w'' \to s'' = min_<(ty_\Omega(f, w))$, $w \leqslant w''$, and $s'' \leqslant s$.*

*Now observe that since $\Omega \prec \Sigma$ and $t \in T_\Omega$, we have $ty_\Omega(f, w) \neq \varnothing$. Thus, $min_<(ty_\Sigma(f, w)) \in ty_\Omega(f, w)$. Since $ty_\Omega(f, w) \subseteq ty_\Sigma(f, w)$ then we must have $(\exists f : w'' \to s'' \in F)$ with $f : w'' \to s'' = min_<(ty_\Omega(f, w)) = min_<(ty_\Sigma(f, w))$. But then by the argument above, $ls_\Omega(t) = ls_\Sigma(t) = s''$.□*

**Lemma C.4** $(\forall t \in T_{\Omega^\downarrow}(X^c)/X^c)\ ty_{\Omega^\downarrow(X^c)}(t) = ty_{\Omega^\downarrow(X^\downarrow)}(t) = ty_{\Omega^\downarrow_\bullet(X^\downarrow)}(t)$

**Proof C.7** *The base case where $t = c \in T_{\Omega^\downarrow}(X^c)/X^c$, a constant, is trivial, so suppose $t = f(t_1, \cdots, t_n)$. There are two cases: either for each $1 \leqslant i \leqslant n$, we have $vars(t_i) \subseteq X^\downarrow$ or not. If not, $ty_{\Omega^\downarrow(X^c)}(t) = ty_{\Omega^\downarrow(X^\downarrow)}(t) = ty_{\Omega^\downarrow_\bullet(X^\downarrow)}(t) = \varnothing$ since these three signatures share the same non-variable operators $F^\downarrow_\Omega$ whose arity is contained in $(S^\downarrow)^*$. Otherwise, by induction hypothesis, for $1 \leqslant i \leqslant n$, we have $ty_{\Omega^\downarrow(X^c)}(t_i) = ty_{\Omega^\downarrow(X^\downarrow)}(t_i) = ty_{\Omega^\downarrow_\bullet(X^\downarrow)}(t_i)$, and since operators $F^\downarrow_\Omega$ are shared, we have $ty_{\Omega^\downarrow(X^c)}(t) = ty_{\Omega^\downarrow(X^\downarrow)}(t) = ty_{\Omega^\downarrow_\bullet(X^\downarrow)}(t)$.*
□

**Lemma C.5** $(\forall t \in T_{\Sigma^c}(X^c)/X^c)\ ty_{\Sigma^+(X^c)}(t) = ty_{\Sigma(X)}(t^\bullet)$

**Proof C.8** *The case where $t = c \in T_{\Sigma^c}(X^c)/T_{\Omega^\downarrow}(X^c)$, a constant, is trivial, so suppose $t = f(t_1, \cdots, t_n)$. By definition, $\exists f : s_1 \cdots s_n \to s \in F$ with $s_i \in S$, $t_i : s'_i$, and $s'_i \leqslant^c s_i$ for $1 \leqslant i \leqslant n$. But $(^\bullet) : (S, <^c) \to (S, <)$—also $(^\bullet) : \Sigma^+(X^c) \to \Sigma(X) \subseteq \Sigma^+(X^c)$—is a poset/signature morphism, so $s'^\bullet_i \leqslant s^\bullet_i = s_i$, $t^\bullet_i \in T_{\Sigma(X)}$, and $ty_{\Sigma^+(X^c)}(t) = ty_{\Sigma^+(X^c)}(t^\bullet)$. Also note $ty_{\Sigma^+(X^c)}|_{T_{\Sigma(X)}} = ty_{\Sigma(X)}$, since $f : s_1 \cdots s_n \to s \in F \cup X^c$ with $s_1 \cdots s_n \in S^*$ iff $f : s_1 \cdots s_n \to s \in F \cup X$. But $t^\bullet \in T_{\Sigma(X)}$, thus $ty_{\Sigma^+(X^c)}(t) = ty_{\Sigma^+(X^c)}(t^\bullet) = ty_{\Sigma(X)}(t^\bullet)$, as required.* □

**Lemma C.6** *$\Sigma = ((S, <), F)$ is sensible iff $\widehat{\Sigma} = ((\widehat{S}, \varnothing), \widehat{F})$ is sensible where we define $\widehat{F}$ as the set $f : [s_1] \cdots [s_n] \to [s_0] \in \widehat{F}$ iff $\exists f : s'_1 \cdots s'_n \to s'_0 \in F$ with $s'_i \in [s_i]$ for $0 \leqslant i \leqslant n$.*

**Proof C.9** *Given a tuple of sorts $w = s_1 \cdots s_n$, let $[w] = [s_1] \cdots [s_n]$. To see ($\Rightarrow$), suppose if $f : w \to s, f : w' \to s' \in F$ and $w \equiv_\leqslant w'$ then $s \equiv_\leqslant s'$. But note $w \equiv_\leqslant w'$ iff $w, w' \in [w]$ and $s \equiv_\leqslant s'$ iff $s, s' \in [s]$. To see ($\Leftarrow$), assume $f : [w] \to [s], f : [w'] \to [s'] \in \widehat{F}$ and $[w] \equiv_\leqslant [w']$ then $[s] \equiv_\leqslant [s']$. But note $[w] \equiv_\leqslant [w']$ iff $[w] = [w']$ and $[s] \equiv_\leqslant [s']$ iff $[s] = [s']$ since in $\widehat{\Sigma}$, $(\leqslant) = \varnothing$. Then assume towards a contradiction that $\exists f : w_1 \to s_1, f : w_2 \to s_2 \in F$ with $w_1 \equiv_\leqslant w_2$ and $s_1 \not\equiv_\leqslant s_2$. But then $w_1, w_2 \in [w_1] = [w_2]$ and $s_1 \in [s_1]$ and $s_2 \in [s_2]$ with $[s_1] \neq [s_2]$, a contradiction.*

# APPENDIX D OMITTED PROOFS FROM CHAPTER 5

### Proof of Lemma 5.1

**Proof D.1** *The show the* $\supseteq$ *part, let* $\alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u, v)$ *and* $\tau \in [(\mathit{vars}((u \mid \varphi)\alpha) \cup \mathit{vars}((v \mid \phi)\alpha)) \to T_\Omega]$ *be such that* $[u\alpha\tau!] \in \llbracket (u \mid \varphi \wedge \phi)\alpha \rrbracket$*. Then, for* $\rho = (\alpha\tau)|_Y$ *we have* $[u\alpha\tau!] \in \llbracket (u \mid \varphi)\rho \rrbracket \cap \llbracket (v \mid \phi)\rho \rrbracket$*, as desired.*

*To show the* $\subseteq$ *part, let* $[w] \in \llbracket (u \mid \varphi)\rho \rrbracket \cap \llbracket (v \mid \phi)\rho \rrbracket$ *for some* $\rho \in [Y \to T_\Omega]$*. Note that* $\mathit{vars}(u \mid \varphi) \cup \mathit{vars}(v \mid \phi) = Y \uplus \mathit{vars}((u \mid \varphi)\rho) \uplus \mathit{vars}((v \mid \phi)\rho)$*. Therefore, we have disjoint substitutions* $\tau \in [\mathit{vars}((u \mid \varphi)\rho) \to T_\Omega]$ $\gamma \in [\mathit{vars}((v \mid \phi)\rho) \to T_\Omega]$ *such that* $[w] = [(u(\rho \uplus \tau))!] = [(v(\rho \uplus \gamma))!]$ *and* $T_{\Sigma/E\cup B} \models (\varphi \wedge \phi)(\rho \uplus \tau \uplus \gamma)$*. But this means that there is a substitution* $\alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u, v)$ *and a ground substitution* $\delta \in [(\mathit{vars}((u \mid \varphi)\alpha) \cup \mathit{vars}((v \mid \phi)\alpha)) \to T_\Omega]$ *such that* $\rho \uplus \tau \uplus \gamma =_{E_\Omega \cup B_\Omega} (\alpha\delta)|_{\mathit{vars}(u|\varphi) \cup \mathit{vars}(v|\phi)}$*, and therefore, that* $[w] = [u\alpha\delta!] \in \llbracket (u \mid \varphi \wedge \phi)\alpha \rrbracket$*, as desired.* □

### Proof of Lemma 5.2

**Proof D.2** *We have to prove that if* $T_{\Sigma/E\cup B} \models \varphi \Rightarrow \bigvee_{(i,\beta) \in \mathrm{MATCH}(u,\ \{v_i\}_{i\in I}, Y)} \psi_i\beta$*, then for each* $\rho \in [Y \to T_\Omega]$ *we have* $\llbracket (u \mid \varphi)\rho \rrbracket \subseteq \llbracket (\bigvee_{i\in I} v_i \mid \psi_i)\rho \rrbracket$*. Indeed, if* $[w] \in \llbracket (u \mid \varphi)\rho \rrbracket$ *there is a ground substitution* $\tau \in [X \to T_\Omega]$ *such that* $[w] = [(u\rho\tau)!]$ *and* $T_{\Sigma/E\cup B} \models \varphi\rho\tau$*. But since* $T_\Omega \subseteq T_\Sigma(X)$*, we can view* $\rho\tau$ *as a composed substitution* $\rho\tau \in [X \to T_\Omega]$*, and therefore* $T_{\Sigma/E\cup B} \models \bigvee_{(i,\beta) \in \mathrm{MATCH}(u,\ \{v_i\}_{i\in I}, Y)} \psi_i\beta\rho\tau$*. That is, there is a pair* $(j, \gamma) \in \mathrm{MATCH}(u,\ \{v_i\}_{i\in I}, Y)$ *such that* $T_{\Sigma/E\cup B} \models \psi_j\gamma\rho\tau$*, and, since by construction,* $u =_{E_\Omega \cup B_\Omega} v_j\gamma$ *and* $\rho$ *and* $\gamma$ *have disjoint domains, using again the containment* $T_\Omega \subseteq T_\Sigma(X)$*, we have an identity of composed substitutions* $\gamma\rho\tau = \rho\gamma\tau$*, and therefore* $[w] = [(u\rho\tau)!] = [(v_j\gamma\rho\tau)!] = [(v_j\rho\gamma\tau)!]$ *with* $T_{\Sigma/E\cup B} \models \psi_j\rho\gamma\tau$*. Therefore,* $[w] \in \llbracket (v_j \mid \psi_j)\rho \rrbracket \subseteq \llbracket (\bigvee_{i\in I} v_i \mid \psi_i)\rho \rrbracket$*, as desired.* □

### Proof of Lemma 5.3

**Proof D.3** *First of all note that* $\mathit{vars}(\alpha(Y)) = (Y \backslash \mathit{dom}(\alpha)) \uplus \mathit{ran}(\alpha|_Y)$*. Let* $U_0 = U \backslash Y$ *and* $Z_0 = Z \backslash Y$*, so that* $U_0 \cap Z_0 = \varnothing$*. We then can derive equalities* $\mathit{vars}(\alpha(U)) = (U_0 \backslash \mathit{dom}(\alpha)) \uplus \mathit{ran}(\alpha) \uplus (Y \backslash \mathit{dom}(\alpha))$*, and* $\mathit{vars}(\alpha(Z)) = Z_0 \uplus \mathit{ran}(\alpha|_Y) \uplus (Y \backslash \mathit{dom}(\alpha))$*. Therefore, by the disjointness of* $U_0$*,* $Z_0$*, and* $\mathit{ran}(\alpha)$*, we get,* $\mathit{vars}(\alpha(U)) \cap \mathit{vars}(\alpha(Z)) = (Y \backslash \mathit{dom}(\alpha)) \uplus \mathit{ran}(\alpha|_Y) = \mathit{vars}(\alpha(Y))$*, as desired.* □

### Proof of Lemma 5.4

**Proof D.4** *Since* $\llbracket u \mid \varphi \rrbracket \subseteq \mathit{Term}_{\mathcal{R}}$ *and* $\llbracket T \rrbracket \subseteq \mathit{Term}_{\mathcal{R}}$*,* $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ *iff for each* $\rho \in [Y \to T_\Omega]$ *and each* $[w] \in \llbracket (u \mid \varphi)\rho \rrbracket$*, if* $[w] \in \llbracket T \rrbracket$ *then* $[w] \in \llbracket (\bigvee_{j\in J} v_j \mid \phi_j)\rho \rrbracket$*. But this is exactly what the* $T$*-consistency of* $u \mid \varphi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ *ensures.* □.

### Proof of Lemma 5.5.

**Proof D.5** *If* $[u] \to_{\mathcal{R}} [v]$ *corresponds to the topmost* $R, B$*-rewrite* $u \to_{R,B} u'$*, performed with a rewrite rule* $l \to r$ *if* $\phi \in R$ *and a ground substitution* $\sigma \in [Y \to T_\Sigma]$*, with* $Y$ *the rule's variables, and such that* $u =_{B_\Omega} l\sigma$*,* $u' = r\sigma$*, and* $[u'!] = [v]$*, this is also a rewrite with the rule* $l \to r'$ *if* $\phi \wedge \hat{\theta}$*, by extending* $\sigma$ *to the fresh variables* $X_P = \{x_p \mid p \in P\}$ *with the assignments* $x_p \mapsto (r\sigma)|_p$*, so that we have* $[u] \to_{\hat{\mathcal{R}}} [v]$*.*

*Conversely, if* $[u] \to_{\hat{\mathcal{R}}} [v]$ *corresponds to the topmost* $\hat{R}, B$*-rewrite* $u \to_{R,B} w$*, performed with rewrite rule* $l \to r'$ *if* $\phi \wedge \hat{\theta}$ *in* $\hat{R}$ *and ground substitution* $\rho \in [Y \uplus X_P \to T_\Sigma]$*, so that* $w = r'\rho$ *and* $[w!] = [v]$*, then we can perform a corresponding rewrite with rule* $l \to r$ *if* $\phi \in R$ *and substitution* $\rho|_Y$*, because* $T_{\Sigma/E\cup B} \models \phi\rho$*. Furthermore, since* $T_{\Sigma/E\cup B} \models \hat{\theta}\rho$*, we must have* $[w!] = [(r\rho)!] = [v]$*, so that* $[u] \to_{\mathcal{R}} [v]$*.* □

**Proof of Theorem** 5.1.

**Proof D.6** *A state $[\langle u_1,\ldots,u_n\rangle]_{B\Omega} \in \mathcal{C}_{\mathcal{R},State}$ is reachable from $[\![S_0]\!]$ iff $[[u_1,\ldots,u_n]]_{B\Omega}$ is reachable from $[\![S_0]\!]$ in $\mathcal{C}_{\mathcal{R}_{stop}}$. Therefore, $[\![P]\!]$ is an invariant of $(\mathcal{C}_{\mathcal{R},State},\rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$ iff $\mathcal{R}_{stop} \models^{\forall}_{[]} S_0 \rightarrow^{\circledast} [P]$.* □

**Proof of Theorem** 5.2.

**Proof D.7** *We need to show that (i) and (ii) in the theorem's statement hold iff for each $\rho \in [Y{\rightarrow}T_\Omega]$ we have: (i′) $[\![S_0\rho]\!] \subseteq [\![P\rho]\!]$, and (ii′) $Reach_{\mathcal{R}}([\![P\rho]\!]) = [\![P\rho]\!]$. Since (i) is equivalent to (i′) holding for each $\rho$, we just need to show that (ii) holds iff (ii′) holds for each $\rho$. But this follows easily from the earlier remarks explaining the implicit universal and existential quantification in reachability logic formulas, plus the two crucial observation that: (a) in the pattern formula $P \rightarrow^{\circledast} [P\sigma]$ the pattern predicate $[P\sigma]$ is a* postcondition, *and (b) in $\mathcal{R}_{stop}$ a* terminating *sequence from $[u_0] \in [\![P\rho]\!]$ always has the form:*

$$[u_0] \rightarrow_{\mathcal{R}} [u_1] \ldots [u_{n-1}] \rightarrow_{\mathcal{R}} [u_n] \rightarrow_{\mathcal{R}_{stop}} [[u_n]] \tag{D.1}$$

*for $n \geqslant 0$ (where, by convention, if $u_n = \langle u_{n_1},\ldots,u_{n_k}\rangle$, then $[u_n]$ abbreviates $[u_{n_1},\ldots,u_{n_k}]$), thus putting in one-to-one correspondence such sequences with elements $[u_n] \in Reach_{\mathcal{R}}([\![P\rho]\!])$ reachable from a $[u_0] \in [\![P\rho]\!]$, and, since $P \rightarrow^{\circledast} [P\sigma]$ holds on such a sequence, showing that $[[u_n]] \in [\![[P\sigma\rho]]\!] = [\![[P\rho\sigma]]\!] = [\![[P\rho]]\!]$ and therefore that $[u_n] \in [\![P\rho]\!]$, as desired.* □

**Proof of Theorem** 5.3.

**Proof D.8** *The proof is completely analogous to that of Theorem 5.2. We need to show that (i) and (ii) in the theorem's statement hold iff for each $\rho \in [Y{\rightarrow}T_\Omega]$ we have: (i′) $[\![S_0\rho]\!] \cap [\![Q\rho]\!] = \varnothing$, and (ii′) $Reach_{\mathcal{R}^{-1}}([\![Q\rho]\!]) = [\![Q\rho]\!]$. As before, (i) holds iff (i′) does for each $\rho$. The proof that (ii) is equivalent to (ii′) holding for each $\rho$ is entirely analogous to that in Theorem 5.2 and is left to the reader.* □

**Proof of Theorem** 5.4.

**Proof D.9** *We begin by introducing the following auxiliary notation*

***Definition D.1*** *Let $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ be a $T$-consistent reachability formula with parameters $Y$. By definition, $\mathcal{R} \models^{\forall,n}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ iff for each $[u_0] = [u\rho!] \in [\![u \mid \varphi]\!]$ and for each $T$-terminating sequence $[u_0] \rightarrow_{\mathcal{R}} [u_1] \rightarrow_{\mathcal{R}} \ldots \rightarrow_{\mathcal{R}} [u_m]$ with $m \leqslant n$, there exist $j$, $0 \leqslant j \leqslant m$, $\tau$ and $i$ such that $[u_j] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$. Note that, since $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ is $T$-consistent, $\mathcal{R} \models^{\forall,0}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ always holds.*

*With this notation, we state the following auxiliary lemma:*

***Lemma D.1*** *Let $[\mathcal{A},\ \mathcal{C}'] \vdash_T u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ be a closed goal with parameters $Y$ (and therefore $T$-consistent), derived by our inference system for $\mathcal{R}$ from some initial set of goals $[\mathcal{L},\ \mathcal{C}] \vdash_T \mathcal{C}$. Then, for each $n > 1$, if $\mathcal{R} \models^{\forall,n}_{T} \mathcal{A}$ and $\mathcal{R} \models^{\forall,n-1}_{T} \mathcal{C}'$, then $\mathcal{R} \models^{\forall,n}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$.*

***Proof D.10*** *We prove the lemma by contradiction. Assume it does not hold; let $n_{min}$ be the smallest $n$ for which the lemma does not hold for the closed goals derivable from the initial goals $[\mathcal{L},\ \mathcal{C}] \vdash_T \mathcal{C}$. Let $[\mathcal{A},\ \mathcal{C}'] \vdash_T u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ be a closed goal among these for which the lemma does not hold for $n_{min}$ and, among such closed goals, one having a closed proof tree $\mathcal{P}$ of the smallest possible size. Note that this means that: (i) the Lemma holds for any $n \leqslant n_{min}$ for each non-root closed subgoal appearing in the closed proof* *tree $\mathcal{P}$ (otherwise $\mathcal{P}$ would not be of smallest possible size); and (ii) $\mathcal{R} \not\models^{\forall, n_{min}} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, but (a) the Lemma's hypotheses hold for $n = n_{min}$; and (b) for any $n < n_{min}$ $\mathcal{R} \models^{\forall, n} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$. We then show that, for any $[u_0] = [u\rho!] \in \llbracket u \mid \varphi \rrbracket$ and for any $T$-terminating path, $[u_0] \to_{\mathcal{R}} [u_1] \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} [u_{n_{min}}]$ there exists a $k$, $0 \leqslant k \leqslant n_{min}$, $\tau$ and $i$ with $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$, so that, since $\mathcal{R} \models_T^{\forall, n} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ for any $n < n_{min}$, we get $\mathcal{R} \models_T^{\forall, n_{min}} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, contradicting the assumption $\mathcal{R} \not\models^{\forall, n_{min}} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ and completing the proof.*

*We distinguish the following cases, according to the proof rule applied to the root goal $[\mathcal{A},\ \mathcal{C}'] \vdash_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ in its closed proof tree:*

**Subsumption.** *Then $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ is a trivial formula, so that $\mathcal{R} \models^{\forall} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, and, a fortiori, $\mathcal{R} \models_T^{\forall, n_{min}} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$.*

**Step$^{\forall}$.** *Let $[\mathcal{A},\ \mathcal{C}'] \vdash_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ be a closed goal with a minimal closed proof tree $\mathcal{P}$ for which the lemma does not hold for $n_{min}$. First, notice that*

$$\varphi \Leftrightarrow (\varphi \wedge \bigvee_{(i,\beta) \in \text{MATCH}(u,\ \{v_i\}, Y)} \psi_i\beta) \vee \varphi'. \tag{D.2}$$

*Therefore, $\llbracket u \mid \varphi \rrbracket = \llbracket u \mid \varphi \wedge \bigvee_{(i,\beta) \in \text{MATCH}(u,\ \{v_i\}, Y)} \psi_i\beta) \rrbracket \cup \llbracket u \mid \varphi' \rrbracket$. But, since each $(i, \beta) \in \text{MATCH}(u,\ \{v_i\}, Y)$ we have $u =_{E_\Omega \cup B_\Omega} v_i\beta$, and for each ground $\Omega$-substitution $\gamma \uplus \tau$ with $dom(\gamma) = Y$ the goal's parameters, and for each $[w] = [(u(\gamma \uplus \tau))!] \in \llbracket u \mid \varphi \wedge \bigvee_{(i,\beta) \in \text{MATCH}(u,\ \{v_i\}, Y)} \psi_i\beta) \rrbracket$ there must be an $i$ such that $T_{\Sigma/E \cup B} \models \psi_i\beta(\gamma \uplus \tau)$ and $u =_{E_\Omega \cup B_\Omega} v_i\beta$, we must have $[w] \in \llbracket (v_i \mid \psi_i)\beta\gamma \rrbracket$, and, since $\gamma$ and $\beta$ have disjoint domains, so that $\gamma\beta = \beta\gamma$, a fortiori, $[w] \in \llbracket (\bigvee_i v_i \mid \psi_i)\gamma \rrbracket$. But this means that $\llbracket u \mid \varphi \wedge \bigvee_{(i,\beta) \in \text{MATCH}(u,\ \{v_i\}, Y)} \psi_i\beta) \rrbracket \subseteq \llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket$. Therefore, since we have: (i) $\llbracket u \mid \varphi \rrbracket = \llbracket u \mid \varphi \rrbracket \backslash (\llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket) \uplus (\llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket)$, (ii) $\llbracket u \mid \varphi \wedge \bigvee_{(i,\beta) \in \text{MATCH}(u,\ \{v_i\}, Y)} \psi_i\beta) \rrbracket \subseteq \llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket$, and (iii) $\llbracket u \mid \varphi \rrbracket = \llbracket u \mid \varphi \wedge \bigvee_{(i,\beta) \in \text{MATCH}(u,\ \{v_i\}, Y)} \psi_i\beta) \rrbracket \cup \llbracket u \mid \varphi' \rrbracket$, the set-theoretic equalities (i)–(iii) force the containment $\llbracket u \mid \varphi' \rrbracket \supseteq \llbracket u \mid \varphi \rrbracket \backslash (\llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket)$, giving us the desired over-approximation claimed in Fact (3) in the explanation of the* Step$^{\forall}$ *rule. Therefore, since, as pointed out in Fact (1) of the same explanation, any state in the set $\llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket$ automatically satisfies the formula $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, and $\mathcal{R} \models_T^{\forall, n_{min}-1} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, to reach the desired contradiction $\mathcal{R} \models_T^{\forall, n_{min}} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ it is enough to consider $T$-terminating paths of length $n_{min}$, $[u_0] \to_{\mathcal{R}} [u_1] \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} [u_{n_{min}}]$ with $[u_0] = [u\rho!] \in \llbracket u \mid \varphi \rrbracket \backslash (\llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket)$. Note that if such a path is going to satisfy $\mathcal{R} \models_T^{\forall, n_{min}} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ by means of some $k$, $0 \leqslant k \leqslant n_{min}$, with $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$, we must have $k \geqslant 1$. Therefore, it is enough to show that in the length $n_{min} - 1$ path $[u_1] \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} [u_{n_{min}}]$ there is a $k$, $1 \leqslant k \leqslant n_{min}$ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$. But, since we have the over-approximation $\llbracket u \mid \varphi' \rrbracket \supseteq \llbracket u \mid \varphi \rrbracket \backslash (\llbracket u \mid \varphi \rrbracket \cap_Y \llbracket \bigvee_i v_i \mid \psi_i \rrbracket)$, and, by the assumptions on the minimal proof tree $\mathcal{P}$, we have $\mathcal{R} \models_T^{\forall, n_{min}} (r_j \mid \varphi' \wedge \phi_j)\alpha \to^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha$ for each $(j, \alpha) \in \text{UNIFY}(u \mid \varphi',\ R)$, it will be enough to show that: (a) $[u_1] = [(r_j\alpha\delta)!] \in \llbracket (r_j \mid \varphi' \wedge \phi_j)\alpha \rrbracket$ for some $(j, \alpha) \in \text{UNIFY}(u \mid \varphi',\ R)$; (b) $\rho =_{E_\Omega \cup B_\Omega} (\alpha\delta)|_U$ for some ground substitution $\delta$, where $U = vars(u \mid \varphi')$ (and of course, thanks to the over-approximation, $T_{\Sigma/E \cup B} \models \varphi'\rho$), and (c) the parameters of $(r_j \mid \varphi' \wedge \phi_j)\alpha \to^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha$ are exactly $vars(\alpha(Y))$.*

*Indeed, let us prove (a)–(c) hold, and then show that there is a $k$, $1 \leqslant k \leqslant n_{min}$ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$. First of all, since $[u_0] \to_{\mathcal{R}} [u_1]$, there is an unforgetful rule $l_j \to r_j$ if $\phi_j$ in $R$ such that $[u_0] = [(u\rho)!] = [(l_j\gamma)!] \to_{\mathcal{R}} [(r_j\gamma)!] = [u_1]$, and $T_{\Sigma/E \cup B} \models \phi_j\gamma$. But since the variables of all sequents and those of $l_j \to r_j$ if $\phi_j$ are always assumed disjoint, this just means that $\rho \uplus \gamma$ is a*

$E_\Omega \cup B_\Omega$-unifier of $u = l_j$. Therefore, there is a $(j, \alpha) \in \textsc{unify}(u \mid \varphi',\ R)$ and a ground substitution $\delta$ such that $\rho \uplus \gamma$ is equal modulo $E_\Omega \cup B_\Omega$ to $\alpha\delta$, which proves (b). But since $\rho \uplus \gamma$ is equal modulo $E_\Omega \cup B_\Omega$ to $\alpha\delta$, $T_{\Sigma/E\cup B} \models \varphi'\rho$, and $T_{\Sigma/E\cup B} \models \phi_j\gamma$, we have $T_{\Sigma/E\cup B} \models (\varphi' \wedge \phi_j)\alpha\delta$, which proves (a). Now note that, by the variable disjointness between rules in $R$ and sequents, $(v_i \mid \psi_i)\alpha = (v_i \mid \psi_i)\alpha|_U = (v_i \mid \psi_i)\alpha|_Y$. Therefore, if $Z = vars(v_i \mid \psi_i)$, assuming without loss of generality that all variables in the range of $\alpha$ are fresh, we have $vars((v_i \mid \psi_i)\alpha) = Z\backslash Y \uplus vars(\alpha(Y))$. Furthermore, since $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ satisfies the invariant $vars(\psi_i) \subseteq vars(v_i) \cup vars(u \mid \varphi)$ for each $i$, and for each $(i, \beta) \in \textsc{match}(u,\ \{v_i\}, Y)$ $u\beta = u =_{E_\Omega \cup B_\Omega} v_i\beta$, and the equations $E_\Omega \cup B_\Omega$ are regular, we have $vars(\psi_i\beta) \subseteq vars(u \mid \varphi)$, and therefore $vars(u \mid \varphi) = vars(u \mid \varphi')$. But since $l_j \to r_j$ if $\phi_j$ is unforgetful, we have $vars(l_j) \subseteq vars(r_j) \cup vars(\phi_j) = W$, and therefore, by the freshness assumption on $\alpha$ and regularity of the equations $E_\Omega \cup B_\Omega$, $vars((r_j \mid \phi_j)\alpha) = vars(l_j\alpha) \uplus W\backslash vars(l_j) = vars(u\alpha) \uplus W\backslash vars(l_j)$. This then yields $vars((r_j \mid \varphi' \wedge \phi_j)\alpha) = vars((u \mid \varphi)\alpha) \uplus W\backslash vars(l_j)$. And since $Z$ and $W$ are disjoint sets of variables, again by the freshness of $\alpha$, we finally have, $vars((r_j \mid \varphi' \wedge \phi_j)\alpha) \cap vars((v_i \mid \psi_i)\alpha) = (Z\backslash Y \uplus vars(\alpha(Y))) \cap (vars((u \mid \varphi)\alpha) \uplus W\backslash vars(l_j)) = vars(\alpha(Y)) \cap (vars((u \mid \varphi)\alpha) = vars(\alpha(Y))$, proving (c).

Having proved (a)–(c) let us now finish this case by proving that in the path $[u_1] \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} [u_{n_{min}}]$ there is a $k$, $1 \leqslant k \leqslant n_{min}$, such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$, using the fact that $\mathcal{R} \models_T^{\forall, n_{min}} (r_j \mid \varphi' \wedge \phi_j)\alpha \to^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha$ for each $(j, \alpha) \in \textsc{unify}(u \mid \varphi',\ R)$. But we already know that $[u_1] = [(r_j\alpha\delta)!]$ and $T_{\Sigma/E\cup B} \models (\varphi' \wedge \phi_j)\alpha\delta$. But by $\mathcal{R} \models_T^{\forall, n_{min}} (r_j \mid \varphi' \wedge \phi_j)\alpha \to^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha$ and (c), this ensures that there is a $k$, $1 \leqslant k \leqslant n_{min}$ such that $[u_k] \in \llbracket ((v_i \mid \psi_i)\alpha)\delta|_{vars(\alpha(Y))} \rrbracket = \llbracket ((v_i \mid \psi_i)\alpha|_Y)\delta|_{vars(\alpha(Y))} \rrbracket = \llbracket ((v_i \mid \psi_i)(\alpha\delta)|_Y \rrbracket = \llbracket ((v_i \mid \psi_i)\rho|_Y \rrbracket$. Therefore, there is a ground substitution $\tau$ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$, as desired.

**Axiom.** Let $[\mathcal{A},\ \mathcal{C}'] \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ be a closed goal with parameters $Y$ and with a smallest possible closed proof tree $\mathcal{P}$ for which the lemma does not hold for $n_{min}$. In particular we know that $\mathcal{R} \models_T^{\forall, n_{min}} \mathcal{A}$. To reach the desired contradiction we need to show that for any $\rho \in [U{\to}T_\Omega]$ such that $[u_0] = [(u\rho)!] \in \llbracket u \mid \varphi \rrbracket$ and any $T$-terminating path, $[u_0] \to_{\mathcal{R}} [u_1] \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} [u_{n_{min}}]$ there exists a $k$, $0 \leqslant k \leqslant n_{min}$, an $i$, and a ground substitution $\tau$ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$.

Let $u' \mid \varphi' \to^{\circledast} \bigvee_j v'_j \mid \psi'_j$ with parameters $Y'$ such that $Y = vars(\alpha(Y')$ be the axiom in $\mathcal{A}$ used in the rule application. Since $u =_{E_\Omega \cup B_\Omega} u'\alpha$ and $u_0 =_{E_\Omega \cup B_\Omega} u\rho$ we have that $u_0 =_{E_\Omega \cup B_\Omega} u'\alpha\rho$. Further, since $T_{\Sigma/E\cup B} \models \varphi \Rightarrow \varphi'\alpha$ and $T_{\Sigma/E\cup B} \models \varphi\rho$, we have that $T_{\Sigma/E\cup B} \models \varphi'\alpha\rho$. Thus, $[u_0] = [(u'\alpha\rho)!] \in \llbracket u' \mid \varphi' \rrbracket$. Since $\mathcal{R} \models_T^{\forall, n_{min}} u' \mid \varphi' \to^{\circledast} \bigvee_j v'_j \mid \psi'_j$, there exists $j$ and $0 \leqslant k' \leqslant n_{min}$ such that $[u_{k'}] \in \llbracket (v'_j \mid \psi'_j)(\alpha\rho)|_{Y'} \rrbracket$. But by $(v'_j \mid \psi'_j)\alpha = (v'_j \mid \psi'_j)\alpha|_{Y'}$ and $Y = vars(\alpha(Y')$, we have $\llbracket (v'_j \mid \psi'_j)(\alpha\rho)|_{Y'} \rrbracket = \llbracket (v'_j\alpha \mid \psi'_j\alpha)\rho|_Y \rrbracket$. We then will be done of we show that:

1. $Y = vars(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi) \cap vars(\bigvee_i v_i \mid \psi_i)$, and

2. for any $\rho \in [U{\to}T_\Omega]$ such that $T_{\Sigma/E\cup B} \models \varphi\rho$, $\llbracket (v'_j\alpha \mid \psi'_j\alpha)\rho|_Y \rrbracket = \llbracket (v'_j\alpha \mid \psi'_j\alpha \wedge \varphi)\rho|_Y \rrbracket$.

Indeed, since $v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \to^{\circledast} \bigvee_i v_i \mid \psi_i$ is a closed subgoal in $\mathcal{P}$, we must have $\mathcal{R} \models_T^{\forall, n_{min}} v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \to^{\circledast} \bigvee_i v_i \mid \psi_i$. But, by (1), $v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \to^{\circledast} \bigvee_i v_i \mid \psi_i$ has parameters $Y$ and, by (2), $[u_{k'}] \in \llbracket (v'_j\alpha \mid \psi'_j\alpha \wedge \varphi)\rho|_Y \rrbracket$. But since the sequence $[u_{k'}] \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} [u_{n_{min}}]$ has length $n \leqslant n_{min}$, there exist a $k$, $k' \leqslant k \leqslant n_{min}$, an $i$, and a ground substitution $\tau$ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in \llbracket (v_i \mid \psi_i)\rho|_Y \rrbracket$, as desired.

To see (1), note that, by the parameter preservation assumption, we have $Y = vars(v'_j\alpha \mid \psi'_j\alpha) \cap vars(\bigvee_i v_i \mid \psi_i)$, so that $Y \subseteq vars(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi) \cap vars(\bigvee_i v_i \mid \psi_i)$. But since $vars(\varphi) = (vars(\varphi) \cap Y) \uplus (vars(\varphi) \cap U_0)$, where $U_0 = U\backslash Y$, if $x \in (vars(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi) \cap vars(\bigvee_i v_i \mid \psi_i))\backslash Y$, then we must have $x \in (vars(\varphi) \cap U_0)$, which is impossible, since $U_0 \cap vars(\bigvee_i v_i \mid \psi_i) = \emptyset$. To see (2), note that we always have $[\![(v'_j\alpha \mid \psi'_j\alpha)\rho|_Y]\!] \supseteq [\![(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi)\rho|_Y]\!]$. But since $(v'_j\alpha \mid \psi'_j\alpha)$ and $(vars(\varphi) \cap U_0)$ have disjoint variables, any $[(v'_j\alpha(\rho|_Y \uplus \theta))!] \in [\![(v'_j\alpha \mid \psi'_j\alpha)\rho|_Y]\!]$ has also the form $[(v'_j\alpha(\rho \uplus \theta))!]$, and since by assumption $E \cup B \models \varphi\rho$, we get $[(v'_j\alpha(\rho|_Y \uplus \theta))!] \in [\![(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi)\rho|_Y]\!]$, and therefore $[\![(v'_j\alpha \mid \psi'_j\alpha)\rho|_Y]\!] \subseteq [\![(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi)\rho|_Y]\!]$, as desired. This finishes the proof for the AXIOM case and for the lemma. □

Now we prove the main result (Theorem 5.4) using Lemma D.1. Indeed, assume by contradiction that the theorem does not hold. Then, there must be a closed goal $(u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i) \in \mathcal{C}$ such that $[\mathcal{L},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ is a closed subgoal derived by our inference system for $\mathcal{R}$, but $\mathcal{R} \not\models u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i \in \mathcal{C}$. Further, we can choose such a closed subgoal in $\mathcal{C}$ with $n_{min}$ the smallest possible natural number such that $\mathcal{R} \not\models^{\forall}_{n_{min}} \mathcal{C}$. By $T$-consistency of all goals in $\mathcal{C}$ we must have $n_{min} > 0$. Then, $\mathcal{R} \models^{\forall, n_{min}-1}_T \mathcal{C}$ and, since by hypothesis $\mathcal{R} \models^{\forall}_{n_{min}} \mathcal{L}$, we have, a fortiori, $\mathcal{R} \models^{\forall, n_{min}}_T \mathcal{L}$. Thus, by Lemma D.1, we have $\mathcal{R} \models^{\forall, n_{min}}_T \varphi \to^{\circledast} \bigvee_i \psi_i$. This contradicts the assumption $\mathcal{R} \not\models^{\forall}_{n_{min}} u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ and completes the proof. □

**Proof of Lemma 5.6**

**Proof D.11** Since $\varphi$ is semantically equivalent to $\psi \vee \phi$ we have $[\![u \mid \varphi]\!] = [\![u \mid \psi]\!] \cup [\![u \mid \phi]\!]$. The lemma then follows easily from Definition 5.5, using the parameter preservation condition. □

**Proof of Lemma 5.7**

**Proof D.12** Let $Y$ be the parameters in $[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} A$. We have two cases. (1) If $x{:}s \notin Y$, then $A\{x{:}s \mapsto u_i\} = A$, $1 \leqslant i \leqslant k$, and the result just follows from: (i) the parameters $Y$ being the same in $[\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} A$ and in its $k$ instances in the premise, and (ii) $[\![u \mid \varphi]\!] = \bigcup_{1\leqslant i\leqslant k}[\![(u \mid \varphi)\{x{:}s \mapsto u_i\}]\!]$. (2) If $x{:}s \in Y$, then the parameters of each $[\mathcal{A},\ \mathcal{C}] \vdash_T (u \mid \varphi)\{x{:}s \mapsto u_i\} \to^{\circledast} A\{x{:}s \mapsto u_i\}$ are $(Y - \{x{:}s\}) \cup vars(u_i)$. Observe that, by the definition of pattern set for $s$, $[Y \to T_\Omega] = \bigcup_{1\leqslant i\leqslant k}\{\{x{:}s \mapsto u_i\}\tau_i \mid \tau_i \in [(Y - \{x{:}s\}) \cup vars(u_i) \to T_\Omega]\}$. Therefore, $\mathcal{R} \models^{\forall}_T [\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} A$ iff $\forall\, \rho \in [Y \to T_\Omega]$ $\mathcal{R} \models^{\forall}_T ([\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} A)\rho$ iff $(\forall\, i,\ 1 \leqslant i \leqslant k)$ $(\forall\, \tau_i \in [(Y - \{x{:}s\}) \cup vars(u_i) \to T_\Omega])$ $\mathcal{R} \models^{\forall}_T ([\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \to^{\circledast} A)\{x{:}s \mapsto u_i\}\tau_i$ iff $\bigwedge_{1\leqslant i\leqslant k}[\mathcal{A},\ \mathcal{C}] \vdash_T (u \mid \varphi)\{x{:}s \mapsto u_i\} \to^{\circledast} A\{x{:}s \mapsto u_i\}$, as desired. □

**Proof of Lemma 5.8**

**Proof D.13** Suppose the SUBSTITUTION rule is applied to $u \mid \bigwedge_i w_i = w'_i \wedge \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ having parameters $Y$. Let $U = vars(u \mid \bigwedge_i w_i = w'_i \wedge \varphi)$, $U_0 = vars(\bigwedge_i w_i = w'_i)$, and $Z = vars(\bigvee_{j\in J} v_j \mid \phi_j)$. Then $Y = U \cap Z$. Let $W = Z \backslash Y$. Note that the following facts hold for each $\alpha \in Unif_{E_1 \cup B_1}(\bigwedge_i w_i = w'_i)$:

1. $[\![(u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha]\!] = [\![u\alpha \mid \varphi\alpha \wedge \widehat{\alpha}]\!]$.

2. $vars((u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha) \cap vars(\alpha(Z)) = vars(\alpha(Y)) = vars(u\alpha \mid \varphi\alpha \wedge \widehat{\alpha}) \cap vars(\alpha(Z))$.

To see (1), note that, since $\alpha \in Unif_{E_1 \cup B_1}(\bigwedge_i w_i = w'_i)$, $[\![(u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha]\!] = [\![(u \mid \varphi)\alpha]\!]$, and $[\![(u \mid \varphi)\alpha]\!] \supseteq [\![u\alpha \mid \varphi\alpha \wedge \widehat{\alpha}]\!]$. So we just need to show $[\![(u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha]\!] \subseteq [\![u\alpha \mid \varphi\alpha \wedge \widehat{\alpha}]\!]$. Indeed, suppose $\rho \in [(U \backslash U_0 \uplus ran(\alpha)) \to T_\Omega]$ is such that $[(u\alpha\rho)!] \in [\![(u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha]\!]$. Then $T_{\Sigma/E \cup B} \models \widehat{\alpha}(\rho \uplus (\alpha\rho)|_{U_0})$, and therefore, $[(u\alpha\rho)!] = [(u\alpha(\rho \uplus (\alpha\rho)|_{U_0}))!] \in [\![u\alpha \mid \varphi\alpha \wedge \widehat{\alpha}]\!]$, as desired.

To see (2), note that $vars((u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha) = (U \backslash U_0) \uplus ran(\alpha)$, $vars(u\alpha \mid \varphi\alpha \wedge \widehat{\alpha}) = U \uplus ran(\alpha)$, and $vars(\alpha(Z)) = W \uplus (Y \backslash U_0) \uplus vars(\alpha(Y \cap U_0))$. Therefore, $vars((u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha) \cap vars(\alpha(Z)) = vars(\alpha(Y)) = ((U \backslash U_0) \uplus ran(\alpha)) \cap W \uplus (Y \backslash U_0) \uplus vars(\alpha(Y \cap U_0)) = (Y \backslash U_0) \uplus vars(\alpha(Y \cap U_0)) = (U \uplus ran(\alpha)) \cap W \uplus (Y \backslash U_0) \uplus vars(\alpha(Y \cap U_0)) = vars(u\alpha \mid \varphi\alpha \wedge \widehat{\alpha})$, as desired.

*Now note that (1) and (2) yield the equivalence:*

$$\mathcal{R} \models^{\forall}_{T} (u \mid \bigwedge_{i} w_i = w'_i \wedge \varphi)\alpha \to^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\alpha \;\Leftrightarrow\; \mathcal{R} \models^{\forall}_{T} u\alpha \mid \varphi\alpha \wedge \widehat{\alpha} \to^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\alpha. \tag{D.3}$$

*The ($\Leftarrow$) implication in the Lemma's proof now follows immediately from the above equivalence and the following* Instance Lemma, *where $\psi$ is chosen to be the formula $\bigwedge_i w_i = w'_i \wedge \varphi$.*

**Lemma** 5.9 *(Instance Lemma)* Suppose $\mathcal{R} \models^{\forall}_{T} u \mid \psi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ with parameters $Y$, and let $\beta$ be a substitution whose domain $V$ is contained in $vars(u \mid \psi)$ and where the variables in $ran(\beta)$ are all fresh. Then $\mathcal{R} \models^{\forall}_{T} (u \mid \psi)\beta \to^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\beta$.

***Proof D.14*** *Let $U = vars(u \mid \psi)$. Note that, by the freshness assumption on $\beta$ and $V \subseteq U$, the formula $(u \mid \psi)\beta \to^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\beta$ has parameters $vars(\beta(Y))$. We then need to show that for each $\delta \in [(U\backslash V) \uplus ran(\beta) {\to} T_\Omega]$, $[u_0] = [(u\beta\delta)!] \in \llbracket (u \mid \psi)(\beta\delta)|_{vars(\beta(Y))} \rrbracket$ and $T$-terminating sequence $[u_0] \to_{\mathcal{R}} [u_1] \ldots [u_{n-1}] \to_{\mathcal{R}} [u_n]$ there is a $0 \leqslant k \leqslant n$ such that $[u_k] \in \llbracket (\bigvee_{j\in J} v_j \mid \phi_j)(\beta\delta)|_{vars(\beta(Y))} \rrbracket$. But $(\beta\delta)|_{vars(\beta(Y))} = \delta|_{Y\backslash V} \uplus (\beta\delta)|_{vars(\beta(Y\cap V))} = \delta|_{Y\backslash V} \uplus (\beta|_{Y\cap V})(\delta|_{vars(\beta(Y\cap V))}) = \delta|_{Y\backslash V} \uplus (\beta\delta)|_{Y\cap V} = (\beta\delta)|_{Y}$. Therefore, $[u_0] = [(u\beta\delta)!] \in \llbracket (u \mid \psi)(\beta\delta)|_{Y} \rrbracket$, so that, by the assumption $\mathcal{R} \models^{\forall}_{T} u \mid \psi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ with parameters $Y$, for the same $T$-terminating sequence there is a $0 \leqslant k \leqslant n$ such that $[u_k] \in \llbracket (\bigvee_{j\in J} v_j \mid \phi_j)(\beta\delta)|_{Y} \rrbracket = \llbracket (\bigvee_{j\in J} v_j \mid \phi_j)(\beta\delta)|_{vars(\beta(Y))} \rrbracket$, as desired.* □

*We now resume the proof of the ($\Rightarrow$) implication for Lemma 5.8. Recall that $U = vars(u \mid \bigwedge_i w_i = w'_i \wedge \varphi)$ and $Z = vars(\bigvee_{j\in J} v_j \mid \phi_j)$, so that $Y = U \cap Z$. We need to show that for each ground substitution $\gamma \in [U {\to} T_\Omega]$ such that $[u_0] = [(u\gamma)!] \in \llbracket u \mid \bigwedge_i w_i = w'_i \wedge \varphi \rrbracket$ and each $T$-terminating sequence $[u_0] \to_{\mathcal{R}} [u_1] \ldots [u_{n-1}] \to_{\mathcal{R}} [u_n]$ there is a $0 \leqslant k \leqslant n$, a $j \in J$, and a ground substitution $\tau \in [Z\backslash Y {\to} T_\Omega]$ such that $[u_k] = [(v_j(\rho|_Y \uplus \tau))!] \in \llbracket (\bigvee_{j\in J} v_j \mid \phi_j)\gamma|_Y \rrbracket$.*

*This can be shown as follows. Let $U_0 = vars(\bigwedge_i w_i = w'_i)$. Since $\gamma$ unifies $\bigwedge_i w_i = w'_i$, there must be a unifier $\alpha \in Unif_{E_1 \cup B_1}(\bigwedge_i w_i = w'_i)$ and a ground substitution $\delta \in [(U\backslash U_0) \uplus ran(\alpha) {\to} T_\Omega]$ such that $\gamma =_{E_\Omega \cup B_\Omega} \alpha\delta$. Therefore, by our earlier Fact (1), $[u_0] = [(u\gamma)!] = [(u\alpha\delta)!] \in \llbracket u \mid \bigwedge_i w_i = w'_i \wedge \varphi \wedge \widehat{\alpha} \rrbracket$. And, since we assume that $\mathcal{R} \models^{\forall}_{T} (u\alpha \mid \varphi\alpha \wedge \widehat{\alpha} \to^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\alpha$ (with parameters $vars(\alpha(Y))$ by Fact (2)), there is a $0 \leqslant k \leqslant n$, a $j \in J$, and a ground substitution $\tau \in [Z\backslash Y {\to} T_\Omega] = [(vars(\alpha(Z))\backslash vars(\alpha(Y))) {\to} T_\Omega]$ such that $[u_k] = [(v_j\alpha(\delta|_{vars(\alpha(Y))} \uplus \tau))!] \in \llbracket (\bigvee_{j\in J} v_j \mid \phi_j)\alpha\delta|_{vars(\alpha(Y))} \rrbracket$. But since (i) $(\bigvee_{j\in J} v_j \mid \phi_j)\alpha = (\bigvee_{j\in J} v_j \mid \phi_j)\alpha|_Y$, and (ii) $\alpha|_Y \delta|_{vars(\alpha(Y))} = (\alpha\delta)|_Y =_{E_\Omega \cup B_\Omega} \gamma|_Y$, we have $[u_k] = [(v_j(\gamma|_Y \uplus \tau))!] \in \llbracket (\bigvee_{j\in J} v_j \mid \phi_j)\gamma|_Y \rrbracket$, as desired.* □

# APPENDIX E OMITTED PROOFS FROM CHAPTER 6

## E.1 INTRODUCTION

In Appendix B, we saw several techniques for proving ground convergence of standard rewrite theories $\mathcal{R} = (\Sigma, B, R)$ where rules $R$ may be conditional. In this Appendix, we utilize these techniques to prove the ground convergence and sufficient completeness of the standard rewrite theory underlying our Maude system and property specification of IBOS. Since the specification is quite large, we believe that writing down the proof in a fully formalized way would overwhelm the reader; instead, we will illustrate the key practical concerns used when applying our hierarchical proof system in Table B.1, so that the reader may obtain a clear intuition. Before proceeding, it is necessary to Chapter 6 which gives an overview of our IBOS specification from a high level.

In particular, recall from Def. B.15 that for ground confluence proofs at each level, the initial set of starting goals is the set of all most general critical pairs between the rules in theory level $R_i$ and between rules in $R_i$ and rules in theory level $R_{i-1}$ where $i \neq 0$. For complex functions whose datatypes are free modulo associativity and/or commutativity, the number of most general critical pairs may be quite large. In our IBOS property specification, some function symbols have hundreds of critical pairs, which means that writing down the fully formalized proof would require closing hundreds of proof trees—*and that is only for a single function*. For such a large number of proof goals, automated support becomes essential. For this reason, large portions of the proof have been entirely automated in Maude, either as entirely new tools or extensions of previous tools, e.g., the Maude Church-Rosser Checker [45].

Our outline for this section is as follows: (1) we show the hierarchical dependencies between functions and outline our general proof strategy for proving ground convergence and sufficient completeness; (2) we illustrate how our hierarhical proof operates in a few representative examples.

For any omitted proof details, we refer the reader to our IBOS case study repository avaiable at https://github.com/sskeirik/ibos-case-study which contains the Maude source code for our automated proofs as well.

## E.2 IBOS CONVERGENCE/SUFFICIENT COMPLETENESS PROOF STRATEGY

Our Maude system specification for IBOS has 53 sorts, 80 constructors, and uses one defined symbol. Our Maude invariant property specification for IBOS defines another 31 sorts, 34 constructors (some are subsort-overloadings of previous constructors) and another 47 defined symbols.

Both specifications make extensive use of subsort-overloading, which greatly helps with nailing down precisely the input and output of functions. One may wonder if so many sorts are useful. In fact, subsort-overloading constructors can be understood as adding additional kind-level predicates to our specification, but with the advantage that Maude's built-in and highly efficient unification algorithm respects the subsort relation.

Since we will verify ground convergence and sufficient completeness hierarchically, we require a definitional order (Def. B.11) upon which to construct our theory telescope. Such an order is constructed from the non-constructor call graph (Def. B.10). We present a simplified non-constructor call graph in Figures E.1 and E.2—rendered by the versatile graphviz program—where (i) a symbol's dependencies are its leftward edges;

(ii) all transitive and self-edges have been removed; (iii) nodes that did not fit in the four columns of Figure E.1 appear on second column of Figure E.2; (iv) all defined symbols that have no edges are omitted; (v) the boolean functions **and** and **not** are also omitted due to excessive edges. Based on our layout, note that defined symbols in the same column cannot depend on each other. Thus, to construct a definitional order it is sufficient to first add all omitted symbols and then do a left-to-right, top-to-bottom traversal of symbols in Figures E.1 and E.2.

Thus, our general strategy is:

1. use standard, non-hierarchical methods to prove termination and sort-decreasingness;
2. for each theory $R_{i\neq 0}$ in our inductive telescope:
   (a) prove sufficient completeness and ground local confluence of $\mathcal{R}_i$ assuming sufficient completeness and ground convergence of $R_{i-1}$;
   (b) if any symbol $f \in \Delta_i$ is annotated with *fvp-constraints*$(f, \mathcal{R}) \neq \emptyset$, check obligations in Lemma B.1 to increase set of variant solvable equalities.

For termination checking we rely on recursive path orderings as mentioned in Sec. B.2 and mechanized using the Maude Termination Assistant [140]. On the other hand, since none of the defined symbols in the IBOS specification have subsort-overloading, the sort decreasingness check becomes trivial—it is sufficient to *only* consider the identity sort specialization. Additionally, in our IBOS system and property specification, all constructors are free modulo associativity, commutativity, and/or identity axioms. Thus, the base case of our hierarchical proof system where any rewrite rules in the constructor subtheory must be proved convergent has zero proof obligations and therefore holds trivially.

## E.3 IBOS CONVERGENCE/SUFFICIENT COMPLETENESS PROOF EXAMPLES

As an example of the kinds of proofs and proof obligations that we generate, consider the predicate `in-conf?` : *ProcessId Configuration* → *Bool*, which checks whether a process with identifier *ProcessId* exists in *Configuration*. We provide a partial specification of the predicate in Figure E.3 in Maude:

```
1  op in-conf? : ProcessId Configuration -> Bool [metadata "116 true"] .
2  eq in-conf?(P,  < P  | A > C) = true [variant] .
3 ceq in-conf?(P,  < P’ | A > C) = in-conf?(P,C) if P ~p P’ = false .
4  eq in-conf?(P,  none)         = false .
5 ceq in-conf?(NP, C)            = false if fresh-np-id?(NP,C) = true .
6 ceq in-conf?(WP, C)            = false if fresh-wp-id?(WP,C) = true .
```

Figure E.3: Partial Specification of `in-conf?`

where variables `P`,`P’` have sort *ProcessId*, `C` has sort *Configuration*, `A` has sort *AttributeSet*, and `NP` (resp. `WP`) has sort *NetProcessId* (resp. *WebProcessId*). Line 1 of Figure E.3 is the Maude operator declaration; the operator metadata encodes two pieces of information: (a) the natural number encodes the operator's precedence in the RPO; (b) the constant `true` is only element in the set *fvp-constraints*(`in-conf?`, `IBOS`). By examining *fvp-constraints*(`in-conf?`, `IBOS`), we see that equalities of the form `in-conf?(P,C)` = `true`

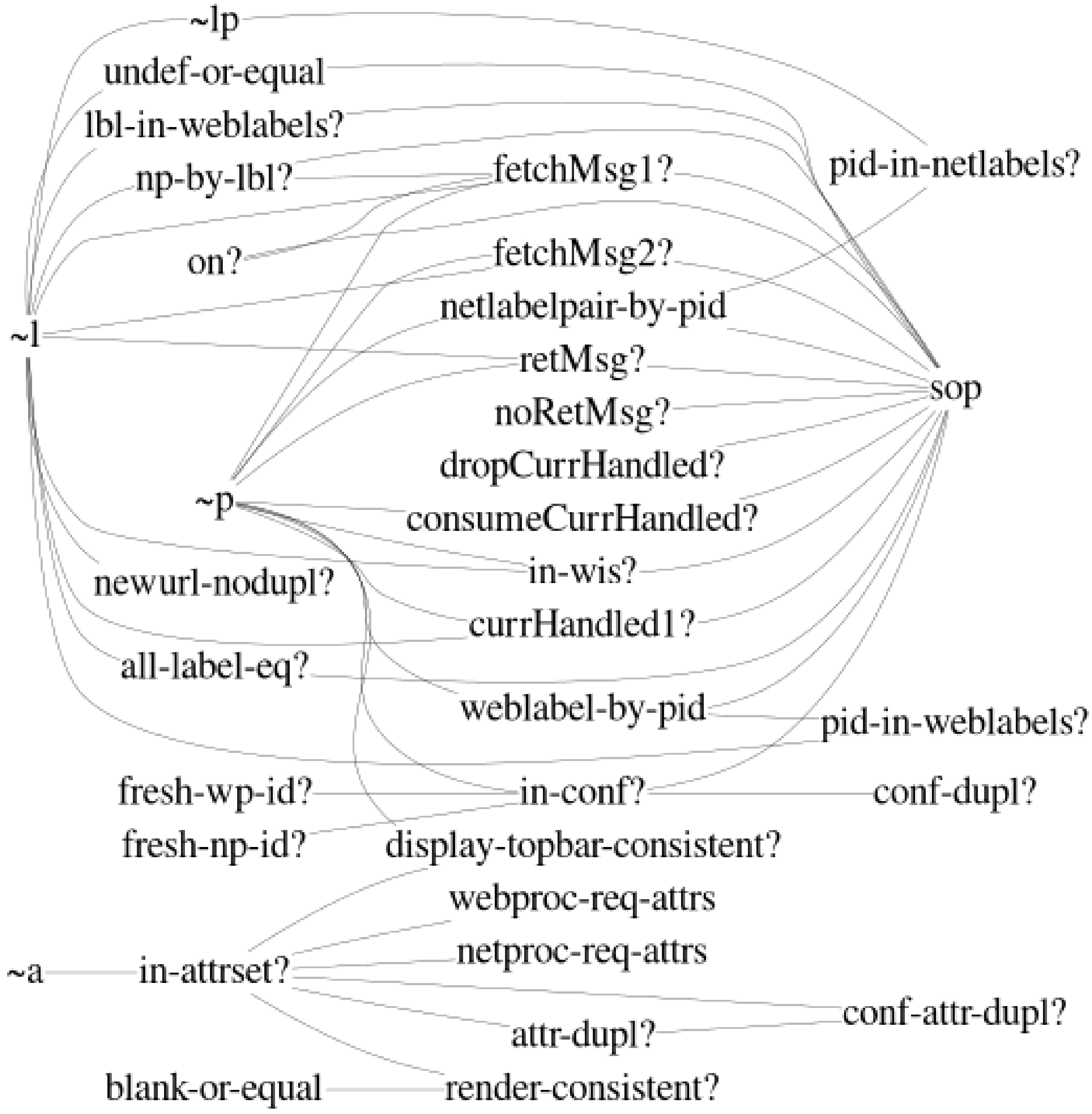


Figure E.1: IBOS Defined Symbol Dependencies

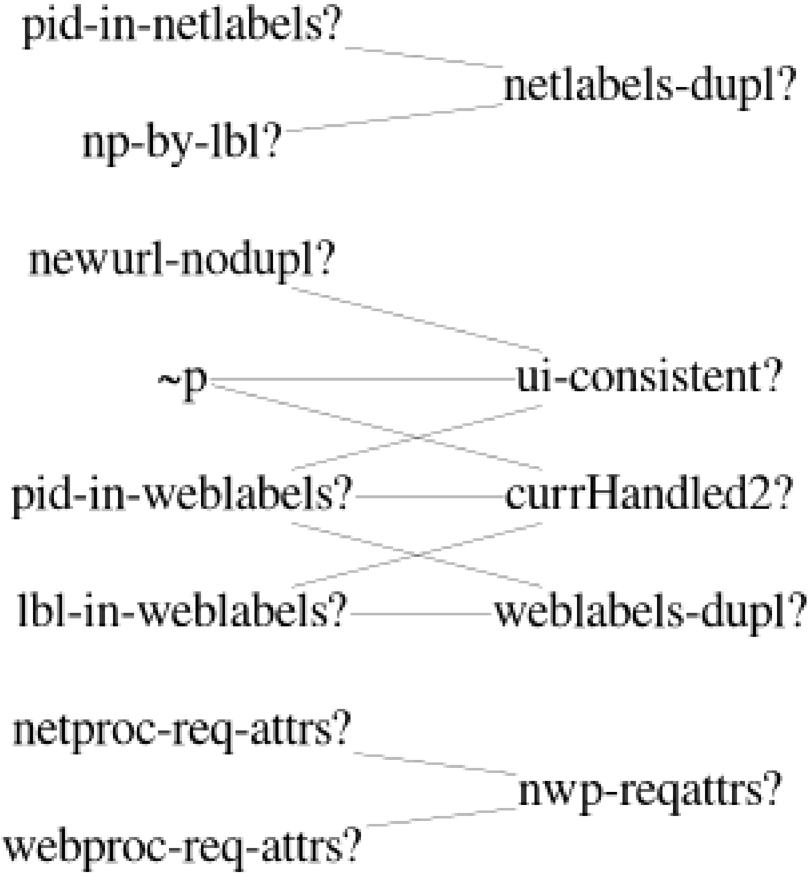


Figure E.2: Other IBOS Defined Symbol Dependencies

should be variant sovleable using *only* the equation on line 2—which follows by checking the conditions specified in Lemma B.1. Our intent that this equation is part of an FVP specification is indicated by adding the `variant` keyword to the equation's attribute set.

The intuition behind the definition of this predicate on lines 2–4 is the following: if identifier `P` is equal to the identifier of some process `< P | A >` in the configuration, we are done; otherwise, if `ProcessId` equality predicate `~p` returns `false`, throw away the current process and keep searching the remainder of the configuration; finally, if the configuration is `none`, process identifier `P` cannot appear in the configuration. The equations in lines 5–6 are useful lemmas for inductive theorem proving purposes which state, if the process identifier `NP` (resp. `WP`) is provably fresh (greater than all process identifers) in a configuration, it *cannot* appear in that configuration.

At this point, according to the strategy outlined above, we first assume that operations `~p`, `fresh-np-id?`, and `fresh-wp-id?` are sufficiently complete and ground convergent. Under this assumption, we then prove the sufficient completeness and the ground local confluence of the specification of `in-conf?`. Finally, we prove the conditions of Lemma B.1 are satisfied.

### E.3.1 Example Sufficient Completeness Proof

We first prove sufficient completeness. Recall from Def. B.15, the initial proof sequent in the sufficient completeness proof of $\mathcal{R}_i$ for each $f : s_1 \cdots s_n \to s \in \Delta_i$ is the constrained term $(f(x_n : s_1 \cdots x_n : s_n) \mid \top)$. In our case, the sole sequent is the constrained term is $(\texttt{in-conf?(P,C)} \mid \top)$. The entire proof is shown in Figure E.4, where we simplify notation by (a) displaying terms with the constraint $\top$ as unconstrained; (b) not repeating identical parts of sequents in Split or Case Analysis subproofs.

$$P_1 = \left\{ \textsc{Split}\ \dfrac{\textsc{Substitution}\ \dfrac{\textsc{Closure}\ \dfrac{}{\texttt{in-conf?(P,< P | A > C)}}}{\texttt{...} \mid \texttt{P} \sim p\ \texttt{P'} = \texttt{true}} \qquad \dfrac{}{\texttt{...} \mid \texttt{P} \sim p\ \texttt{P'} = \texttt{false}}\ \textsc{Closure}}{\texttt{in-conf?(P,< P' | A > C)}} \right.$$

$$\textsc{Case Analysis}\ \dfrac{\textsc{Closure}\ \dfrac{}{\texttt{in-conf?(P,none)}} \qquad \dfrac{P_1}{\texttt{in-conf?(P,< P' | A > C)}}}{\texttt{in-conf?(P,C)}}$$

Figure E.4: Sufficient Completeness Proof for `in-conf?`

Recall that the applications of the Case Analysis, Split, and Substitution rules all have non-trivial side-conditions that must be checked. For the Case Analysis rule, the completeness of the case analysis on variable `C` of sort *Configuration* follows from the fact that a configuration is syntactically just a multiset of processes. By definition, a multiset is empty or contains an element (process). For the Split rule, completeness follows from two facts: (a) symbol $\sim p$ is assumed to be sufficiently complete; (b) `true` and `false` is a coverset for sort *Bool*. Together, this implies that $\texttt{P} \sim p\ \texttt{P'} = \texttt{true} \vee \texttt{P} \sim p\ \texttt{P'} = \texttt{false} \Leftrightarrow \top$. Finally, the soundness of the Substitution step follows from the fact that, in the constraint $\texttt{P} \sim p\ \texttt{P'} = \texttt{true}$ the symbol $\sim p$ is an equality predicate. Thus, the most general unifier is naturally one where both process identifiers are made equal.

Note that the addition of the inductive lemmas on lines 5–6 in Figure E.3 does not affect sufficient completeness whatsoever; however, these lemmas do affect the confluence proof.

### E.3.2 Example Ground Local Confluence Proof

We must prove ground local confluence of `in-conf?`. Recall from Def. B.15, the initial proof sequents in the ground local confluence proof of $\mathcal{R}_i$ for each $f \in \Delta_i$ are the most general critical pairs among rules in $\mathcal{R}_{\Delta_i}$ and between any rules in $\mathcal{R}_{\Delta_i}$ and $\mathcal{R}_{i-1}$. In this case, the optimization from Def. B.15 applies because the symbol `in-conf?` has no axioms and is not subsort-overloaded, so no overlapping rules cannot appear in $R_{i-1}$. This implies that $MCP(\mathcal{R}_{i-1}, \mathcal{R}_{\Delta_i}) = \varnothing$. Thus, it is sufficient to consider the critical pairs $MCP(\mathcal{R}_{\Delta_i}, \mathcal{R}_{\Delta_i})$.

For the purposes of generating critical pairs, let us denote the equations on lines 2–6 in Figure E.3 as $E_2$ through $E_6$. Firstly, let us make some simplifying observations:

1. each left-hand side can only meaningfully unify at the topmost position;

2. thus, it is sufficient to consider critical pairs between unordered pairs of equations;

3. for equations $E_4$, $E_5$, and $E_6$, the self critical pairs are all trivial, because for each equation left-hand side, the most general self-unifier is the identity substitution;

4. equation $E_4$ cannot unify with equations $E_2$ and $E_3$;

5. the critical pairs generated by unifiers among $(E_2, E_2)$ $(E_4, E_5)$ $(E_4, E_6)$ $(E_5, E_6)$ are all either $(\texttt{true} \downarrow \texttt{true} \mid \phi)$ or $(\texttt{false} \downarrow \texttt{false} \mid \phi)$ and thus trivially joinable.

Thus, we are left with five pairs of equations with viable, non-trival left-hand side unifiers that produce non-trivial critical pairs: $(E_3, E_3)$ $(E_2, E_3)$ $(E_2, E_5)$ $(E_2, E_6)$ $(E_3, E_5)$ $(E_3, E_6)$.

We first tackle the critical pair sequents generated by $(E_2, E_5)$ and $(E_2, E_6)$, i.e., $MCP(\{E_2\}, \{E_5, E_6\})$.

$$\{(\texttt{true} \downarrow \texttt{false} \mid \texttt{fresh-np-id?(NP,< NP | A > C)} = \texttt{true}) \tag{E.1}$$

$$(\texttt{true} \downarrow \texttt{false} \mid \texttt{fresh-wp-id?(WP,< WP | A > C)} = \texttt{true})\} \tag{E.2}$$

Here, the condition for both pairs is unsatisfiable, since the instantion of the function `fresh-np-id?` (resp. `fresh-wp-id?`) is transformed to `false` by an application of the Simplify rule, leading to the unsatisfiable equality `false` = `true`.

We next tackle the critical pairs generated by equation pair $(E_3, E_3)$, i.e., $MCP(\{E_3\}, \{E_3\})$:

$$\{(\texttt{in-conf?(P,C)} \downarrow \texttt{in-conf?(P,C)} \mid \texttt{P} \sim p\ \texttt{P'} = \texttt{false}) \tag{E.3}$$

$$\begin{aligned}(\texttt{in-conf?(P,< P1 | A1 > C)} \downarrow \texttt{in-conf?(P,< P2 | A2 > C)} \mid \\ \texttt{P} \sim p\ \texttt{P1} = \texttt{false} \wedge \texttt{P} \sim p\ \texttt{P2} = \texttt{false})\}\end{aligned} \tag{E.4}$$

Here the first critical pair is trivially joinable. The second is joinable by Context Joinability; when the constraints are added to the specification as rewrite rules as follows:

$$\begin{aligned}\mathcal{R}_i \uplus \{\overline{\texttt{P}} \sim p\ \overline{\texttt{P1}} = \texttt{false}, \overline{\texttt{P}} \sim p\ \overline{\texttt{P2}} = \texttt{false}\} \vdash \\ \texttt{in-conf?(}\overline{\texttt{P}}\texttt{,< }\overline{\texttt{P1}}\texttt{ | A1 > C)} \downarrow \texttt{in-conf?(}\overline{\texttt{P}}\texttt{,< }\overline{\texttt{P2}}\texttt{ | A2 > C)}\end{aligned} \tag{E.5}$$

which rewrites by two applications of $E_3$ to:

$$\mathcal{R}_i \uplus \{\overline{\mathtt{P}} \sim\!p\ \overline{\mathtt{P1}} = \mathtt{false}, \overline{\mathtt{P}} \sim\!p\ \overline{\mathtt{P2}} = \mathtt{false}\} \vdash \\ \mathtt{in\text{-}conf?(}\overline{\mathtt{P}}\mathtt{,C)} \downarrow \mathtt{in\text{-}conf?(}\overline{\mathtt{P}}\mathtt{,C)}. \tag{E.6}$$

We next consider the proof sequents from critical pairs in $MCP(\{E_2\}, \{E_3\})$:

$$\{(\mathtt{true} \downarrow \mathtt{in\text{-}conf?(P,C)} \mid \mathtt{P} \sim\!p\ \mathtt{P} = \mathtt{false}), \tag{E.7}$$

$$(\mathtt{true} \downarrow \mathtt{in\text{-}conf?(P,<\ P\ \mid\ A2\ >\ C)} \mid \mathtt{P} \sim\!p\ \mathtt{P'} = \mathtt{false})\} \tag{E.8}$$

The first one is solvable by Simplify followed by Unfeasibility; the second is solvable by Context Joinability and a single application of $E_2$.

Our final case to consider are the critical pairs $MCP(\{E_3\}, \{E_5, E_6\})$:

$$\{(\mathtt{false} \downarrow \mathtt{in\text{-}conf?(NP,C)} \mid \mathtt{NP} \sim\!p\ \mathtt{P'} = \mathtt{false} \wedge \\ \mathtt{fresh\text{-}np\text{-}id?(NP,<\ P'\ \mid\ A\ >\ C)} = \mathtt{true}), \tag{E.9}$$

$$(\mathtt{false} \downarrow \mathtt{in\text{-}conf?(WP,C)} \mid \mathtt{WP} \sim\!p\ \mathtt{P'} = \mathtt{false} \wedge \\ \mathtt{fresh\text{-}wp\text{-}id?(WP,<\ P'\ \mid\ A\ >\ C)} = \mathtt{true})\} \tag{E.10}$$

In Figure E.5, we show the proof for the first critical pair; the second follows analogously. Since this example is quite complex, a few observations are in order to explain the proof:

1. the function `fresh-np-id?` only checks if its *NetProcessId* argument is fresh with respect to other processes with a *NetProcessId* identifier;

2. In proof subtree $P_2$, we split on the FVP less-than-or-equal predicate `<=` for natural numbers in order to enable the `fresh-np-id?` predicate to rewrite;

3. in proof subtree $P_4$ the variable `Z` is a non-zero natural number.

### E.3.3 Example Application of Variant Solveability Lemma

Finally, we apply Lemma B.1 to the `in-conf?` predicate. Firstly, observe that `in-conf?` is essentially tail-recursive. Secondly, since we have already proved confluence of `in-conf?`, observe that the righthand side of final rules of `in-conf?` are ground, so that sets $R^{+}_{\mathtt{in\text{-}conf?}} = \{E_2\}$ and $R^{-}_{\mathtt{in\text{-}conf?}} = \{E_4, E_5, E_6\}$ immediately satisfy condition 1 of Lemma B.1. Thus, we need only verify condition 2 of Lemma B.1 on the rule set $R_{rec,\mathtt{in\text{-}conf?}} = \{E_3\}$. For convenience, we recall the rules in question below:

$$R^{+}_{\mathtt{in\text{-}conf?}} = \{\mathtt{eq\ in\text{-}conf?(P2,\ <\ P2\ \mid\ A2\ >\ C2)\ =\ true\ [variant]\ .}\} \tag{E.11}$$

$$R_{rec,\mathtt{in\text{-}conf?}} = \{\mathtt{ceq\ in\text{-}conf?(P,\ <\ P'\ \mid\ A\ >\ C)\ =} \\ \mathtt{in\text{-}conf?(P,C)\ if\ P\ \sim p\ P'\ =\ false\ .}\} \tag{E.12}$$

To check that the condition is satisfied, we first compute the most general unifiers between the lefthand side of $E_2$ and the righthand side of $E_3$:

$$\mathit{Unif}_{B_\Omega}(lhs(E_2), rhs(E_3)) = \{\mathtt{P} \mapsto \mathtt{P2}, \mathtt{C} \mapsto \mathtt{<\ P2\ \mid\ A2\ >\ C2}\}. \tag{E.13}$$

$$P_4 = \left\{ \textsc{Sub.}\ \dfrac{\textsc{Simplify}\ \dfrac{\textsc{Con. Join.}\ \dfrac{}{(\texttt{false} \downarrow \texttt{in-conf?(network(M + Z),C)} \mid \texttt{fresh-np-id?(network(M + Z),C)} = \texttt{true})}}{(\texttt{false} \downarrow \texttt{in-conf?(network(M + Z),C)} \mid \texttt{fresh-np-id?(network(M + Z),< network(M) | A > C)} = \texttt{true})}}{(\texttt{false} \downarrow \texttt{in-conf?(network(N),C)} \mid \texttt{fresh-np-id?(network(N),< network(M) | A > C)} = \texttt{true} \wedge \texttt{N <= M} = \texttt{false})} \right.$$

$$P_3 = \left\{ \textsc{Sub.}\ \dfrac{\textsc{Simplify}\ \dfrac{\textsc{Unfeasibility}\ \dfrac{}{(\texttt{false} \downarrow \texttt{in-conf?(network(N),C)} \mid \texttt{false} = \texttt{true})}}{(\texttt{false} \downarrow \texttt{in-conf?(network(N),C)} \mid \texttt{fresh-np-id?(network(N),< network(M + N) | A > C)} = \texttt{true})}}{(\texttt{false} \downarrow \texttt{in-conf?(network(N),C)} \mid \texttt{fresh-np-id?(network(N),< network(M) | A > C)} = \texttt{true} \wedge \texttt{N <= M} = \texttt{true})} \right.$$

$$P_2 = \left\{ \textsc{Case}\ \dfrac{\textsc{Split}\ \dfrac{\dfrac{P_3}{\ldots \mid \ldots \wedge \texttt{N <= M} = \texttt{true}} \qquad \dfrac{P_4}{\ldots \mid \ldots \wedge \texttt{N <= M} = \texttt{false}}}{(\texttt{false} \downarrow \texttt{in-conf?(network(N),C)} \mid \texttt{fresh-np-id?(network(N),< network(M) | A > C)} = \texttt{true})}}{(\texttt{false} \downarrow \texttt{in-conf?(NP,C)} \mid \texttt{fresh-np-id?(NP,< network(M) | A > C)} = \texttt{true})} \right.$$

$$P_1 = \left\{ \textsc{Simplify}\ \dfrac{\textsc{Con. Join.}\ \dfrac{}{(\texttt{false} \downarrow \texttt{in-conf?(NP,C)} \mid \texttt{fresh-np-id?(NP,C)} = \texttt{true})}}{(\texttt{false} \downarrow \texttt{in-conf?(NP,C)} \mid \texttt{fresh-np-id?(NP,< NNP:NonNetProcessId | A > C)} = \texttt{true})} \right.$$

$$\textsc{Gen.}\ \dfrac{\textsc{Case Analysis}\ \dfrac{\dfrac{P_1}{\ldots[\texttt{P'/NNP:NonNetProcessId}]} \qquad \dfrac{P_2}{\ldots[\texttt{P'/network(M)}]}}{(\texttt{false} \downarrow \texttt{in-conf?(NP,C)} \mid \texttt{fresh-np-id?(NP,< P' | A > C)} = \texttt{true})}}{(\texttt{false} \downarrow \texttt{in-conf?(NP,C)} \mid \texttt{NP} \sim p\ \texttt{P'} = \texttt{false} \wedge \texttt{fresh-np-id?(NP,< P' | A > C)} = \texttt{true})}$$

Figure E.5: Application of Confluence Proof System for `in-conf?` Critical Pair

Under this single unifier—let's call it $\alpha$—we must show that $lhs(E_3)\alpha$ is rewritable by some rule in $R^{+}_{\texttt{in-conf?}}$ assuming $cond(E_3)\alpha$ holds. Let us compute the resulting constrained term $(lhs(E_3) \mid cond(E_3))\alpha$ below:

$$(\texttt{in-conf?(P2, < P' | A > < P2 | A2 > C2)} \mid \texttt{P2 \textasciitilde p P' = false}) \tag{E.14}$$

But this constrained term is clearly rewritable by a rule in $R^{+}_{\texttt{in-conf?}}$, as required. Thus, we are able to use variant unification to solve equalities of the form `in-conf?(P,C) = true` for any instance of `in-conf?(P,C)` that is FVP below.

## E.4 IBOS INVARIANT SUFFICIENCY PROOF

In this section, we provide an overview of our formal proof of the sufficiency of our reformulation of same-origin policy invariant with respect to an original formulation [116]. Let us make the above sentence more precise. Like our work, [116] specified IBOS as a rewrite theory and specified the same-origin policy for IBOS as a rewrite theory invariant. In particular, they actually specified the entire co-invariant as the union of instances of four simple constructor pattern predicates—let us call these constrained terms CO-SOP$_i$ for $1 \leqslant i \leqslant 4$. They then combined a by-hand proof of abstraction with a bounded model-checking proof which demonstrated that the initial state $S_0$ could not reach any state inside CO-SOP$_i$ for $1 \leqslant i \leqslant 4$. On the other hand, in our reformulation of the same-origin policy invariant, we chose to specify it (a) as an invariant in the standard sense, and (b) as a single constructor pattern predicate—let us call this constrained term SOP. The sufficiency proof of our invariant then can be made precisely as the following claim:

$$\bigwedge_{1 \leqslant i \leqslant 4} [\![\text{SOP}]\!] \cap [\![\text{CO-SOP}_i]\!] = \varnothing \tag{E.15}$$

This claim can be further broken down into a set of unsatisfiability proofs as follows. Let $\mathcal{E}_{\text{IBOS}}$ refer to the equational theory underlying the IBOS specification and let $\mathcal{E}_{\text{IBOS},\Omega}$ refer to its equational constructor subtheory. Since the (co-)invariants specified above are all constructor pattern predicates, they have the form $(t \mid \phi)$ where $t$ is a constructor term and $\phi$ is a quantifier-free formula. Let $t_{\text{SOP}}$ and $t_{\text{CO-SOP}_i}$ refer to the term part of these constrained terms and $\phi_{\text{SOP}}$ and $\phi_{\text{CO-SOP}_i}$ refer the constraint part for $1 \leqslant i \leqslant 4$. Then, by definition, the above claim is equivalent to:

$$\bigwedge_{1 \leqslant i \leqslant 4} \forall \alpha \in \mathit{Unif}_{\mathcal{E}_{\text{IBOS},\Omega}}(t_{\text{SOP}}, t_{\text{CO-SOP}_i})\ T_{\mathcal{E}_{\text{IBOS}}} \not\models (\phi_{\text{SOP}} \wedge \phi_{\text{CO-SOP}_i})\alpha \tag{E.16}$$

One key idea being exploited here is the fact that constructor theory unification is often both decidable and cheap. Thus, by utilizing constrained constructor pattern predicates, we cut through much of the complexity of deciding disjointness by a single unification. Then, of course, the question becomes: how can we decide $T_{\mathcal{E}_{\text{IBOS}}} \not\models (\phi_{\text{SOP}} \wedge \phi_{\text{CO-SOP}_i})\alpha$? Typically, this inductive satisfiability check is not decidable.

The second key idea we exploit here is that of sound constrained constructor pattern predicate transformations. That is, instead of working directly at the level of formulas as in formula (E.16) above, we retain the constrained term structure and utilize very similar proof rules to those in our hierarchical proof system shown in Table B.1. By a combination of applications of rules similar to Split, Case, and Substitution, we were able to take the term $(t_{\text{SOP}} \mid \phi_{\text{SOP}} \wedge \phi_{\text{CO-SOP}_i})\alpha$ and transform it into a logically equivalent term where unsatisfiability of the condition followed quite easily. The only remaining technique utilized was an application of contextual rewriting to detect unsatisfiability of a constraint whenever the equality `true` $=$ `false` was generated. Of course, in the intial model $T_{\mathcal{E}_{\text{IBOS}}}$, these two constants are disjoint, so if any formula, we have $T_{\mathcal{E}_{\text{IBOS}}} \models \phi \Rightarrow$ `true` $=$ `false`, we must have $T_{\mathcal{E}_{\text{IBOS}}} \not\models \phi$.

In fact, these four techniques: variant unification, case analysis, formula splitting, and detecting unsatisfiability by equating `true` and `false` were sufficient to carry out the complete disjointness proof. However, there is one last important principle at work: the use of what we refer to as *structural modularity* in Chapter 6. This modularity principle is the glue that binds the whole proof together and automates away the brunt of the "work." What kind of work are we talking about? Even though we have powerful proof rules at our disposal, it is unfortunately often *not clear* how or where they should be applied. This problem becomes worse as the number of proof goals scales.

The answer to this problem is to *generalize and conquer* via *structural modularity*. What we mean is this: suppose that one instance of a formula $\phi\alpha_i$ is proved unsatisfiable by an application of a formula split. Then, it is extremely likely we will have similar instances $\phi\alpha_j$ with $i \neq j$ that can be solved by application of a nearly identical formula split. Furthermore, the choice of terms to include in this formula split are typically *parameters exposed directly in the structure of* $(t \mid \phi)$ *itself*. Indeed, we preserve the term portion of the constrained term $(t \mid \phi)$ instead of discarding it because it provides an excellent source of structure upon which to *generate* the right applications of the proof rules.

Thus, at the end, we prove emptiness of the constrained terms $(t \mid \phi)$ by *structurally guided* versions of the Case Analysis and Split rules together with fully automated Substitution (via variant unification) and Inductive Unsatisfiability (via contextural rewriting) rules. In fact, all of these proof rules have been fully automated with a simple API in Maude. The full details of the proof are available at the IBOS case study repository: `https://github.com/sskeirik/ibos-case-study`.

# REFERENCES

[1] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, “How amazon web services uses formal methods,” *Commun. ACM*, vol. 58, no. 4, pp. 66–73, 2015.

[2] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, “Formal methods: Practice and experience,” *ACM Comput. Surv.*, vol. 41, no. 4, pp. 19:1–19:36, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1592434.1592436

[3] J. Meseguer, “Rewriting logic as a semantic framework for concurrency: a progress report,” in *Proc. CONCUR’96, Pisa, August 1996.* Springer LNCS 1119, 1996, pp. 331–372.

[4] J. Meseguer and G. Rosu, “The rewriting logic semantics project: A progress report,” *Inf. Comput.*, vol. 231, pp. 38–69, 2013.

[5] F. Durán, C. Rocha, and J. M. Álvarez, *Towards a Maude Formal Environment.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 329–351. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24933-4_17

[6] M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott, *All About Maude – A High-Performance Logical Framework.* Springer LNCS Vol. 4350, 2007.

[7] J.-L. Lassez and K. Marriott, “Explicit representation of terms defined by counter examples,” *J. Autom. Reasoning*, vol. 3, no. 3, pp. 301–317, 1987.

[8] J. l. Lassez, M. Maher, and K. Marriott, “Elimination of negation in term algebras,” in *In Mathematical Foundations of Computer Science.* Springer, 1991, pp. 1–16.

[9] M. Tajine, “The negation elimination from syntactic equational formula is decidable,” in *Proc. RTA-93*, ser. LNCS, vol. 690. Springer, 1993, pp. 316–327.

[10] R. Pichler, “Explicit versus implicit representations of subsets of the Herbrand universe,” *Theor. Comput. Sci.*, vol. 290, no. 1, pp. 1021–1056, 2003.

[11] M. Fernández, “Negation elimination in empty or permutative theories,” *J. Symb. Comput.*, vol. 26, no. 1, pp. 97–133, 1998.

[12] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud, “Introducing OBJ,” in *Software Engineering with OBJ: Algebraic Specification in Action.* Kluwer, 2000, pp. 3–167.

[13] K. Futatsugi and R. Diaconescu, *CafeOBJ Report.* World Scientific, 1998.

[14] M. J. Maher, “Complete axiomatizations of the algebras of finite, rational and infinite trees,” in *Proc. LICS ’88.* IEEE Computer Society, 1988, pp. 348–357.

[15] M. J. Maher, “Complete axiomatizations of the algebras of finite, rational and infinite trees,” IBM T. J. Watson Research Center, Tech. Rep., 1988.

[16] H. Comon and P. Lescanne, “Equational problems and disunification,” *Journal of Symbolic Computation*, vol. 7, pp. 371–425, 1989.

[17] H. Comon, “Equational formulas in order-sorted algebras,” in *Proc. ICALP’90*, ser. LNCS, vol. 443. Springer, 1990, pp. 674–688.

[18] H. Comon and C. Delor, “Equational formulae with membership constraints,” *Inf. Comput.*, vol. 112, no. 2, pp. 167–216, 1994.

[19] G. Nelson and D. C. Oppen, “Simplification by cooperating decision procedures,” *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 2, pp. 245–257, 1979.

[20] D. C. Oppen, “Complexity, convexity and combinations of theories,” *Theor. Comput. Sci.*, vol. 12, pp. 291–302, 1980.

[21] R. E. Shostak, “Deciding combinations of theories,” *Journal of the ACM*, vol. 31, no. 1, pp. 1–12, Jan. 1984.

[22] F. Baader and K. U. Schulz, “Combining constraint solving,” in *Constraints in Computational Logics CCL’99, International Summer School*, vol. 2002. Springer LNCS, 1999, pp. 104–158.

[23] H. Comon-Lundth and S. Delaune, “The finite variant property: how to get rid of some algebraic properties,” in Proc *RTA’05*, Springer LNCS 3467, 294–307, 2005.

[24] S. Escobar, R. Sasse, and J. Meseguer, “Folding variant narrowing and optimal variant termination,” *J. Algebraic and Logic Programming*, vol. 81, pp. 898–928, 2012.

[25] J. Meseguer, “Variant-based satisfiability in initial algebras,” *Sci. Comput. Program.*, vol. 154, pp. 3–41, 2018. [Online]. Available: https://doi.org/10.1016/j.scico.2017.09.001

[26] A. Stefanescu, Ştefan Ciobâcă, R. Mereuta, B. M. Moore, T. Serbanuta, and G. Rosu, “All-path reachability logic,” in *Proc. RTA-TLCA 2014*, vol. 8560. Springer LNCS, 2014, pp. 425–440.

[27] A. Stefanescu, D. Park, S. Yuwen, Y. Li, and G. Rosu, “Semantics-based program verifiers for all languages,” in *Proc. OOPSLA 2016*. ACM, 2016, pp. 74–91.

[28] J. Meseguer, “Twenty years of rewriting logic,” *J. Algebraic and Logic Programming*, vol. 81, pp. 721–781, 2012.

[29] D. Lucanu, V. Rusu, A. Arusoaie, and D. Nowak, “Verifying reachability-logic properties on rewriting-logic specifications,” in *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, vol. 9200. Springer LNCS, 2015, pp. 451–474.

[30] F. Baader and W. Snyder, “Unification theory,” in *Handbook of Automated Reasoning*. Elsevier, 1999.

[31] H. Comon, “Complete axiomatizations of some quotient term algebras,” *Theor. Comput. Sci.*, vol. 118, no. 2, pp. 167–191, 1993.

[32] J. Meseguer, “Membership algebra as a logical framework for equational specification,” in *Proc. WADT’97*. Springer LNCS 1376, 1998, pp. 18–61.

[33] J. Goguen and J. Meseguer, “Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations,” *Theoretical Computer Science*, vol. 105, pp. 217–273, 1992.

[34] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1*. Springer, 1985.

[35] J. Meseguer, “Variant-based satisfiability in initial algebras,” University of Illinois at Urbana-Champaign, Tech. Rep. `http://hdl.handle.net/2142/88408`, November 2015.

[36] J. Meseguer, “Variant-based satisfiability in initial algebras,” in *Proc. FTSCS 2015*, C. Artho and P. Ölveczky, Eds. Springer CCIS 596, 2016, pp. 1—32.

[37] J. Goguen and R. Burstall, “Institutions: Abstract model theory for specification and programming,” *Journal of the ACM*, vol. 39, no. 1, pp. 95–146, 1992.

[38] N. Dershowitz and J.-P. Jouannaud, “Rewrite systems,” in *Handbook of Theoretical Computer Science, Vol. B*, J. van Leeuwen, Ed. North-Holland, 1990, pp. 243–320.

[39] J. Meseguer, “Strict coherence of conditional rewriting modulo axioms,” C.S. Department, University of Illinois at Urbana-Champaign, Tech. Rep. `http://hdl.handle.net/2142/50288`, August 2014, submitted to *Theoretical Computer Science*.

[40] J.-P. Jouannaud and H. Kirchner, “Completion of a set of rules modulo a set of equations,” *SIAM Journal of Computing*, vol. 15, pp. 1155–1194, November 1986.

[41] A. Cholewa, J. Meseguer, and S. Escobar, “Variants of variants and the finite variant property,” CS Dept. University of Illinois at Urbana-Champaign, Tech. Rep., February 2014, available at `http://hdl.handle.net/2142/47117`.

[42] R. Gutiérrez, J. Meseguer, and C. Rocha, “Order-sorted equality enrichments modulo axioms,” *Sci. Comput. Program.*, vol. 99, pp. 235–261, 2015.

[43] J. Meseguer, “Generalized rewrite theories and coherence completion,” in *Proc. Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018*, ser. Lecture Notes in Computer Science, V. Rusu, Ed., vol. 11152. Springer, 2018, pp. 164–183.

[44] S. Lucas and J. Meseguer, “Normal forms and normal theories in conditional rewriting,” *J. Log. Algebr. Meth. Program.*, vol. 85, no. 1, pp. 67–97, 2016. [Online]. Available: https://doi.org/10.1016/j.jlamp.2015.06.001

[45] F. Durán and J. Meseguer, “On the church-rosser and coherence properties of conditional order-sorted rewrite theories,” *J. Log. Algebr. Program.*, vol. 81, no. 7-8, pp. 816–850, 2012.

[46] R. Bruni and J. Meseguer, “Semantic foundations for generalized rewrite theories.” *Theor. Comput. Sci.*, vol. 360, no. 1-3, pp. 386–414, 2006.

[47] S. Skeirik and J. Meseguer, “Metalevel algorithms for variant-based satisfiability,” in *Proc. WRLA 2016*, D. Lucanu, Ed., vol. 9942. Springer LNCS, 2016, pp. 167–184.

[48] P. Hudak, J. Hughes, S. L. P. Jones, and P. Wadler, “A history of haskell: being lazy with class,” in *Proc. Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III)*. ACM, 2007, pp. 1–55.

[49] A. van Deursen, J. Heering, and P. Klint, *Language Prototyping: An Algebraic Specification Approach.* World Scientific, 1996.

[50] J. Meseguer, “Conditional rewriting logic as a unified model of concurrency,” *Theoretical Computer Science*, vol. 96, no. 1, pp. 73–155, 1992.

[51] J. Meseguer, J. Goguen, and G. Smolka, “Order-sorted unification,” to appear in the *Journal of Symbolic Computation*, special issue on unification.

[52] M. Alpuente, S. Escobar, J. Espert, and J. Meseguer, “A modular order-sorted equational generalization algorithm,” *Inf. Comput.*, vol. 235, pp. 98–136, 2014.

[53] J. Guttag, “The specification and application to programming of abstract data types,” Ph.D. dissertation, University of Toronto, 1975, computer Science Department, Report CSRG-59.

[54] J. Meseguer, M. Palomino, and N. Martí-Oliet, “Equational abstractions,” *Theoretical Computer Science*, vol. 403, no. 2-3, pp. 239–264, 2008.

[55] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. L”oding, S. Tison, and M. Tommasi, “Tree automata techniques and applications,” Available at: http://tata.gforge.inria.fr/, 2008, Release November 18, 2008.

[56] J. Hendrix, J. Meseguer, and H. Ohsaki, “A sufficient completeness checker for linear order-sorted specifications modulo axioms,” in *Automated Reasoning, Third International Joint Conference, IJCAR 2006*, 2006, pp. 151–155.

[57] J. D. Hendrix, “Decision procedures for equationally based reasoning,” Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2008, `http://hdl.handle.net/2142/10967`.

[58] M. Clavel, J. Meseguer, and M. Palomino, "Reflection in membership equational logic, many-sorted equational logic, Horn logic with equality, and rewriting logic," *Theoretical Computer Science*, vol. 373, pp. 70–91, 2007.

[59] C. Rocha and J. Meseguer, "Constructors, sufficient completeness, and deadlock freedom of rewrite theories," in *Proc. LPAR 2010*, ser. Lecture Notes in Computer Science, vol. 6397. Springer, 2010, pp. 594–609.

[60] W. S. Brainerd, "Tree generating regular systems," 1969.

[61] S. Skeirik, A. Stefanescu, and J. Meseguer, "A constructor-based reachability logic for rewrite theories," University of Illinois Computer Science Department, Tech. Rep., March 2017, available at : `http://hdl.handle.net/2142/95770`. To appear (shorter version) in *Proc. LOPSTR 2107*, Springer LNCS 2018.

[62] A. R. Bradley and Z. Manna, *The calculus of computation - decision procedures with applications to verification*. Springer, 2007.

[63] D. Kroening and O. Strichman, *Decision Procedures - An Algorithmic Point of View*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[64] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, February 2009, vol. 185, ch. 26, pp. 825–885.

[65] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*, E. Clarke, T. Henzinger, and H. Veith, Eds. Springer, 2014, (to appear).

[66] C. Barrett, I. Shikanian, and C. Tinelli, "An abstract decision procedure for satisfiability in the theory of inductive data types," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 3, pp. 21–46, 2007.

[67] S. Krstic, A. Goel, J. Grundy, and C. Tinelli, "Combined satisfiability modulo parametric theories," in *Proc. TACAS 2007*, vol. 4424. Springer LNCS, 2007, pp. 602–617.

[68] A. Armando, S. Ranise, and M. Rusinowitch, "A rewriting approach to satisfiability procedures," *Inf. Comput.*, vol. 183, no. 2, pp. 140–164, 2003.

[69] C. Dross, S. Conchon, J. Kanig, and A. Paskevich, "Adding Decision Procedures to SMT Solvers using Axioms with Triggers," *Journal of Automated Reasoning*, 2016, accepted for publication. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01221066

[70] M. Clavel, F. Durán, S. Eker, J. Meseguer, and M.-O. Stehr, "Maude as a formal meta-tool," in *FM'99 — Formal Methods*, ser. Springer LNCS, J. Wing and J. Woodcock, Eds., vol. 1709. Springer-Verlag, 1999, pp. 1684–1703.

[71] G. Rosu and T. Serbanuta, "An overview of the K semantic framework," *J. Log. Algebr. Program.*, vol. 79, no. 6, pp. 397–434, 2010.

[72] J. H. Siekmann, "Unification theory," *J. Symb. Comput.*, vol. 7, no. 3/4, pp. 207–274, 1989.

[73] J.-P. Jouannaud and C. Kirchner, "Solving equations in abstract algebras: A rule-based survey of unification." in *Computational Logic - Essays in Honor of Alan Robinson.* MIT Press, 1991, pp. 257–321.

[74] F. Baader and J. H. Siekmann, "Unification theory," in *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2.* Oxford University Press, 1994, pp. 41–126.

[75] F. Baader and W. Snyder, "Unification theory," in *Handbook of Automated Reasoning (in 2 volumes).* Elsevier and MIT Press, 2001, pp. 445–532.

[76] J.-M. Hullot, "Canonical forms and unification," in *Proc. Fifth Conference on Automated Deduction*, ser. LNCS. Springer, 1980, vol. 87, pp. 318–334.

[77] J.-P. Jouannaud, C. Kirchner, and H. Kirchner, "Incremental construction of unification algorithms in equational theories," in *Proc. ICALP'83*. Springer LNCS 154, 1983, pp. 361–373.

[78] J. Meseguer and P. Thati, "Symbolic reachability analysis using narrowing and its application to the verification of cryptographic protocols," *J. Higher-Order and Symbolic Computation*, vol. 20, no. 1–2, pp. 123–160, 2007.

[79] F. Baader and K. U. Schulz, "Combination techniques and decision problems for disunification," *Theor. Comput. Sci.*, vol. 142, no. 2, pp. 229–255, 1995.

[80] J. Meseguer and S. Skeirik, "Equational formulas and pattern operations in initial order-sorted algebras," in *Proc. LOPSTR 2015*, M. Falaschi, Ed., vol. 9527. Springer LNCS, 2015, pp. 36—53.

[81] J. Giesl and D. Kapur, "Decidable classes of inductive theorems," in *Proc. IJCAR 2001*, vol. 2083. Springer LNCS, 2001, pp. 469–484.

[82] J. Giesl and D. Kapur, "Deciding inductive validity of equations," in *Proc. CADE 2003*, vol. 2741. Springer LNCS, 2003, pp. 17–31.

[83] S. Falke and D. Kapur, "Rewriting induction + linear arithmetic = decision procedure," in *Proc. IJCAR 2012*, vol. 7364. Springer LNCS, 2012, pp. 241–255.

[84] T. Aoto and S. Stratulat, "Decision procedures for proving inductive theorems without induction," in *Proc. PPDP2014*. ACM, 2014, pp. 237–248.

[85] K. Bae and J. Meseguer, "Model checking linear temporal logic of rewriting formulas under localized fairness," *Sci. Comput. Program.*, vol. 99, pp. 193–234, 2015.

[86] K. Futatsugi, "Fostering proof scores in CafeOBJ," in *Proc. ICFEM 2010*, vol. 6447. Springer LNCS, 2010, pp. 1–20.

[87] J. Meseguer, "Generalized rewrite theories, coherence completion, and symbolic methods," 2019, to appear in *Journal of Logical and Algebraic Methods in Programming.*

[88] S. Skeirik and J. Meseguer, "Metalevel algorithms for variant satisfiability," *J. Log. Algebr. Meth. Program.*, vol. 96, pp. 81–110, 2018.

[89] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[90] G. Rosu and A. Stefanescu, "From Hoare logic to matching logic reachability," in *FM*, ser. Lecture Notes in Computer Science, D. Giannakopoulou and D. Méry, Eds., vol. 7436. Springer, 2012, pp. 387–402.

[91] J. C. Reynolds, "Separation logic: A logic for shared mutable data structures," in *LICS 2002*. IEEE, 2002, pp. 55–74.

[92] J. Meseguer and G. Roşu, "The rewriting logic semantics project," *Theoretical Computer Science*, vol. 373, pp. 213–237, 2007.

[93] C. Ellison and G. Rosu, "An executable formal semantics of C with applications," in *POPL*, J. Field and M. Hicks, Eds. ACM, 2012, pp. 533–544.

[94] G. Dowek, T. Hardin, and C. Kirchner, "Theorem proving modulo." *J. Autom. Reasoning*, vol. 31, no. 1, pp. 33–72, 2003.

[95] P. Viry, "Adventures in sequent calculus modulo equations," *Electr. Notes Theor. Comput. Sci.*, vol. 15, pp. 21–32, 1998. [Online]. Available: http://dx.doi.org/10.1016/S1571-0661(05)82550-2

[96] C. Rocha and J. Meseguer, "Theorem proving modulo based on boolean equational procedures," in *Proc. RelMiCS 2008*, vol. 4988. Springer LNCS, 2008, pp. 337–351.

[97] F. Durán and P. C. Ölveczky, "A guide to extending full maude illustrated with the implementation of real-time maude," *Electronic Notes in Theoretical Computer Science*, vol. 238, no. 3, pp. 83 – 102, 2009.

[98] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, "Associative unification and symbolic reasoning modulo associativity in maude," in *Proc. Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018*, ser. Lecture Notes in Computer Science, V. Rusu, Ed., vol. 11152. Springer, 2018, pp. 98–114.

[99] G. Rosu and A. Stefanescu, "Checking reachability using matching logic," in *Proc. OOPSLA 2012*. ACM, 2012, pp. 555–574.

[100] S. Skeirik, A. Stefanescu, and J. Meseguer, "A constructor-based reachability logic for rewrite theories," in *Proc. Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017*, ser. Lecture Notes in Computer Science, vol. 10855. Springer, 2017, pp. 201–217.

[101] C. Rocha, J. Meseguer, and C. A. Muñoz, "Rewriting modulo SMT and open system analysis," *Journal of Logic and Algebraic Methods in Programming*, vol. 86, pp. 269–297, 2017.

[102] B. Moore, "Coinductive program verification," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2016, `http://hdl.handle.net/2142/95372`.

[103] D. Lucanu, V. Rusu, and A. Arusoaie, "A generic framework for symbolic execution: A coinductive approach," *J. Symb. Comput.*, vol. 80, pp. 125–163, 2017.

[104] Ştefan Ciobâcă and D. Lucanu, "A coinductive approach to proving reachability properties in logically constrained term rewriting systems," in *Proc. IJCAR 2018*, ser. Lecture Notes in Computer Science, vol. 10900. Springer, 2018, pp. 295–311.

[105] J. Goguen, "OBJ as a theorem prover with application to hardware verification," in *Current Trends in Hardware Verification and Automated Theorem Proving*, P. Subramanyam and G. Birtwistle, Eds. Springer-Verlag, 1989, pp. 218–267.

[106] K. Futatsugi, "Generate & check method for verifying transition systems in cafeobj," in *Software, Services, and Systems - Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering*, ser. Lecture Notes in Computer Science, R. De Nicola and R. Hennicker, Eds., vol. 8950. Springer, 2015, pp. 171–192.

[107] D. Gâinâ, D. Lucanu, K. Ogata, and K. Futatsugi, "On automation of ots/cafeobj method," in *Specification, Algebra, and Software - Essays Dedicated to Kokichi Futatsugi*, ser. Lecture Notes in Computer Science, S. Iida, J. Meseguer, and K. Ogata, Eds., vol. 8373. Springer, 2014, pp. 578–602.

[108] A. Riesco and K. Ogata, "Prove it! inferring formal proof scripts from cafeobj proof scores," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 2, pp. 6:1–6:32, 2018.

[109] C. Rocha and J. Meseguer, "Proving safety properties of rewrite theories," 2011, in Proc. CALCO 2011, Springer LNCS 6859, 314-328.

[110] C. Rocha, "Symbolic reachability analysis for rewrite theories," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2012.

[111] C. Rocha and J. Meseguer, "Mechanical analysis of reliable communication in the alternating bit protocol using the Maude invariant analyzer tool," in *Specification, Algebra, and Software - Essays Dedicated to Kokichi Futatsugi*, ser. Lecture Notes in Computer Science, vol. 8373. Springer, 2014, pp. 603–629.

[112] S. Chen, J. Meseguer, R. Sasse, H. J. Wang, and Y.-M. Wang, "A systematic approach to uncover security flaws in gui logic," in *IEEE Symposium on Security and Privacy*. IEEE, 2007, pp. 71–85.

[113] S. Tang, H. Mai, and S. T. King, "Trust and protection in the illinois browser operating system," in *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*. USENIX Association, 2010, pp. 17–32.

[114] S. Tang, "Towards secure web browsing," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2011, 2011-05-25, `http://hdl.handle.net/2142/24307`.

[115] R. Sasse, "Security models in rewriting logic for cryptographic protocols and browsers," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2012, `http://hdl.handle.net/2142/34373`.

[116] R. Sasse, S. T. King, J. Meseguer, and S. Tang, "IBOS: A correct-by-construction modular browser," in *FACS 2012*, ser. Lecture Notes in Computer Science, vol. 7684. Springer, 2013, pp. 224–241.

[117] J. Meseguer, "Order-sorted rewriting and congruence closure," in *Proc. FOSSACS 2016*, ser. Lecture Notes in Computer Science, vol. 9634. Springer, 2016, pp. 493–509.

[118] J. Meseguer, "A logical theory of concurrent objects and its realization in the Maude language," in *Research Directions in Concurrent Object-Oriented Programming*, G. Agha, P. Wegner, and A. Yonezawa, Eds. MIT Press, 1993, pp. 314–390.

[119] J. Meseguer, "Generalized rewrite theories, coherence completion and symbolic methods," University of Illinois Computer Science Department, Tech. Rep. `http://hdl.handle.net/2142/102183`, December 2018.

[120] S. Chen, D. Ross, and Y.-M. Wang, "An analysis of browser domain-isolation bugs and a light-weight transparent defense mechanism," in *ACM Conference on Computer and Communications Security*. ACM, 2007, pp. 2–11.

[121] J. Schwenk, M. Niemietz, and C. Mainka, "Same-origin policy: Evaluation in modern browsers," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schwenk pp. 713–727.

[122] G. Klein and H. Tuch, "Towards verified virtual memory in l4," *TPHOLs Emerging Trends*, vol. 4, p. 16, 2004.

[123] R. Kolanski and G. Klein, "Formalising the l4 microkernel api," in *Proceedings of the 12th Computing: The Australasian Theroy Symposium-Volume 51*. Australian Computer Society, Inc., 2006, pp. 53–68.

[124] *JavaScript Guide (1.2)*. Netscape Communications Corporation, 1997, originally http://developer.netscape.com/docs/manuals/communicator/jsguide4/index.htm; accessed at https://www.cs.rit.edu/~atk/JavaScript/manuals/jsguide/.

[125] A. Barth, C. Jackson, and J. C. Mitchell, "Securing frame communication in browsers," *Communications of the ACM*, vol. 52, no. 6, pp. 83–91, 2009.

[126] C. Jackson and A. Barth, "Beware of finer-grained origins." Web, 2008.

[127] C. Karlof, U. Shankar, J. D. Tygar, and D. Wagner, "Dynamic pharming attacks and locked same-origin policies for web browsers," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 58–71.

[128] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, "Protecting browser state from web privacy attacks," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 737–744.

[129] L.-S. Huang, Z. Weinberg, C. Evans, and C. Jackson, “Protecting browsers from cross-origin css attacks,” in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS ’10. New York, NY, USA: ACM, 2010. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866376 pp. 619–629.

[130] R. Wang, L. Xing, X. Wang, and S. Chen, “Unauthorized origin crossing on mobile platforms: Threats and mitigation,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, ser. CCS ’13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516727 pp. 635–646.

[131] M. Bugliesi, S. Calzavara, and R. Focardi, “Formal methods for web security,” *Journal of Logical and Algebraic Methods in Programming*, vol. 87, pp. 110–126, 2017.

[132] L. Bauer, S. Cai, L. Jia, T. Passaro, M. Stroucken, and Y. Tian, “Run-time monitoring and formal analysis of information flows in chromium.” in *NDSS*, 2015.

[133] A. Bohannon and B. C. Pierce, “Featherweight firefox: formalizing the core of a web browser,” in *Proceedings of the 2010 USENIX conference on Web application development.* Usenix Association, 2010, pp. 11–11.

[134] A. Bohannon, *Foundations of web script security.* Citeseer, 2012.

[135] D. Jang, Z. Tatlock, and S. Lerner, “Establishing browser security guarantees through formal shim verification,” in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 113–128.

[136] D. Gollmann, “Problems with same origin policy: Know thyself,” in *Security Protocols XVI.* Berlin, Heidelberg: Springer, 2011, pp. 84–85.

[137] D. F. Some, N. Bielova, and T. Rezk, “On the content security policy violations due to the same-origin policy,” ser. WWW ’17, Republic and Canton of Geneva, Switzerland, 2017, pp. 877–886.

[138] A. Rubio, “Automated deduction with constrained clauses,” Ph.D. dissertation, Universitat Politècnica de Catalunya, 1994.

[139] F. Durán, S. Lucas, and J. Meseguer, “Termination modulo combinations of equational theories,” in *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5749. Springer, 2009, pp. 246–262.

[140] R. Gutiérrez, S. Skeirik, and J. Meseuger, “Maude termination assistant,” in Preproceedings of WRLA 2018, Thessaloniki, Greece, April 2018 (distributed in electronic form by the ETAPS 2018 organizers); Proceedings version to appear in LNCS.

[141] F. Durán, J. Meseguer, and C. Rocha, “Proving ground confluence of equational specifications modulo axioms,” in *Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018, Held as a Satellite Event of ETAPS, Thessaloniki, Greece, June 14-15, 2018, Proceedings*, 2018, to appear in Journal of Logic and Algebraic Methods in Programming 2020. [Online]. Available: https://doi.org/10.1007/978-3-319-99840-4_11 pp. 184–204.