A Constructor-Based Reachability Logic for Rewrite Theories

Stephen Skeirik, Andrei Stefanescu and José Meseguer

Department of Computer Science
University of Illinois at Urbana-Champaign, USA

Abstract. Reachability logic has been applied to $\mathbb{K}$ rewrite-rule-based language definitions as a *language-generic* logic of programs to verify a wide range of sophisticated programs in conventional languages. Here we study how reachability logic can be made not just language-generic, but also *rewrite-theory-generic*, so that we can verify *both* conventional programs based on their rewriting logic operational semantics *and* distributed system designs specified as rewrite thories. A theory-generic reachability logic is presented and proved sound for a wide class of rewrite theories. Particular attention is given to increasing the logic's automation by means of *constructor-based* semantic unification, matching, narrowing, and satisfiability procedures. The relationships to Hoare logic and LTL are discussed, new methods for proving invariants of possibly never terminating distributed systems are developed, and experiments with a prototype implementation illustrating the new methods are presented.
Keywords: reachability and rewriting logics, deductive verification.

1 Introduction

The main past applications of reachability logic have been as a *language-generic* logic of programs [1,2,3]. 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 [3]. 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 [2,3], 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 [4,5]? 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 [6], 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 [7], as explained in Section 7 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 4.1, 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 (Σ, E), e.g., [8,9,10,11,12,13,14,15], as well as recent results on decidable satisfiability (and validity) of quantifier-free formulas in initial algebras, e.g., [16,17,18,19,20,21,22,23,24,25,26] to *automate* as much as possible reachability logic deduction. In this regard, the very general foundations of standard reachability logic—which assume any Σ-algebra $\mathcal{A}$ with a first-order-definable transition relation—provide no help at all for automation. As shown in this work 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 [2,3]. Could a reachability logic for rewrite theories be simpler? The main goal of this work 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 reachability logic. Rewriting logic is a *system specification* logic ideally suited for specifying concurrent systems. Instead, reachability logic is a *property specification* logic, were 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 [27], 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.

1.1 Rewriting Logic in a Nutshell

A distributed system can be designed and modeled as a *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$ [4,5] 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

(Σ, E) with function symbols Σ 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 [28] mutual exclusion protocol, explained in detail in Section 2 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 Σ 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

$$w2c : < n \mid w\ i \mid c \mid i; q > \rightarrow < n \mid w \mid c\ i \mid i; q > .$$

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 paper is *theory generic* in the precise sense that, as we explain in Section 5, 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 $A \rightarrow^{\circledast} B$, 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 Σ-terms, and where a state transition $[u] \rightarrow_{\mathcal{R}} [v]$ holds iff $\mathcal{R} \vdash u \rightarrow v$ according to the rewriting logic inference system [4,5] (computation = deduction). As a first approximation, $A \rightarrow^{\circledast} B$ is a Hoare logic *partial correctness* assertion of the form[1] $\{A\}\mathcal{R}\{B\}$, 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 *mid-condition*. More precisely, $A \rightarrow^{\circledast} B$ holds in $\mathcal{T}_{\mathcal{R}}$ iff for each state $[u_0]$ satisfying

[1] The notation $\{A\}\mathcal{R}\{B\}$, and the relation to Hoare logic are explained in Section 4.2.

A and each *terminating* sequence $[u_0] \rightarrow_{\mathcal{R}} [u_1] \ldots \rightarrow_{\mathcal{R}} [u_{n-1}] \rightarrow_{\mathcal{R}} [u_n]$, i.e., $\nexists u\ (u_n \rightarrow_{\mathcal{R}} u)$, 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 $\langle \_ \rangle : Nat \rightarrow State$. The rewrite theory specifying the counter's behavior has two rewrite rules: $\langle n + 1 \rangle \rightarrow \langle n \rangle$, and $\langle n + 1 \rangle \rightarrow \langle n + 1 + 1 \rangle$, i.e., a non-zero counter can increase or decrease by one unit. For n, m variables of sort *Nat*, consider the reachability formula $\langle n + 1 \rangle \mid n + 1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$, 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[2] and φ a quantifier-free (QF) Σ-formula. The state predicate $u \mid \varphi$ holds for a state $[u'] \in T_{\Sigma/E}$ iff there is a ground substitution ρ 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 (Σ, 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.

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 $A \rightarrow^{\circledast} B$. 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 other reachability model. In Section 4.1 we give a systematic solution to this

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

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.

1.4 Paper Outline and Main Contributions

Section 2 gathers preliminaries. Section 3 greatly increases the logic's potential for automation by making state predicates constructor-based. Reachability logic itself is introduced in Section 4. A systematic methodology to prove *invariants* by means of reachability formulas is developed in Section 4.1. The semantic relations of reachability logic to Hoare logic and to LTL are explained in Section 4.2. 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. 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 6. Related work and conclusions are discussed in Section 7. Proofs are relegated to Appendix A. The tool's command grammar is specified in detail in Appendix B.

Comparison with [29]. This work makes the following additional contributions to those in the earlier conference paper [29]:

1. A much fuller treatment of all aspects is given, including considerably more detailed explanations of many topics and detailed proofs of all results.
2. Furthermore, a substantial collection of new definitions and, above all, new results is presented and, for results, proved (9 lemmas, 7 theorems, and 1 corollary, versus 3 lemmas, 5 theorems, and 1 corollary in [29]).
3. All important concepts, and, above all, those regarding reachability logic itself, its formulas, its inference system, how it can be applied to concrete applications, and how such formulas can be proved in practice using our prototype tool are explained and illustrated in detail using a substantial collection of examples.
4. Section 3, that defines our language of *pattern predicates* for specifying state properties, has been quite substantially expanded. In particular, the treatment of, and symbolic operation with, *parametric* pattern predicates, which are crucial for proving many properties, including parametric invariants, is much more extensive and thorough than in [29].
5. Section 5 on reachability logic's inference system has been improved in many ways, including: (i) decomposing one of the inference rules into two for increased flexibility; (ii) extending the applicability conditions of the inference rules, so that they can apply to a wider range of specifications; (iii) adding an automatic check to detect invalid formulas; and (iv) expanding both the generality and the repertoire of auxiliary rules.
6. Section 6 on the logic's implementation and experiments has been substantially extended in various ways. First of all, the prototype implementation itself has been substantially advanced, and various new commands have been

added. Second, detailed examples illustrating the use of these various commands have been given. Finally, a summary of a substantial suite of examples with various measures of proof complexity and degree of automation has been included. One important difference with the smaller suite of examples in [29], is that before only examples where inductive validity of formulas could be decided by variant satisfiability [30,31] could be handled. Instead, now verification of properties for fully general rewrite theories —provided they satisfy the fairly mild requirements imposed by the logic and its inference system— can be handled by the tool. This is illustrated by several of the examples in the new suite.

2 Order-Sorted Algebra and Rewriting Logic

We present some preliminaries on order-sorted algebra and rewriting logic. The material is adapted from [32,26,5]. The presentation is self-contained.

2.1 Many-Sorted and Order-Sorted Algebras

Definition 1 (Many-Sorted Signature). *A* many-sorted (MS) signature *is a pair (S, Σ), where S is called a set of* sorts*, and Σ is called a set of function symbols, which are typed with sorts in S. In more detail, Σ is a $S^* \times S$-indexed family of sets $\Sigma = \{\Sigma_{w,s}\}_{(w,s)\in S^*\times S}$. Notationally, if $f \in \Sigma_{s_1\ldots s_n,s}$ we then display f as $f : s_1 \ldots s_n \to s$. To abbreviate notation we sometimes write $\Sigma = (S, \Sigma)$, leaving the set S of sorts implicit.*

Definition 2 (Many-Sorted Algebra). *Given a many-sorted signature $\Sigma = (S, \Sigma)$, a* Σ-algebra *is an S-indexed set $A = \{A_s\}_{s\in S}$ together with an interpretation of each $f : s_1 \ldots s_n \to s$ in Σ as a function $A_f : A_{s_1} \times \ldots A_{s_n} \to A_s$. For ϵ the empty word in S^*, a symbol $a \in \Sigma_{\epsilon,s}$ is called a* constant symbol *and is interpreted in A as a constant element $A_a \in A_s$. There is no requirement that if $(w, s) \neq (w', s')$ then $\Sigma_{w,s} \cap \Sigma_{w',s'} = \emptyset$. To avoid confusion in such a case, if $f \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ we then write $A_{f:w\to s}$ and $A_{f:w'\to s'}$ for the corresponding interpretations in A, instead of the (now ambiguous) A_f.*

Given Σ-algebras A and B, a Σ-homomophism *is an S-indexed family of functions $h = \{h_s : A_s \to B_s\}_{s\in S}$ such that: (i) for each $s \in S$ and constant symbol $a \in \Sigma_{\epsilon,s}$, $h_s(A_a) = B_a$, and (ii) for each $f : s_1 \ldots s_n \to s$ in Σ and each $(a_1, \ldots, a_n) \in A_{s_1} \times \ldots A_{s_n}$, $h_s(A_f(a_1, \ldots, a_n)) = B_f(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$.*

Definition 3 (OS Signature). *An* order-sorted (OS) signature *is a triple $\Sigma = (S, \leqslant, \Sigma)$ with $(S, \leqslant)$ a poset and (S, Σ) a many-sorted signature. $\widehat{S} = S/\equiv_{\leqslant}$, the quotient of S under the equivalence relation $\equiv_{\leqslant} = (\leqslant \cup \geqslant)^+$, is called the set of* connected components *of $(S, \leqslant)$. The order $\leqslant$ and equivalence $\equiv_{\leqslant}$ are extended to sequences of the same length in the usual way, i.e., $s'_1 \ldots s'_n \leqslant s_1 \ldots s_n$ iff $s'_i \leqslant s_i$, $1 \leqslant i \leqslant n$. Σ is called* sensible *if for any two $f : w \to s, f : w' \to s' \in \Sigma$, with w and w' of same length, we have $w \equiv_{\leqslant} w' \;\Rightarrow\; s \equiv_{\leqslant} s'$. A* many-sorted

signature Σ *is the special case of an order-sorted signature where the poset* $(S, \leqslant)$ *is discrete, i.e.,* $s \leqslant s'$ *iff* $s = s'$.

For connected components $[s_1], \ldots, [s_n], [s] \in \widehat{S}$

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

denotes the family of "subsort polymorphic" operators f. □

Definition 4 (OS Algebra). *For* $\Sigma = (S, \leqslant, \Sigma)$ *an OS signature, an* order-sorted Σ-algebra A *is a many-sorted* (S, Σ)*-algebra* A *such that:*

- *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 Σ-homomorphism $h : A \to B$ *is a many-sorted* (S, Σ)*-homomorphism such that for* $s, s' \in S$*, whenever* $[s] = [s']$ *and* $a \in A_s \cap A_{s'}$*, then we have* $h_s(a) = h_{s'}(a)$. *A* Σ-isomorphism $h : A \to B$ *is a* Σ*-homomorphism* $h : A \to B$ *such that: (i) for each* $s \in S$ *the map* $h_s : A_s \to B_s$ *is bijective, and (ii)* $h^{-1} : B \to A$ *is also a* Σ*-homomorphism. This defines a category* $\mathbf{OSAlg}_\Sigma$. □

Theorem 1 (Initiality). *[32] The category* $\mathbf{OSAlg}_\Sigma$ *has an initial algebra. Furthermore, if* Σ *is sensible, then the term algebra* T_Σ *with:*

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

is initial, i.e., there is a unique Σ*-homomorphism from it to each* Σ*-algebra.*

For any algebra A and $[s] \in \widehat{S}$, $A_{[s]}$ denotes the set $A_{[s]} = \bigcup_{s' \in [s]} A_{s'}$. We let T_Σ (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 Σ is said to *have non-empty sorts* iff for each $s \in S$, $T_{\Sigma,s} \neq \varnothing$. An OS signature Σ is called *preregular* [33] iff for each $t \in T_\Sigma$ the set $\{s \in S \mid t \in T_{\Sigma,s}\}$ has a least element, denoted $ls(t)$. *We will assume throughout that* Σ *has non-empty sorts and is preregular.*

An S-sorted set $X = \{X_s\}_{s\in S}$ of *variables*, satisfies $s \neq s' \Rightarrow X_s \cap X_{s'} = \varnothing$, and the variables in X are always assumed disjoint from all constants in Σ. The Σ*-term algebra* on variables X, $T_\Sigma(X)$, is the *initial* Σ*-algebra* for the signature $\Sigma(X)$ obtained by adding to Σ the variables X *as extra constants*. Since a $\Sigma(X)$-algebra is just a pair (A, α), with A a Σ-algebra, and α 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 theorem:

Theorem 2 (Freeness Theorem). *If* Σ *is sensible, for each* $A \in \mathbf{OSAlg}_\Sigma$ *and* $\alpha \in [X \to A]$*, there exists a unique* Σ*-homomorphism,* $\_\alpha : T_\Sigma(X) \to A$ *extending* α*, 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 σ on Z and is the identity elsewhere.

2.2 Order-Sorted First-Order Logic

The first-order language of *equational Σ-formulas* is defined in the usual way: its atoms are *Σ-equations* $t = t'$, where $t, t' \in T_\Sigma(X)_{[s]}$ for some $[s] \in \widehat{S}$ and each X_s is assumed countably infinite. The set $Form(\Sigma)$ of *equational Σ-formulas*[3] is then inductively built from atoms by: conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and universal ($\forall x_1 : s_1, \ldots, x_n : s_n$) and existential ($\exists x_1{:}s_1, \ldots, x_n{:}s_n$) quantification with distinct sorted variables $x_1{:}s_1, \ldots, x_n{:}s_n$, with $s_1, \ldots, s_n \in S$ (by convention, for $\varnothing$ the empty set of variables and φ a formula, we define $(\forall \varnothing)\ \varphi \equiv (\exists \varnothing)\ \varphi \equiv \varphi$). A literal $\neg(t = t')$ is denoted $t \neq t'$. Given a Σ-algebra A, a formula $\varphi \in Form(\Sigma)$, and an assignment $\alpha \in [Y \to A]$, where $Y \supseteq fvars(\varphi)$, with $fvars(\varphi)$ the free variables of φ, the *satisfaction relation* $A, \alpha \models \varphi$ is defined inductively as usual: for atoms, $A, \alpha \models t = t'$ iff $t\alpha = t'\alpha$; for Boolean connectives it is the corresponding Boolean combination of the satisfaction relations for subformulas; and for quantifiers: $A, \alpha \models (\forall x_1{:}s_1, \ldots, x_n{:}s_n)\ \varphi$ (resp. $A, \alpha \models (\exists x_1{:}s_1, \ldots, x_n{:}s_n)\ \varphi$) holds iff for all $(a_1, \ldots, a_n) \in A_{s_1} \times \ldots \times A_{s_n}$ (resp. for some $(a_1, \ldots, a_n) \in A_{s_1} \times \ldots \times A_{s_n}$) we have $A, \alpha[x_1 : s_1 := a_1, \ldots, x_n : s_n := a_n] \models \varphi$, where if $\alpha \in [Y \to A]$, then $\alpha[x_1{:}s_1 := a_1, \ldots, x_n{:}s_n := a_n] \in [(Y \cup \{x_1{:}s_1, \ldots, x_n{:}s_n\}) \to A]$ and is such that for $y{:}s \in (Y \backslash \{x_1{:}s_1, \ldots, x_n{:}s_n\})$, $\alpha[x_1{:}s_1 := a_1, \ldots, x_n{:}s_n := a_n](y{:}s) = \alpha(y{:}s)$, and $\alpha[x_1 : s_1 := a_1, \ldots, x_n{:}s_n := a_n](x_i{:}s_i) = a_i$, $1 \leqslant i \leqslant n$. We say that φ is *valid* in A (resp. is *satisfiable* in A) iff $A, \underline{\varnothing} \models (\forall Y)\ \varphi$ (resp. $A, \underline{\varnothing} \models (\exists Y)\ \varphi$), where $Y = fvars(\varphi)$ and $\underline{\varnothing} \in [\underline{\varnothing} \to A]$ denotes the empty S-sorted assignment of values in A to the empty S-sorted family $\underline{\varnothing}$ of variables. The notation $A \models \varphi$ abbreviates validity of φ in A. More generally, a set of formulas $\Gamma \subseteq Form(\Sigma)$ is called *valid* in A, denoted $A \models \Gamma$, iff $A \models \varphi$ for each $\varphi \in \Gamma$. For a subsignature $\Omega \subseteq \Sigma$ and $A \in \mathbf{OSAlg}_\Sigma$, the *reduct* $A|_\Omega \in \mathbf{OSAlg}_\Omega$ agrees with A in the interpretation of all sorts and operations in Ω and discards everything in $\Sigma \backslash \Omega$. If $\varphi \in Form(\Omega)$ we have the equivalence $A \models \varphi \Leftrightarrow A|_\Omega \models \varphi$.

An OS *equational theory* is a pair $T = (\Sigma, E)$, with E a set of (possibly conditional) Σ-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 *(Σ, E)-algebras*. $\mathbf{OSAlg}_{(\Sigma,E)}$ has an *initial algebra* $T_{\Sigma/E}$ [32]. Given $T = (\Sigma, E)$ and $\varphi \in Form(\Sigma)$, we call φ *T-valid*, written $E \models \varphi$, iff $A \models \varphi$ for all $A \in \mathbf{OSAlg}_{(\Sigma,E)}$. We call φ *T-satisfiable* iff there exists $A \in \mathbf{OSAlg}_{(\Sigma,E)}$ with φ satisfiable in A. Note that φ is *T-valid* iff $\neg\varphi$ is *T-unsatisfiable*. The inference system in

[3] There is no real loss of generality in assuming that all atomic formulas are equations: predicates can be specified by equational formulas using additional function symbols of an added sort *Pred*. See Section 2.5 for a simple example.

[32] is *sound and complete* for OS equational deduction, i.e., for any OS equational theory (Σ, E), and Σ-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*. An *E-unifier* of a system of Σ-equations, i.e., of a conjunction $\phi = u_1 = v_1 \wedge \ldots \wedge u_n = v_n$ of Σ-equations, is a substitution σ such that $u_i\sigma =_E v_i\sigma$, $1 \leqslant i \leqslant n$. An *E-unification algorithm* for (Σ, E) is an algorithm generating a *complete set* of E-unifiers $Unif_E(\phi)$ for any system of Σ equations ϕ, where "complete" means that for any E-unifier σ of ϕ there is a $\tau \in Unif_E(\phi)$ and a substitution ρ such that $\sigma =_E (\tau\rho)|_{dom(\sigma)\cup dom(\tau)}$, where $=_E$ here means that for any variable x we have $x\sigma =_E x(\tau\rho)|_{dom(\sigma)\cup dom(\tau)}$. The algorithm is *finitary* if it always terminates with a *finite set* $Unif_E(\phi)$ for any ϕ.

Given a set of equations B used for deduction modulo B, a preregular OS signature Σ is called *B-preregular*[4] iff for each $u = v \in B$ and substitutions ρ, $ls(u\rho) = ls(v\rho)$.

2.3 Rewriting Logic

We now recall some basic concepts about *rewriting logic*. The survey in [5] gives a fuller account. The key purpose of a rewrite theory $\mathcal{R}$ is to axiomatize a *distributed system*, so that concurrent computation is modeled as concurrent rewriting with the rules of $\mathcal{R}$ *modulo* the equations E of $\mathcal{R}$.

Recall the notation for term positions, subterms, and term replacement from [35]: (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.

Definition 5 (Rewrite Theory). *A* rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ *is a 3-tuple with $(\Sigma, E \cup B)$ an OS equational theory and R a set of (possibly conditional) Σ-*rewrite rules*, i.e., sequents $l \to r$ if ϕ, with $l, r \in T_\Sigma(X)_{[s]}$ for some $[s] \in \widehat{S}$, and ϕ a quantifier-free Σ-formula.*[5]

We further assume that:

1. *Each equation $u = v \in B$ is unconditional,* 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, Σ is B-*preregular *(in the broader sense of Footnote 4).*

[4] If $B = B_0 \uplus U$, with B_0 associativity and/or commutativity axioms, and U identity axioms, the B-preregularity notion can be *broadened* by requiring only that: (i) Σ is preregular; (ii) Σ is B_0-preregular in the standard sense that $ls(u\rho) = ls(v\rho)$ for all $u = v \in B_0$ and substitutions ρ; and (iii) the axioms U oriented as rules $\vec{U}$ are *sort-decreasing* in the sense of Definition 5 below. Maude automatically checks B-preregularity of an OS signature Σ in this broader sense [34].

[5] Usually, ϕ is assumed to be a *conjunction* of Σ-equations. We give here this more general definition for three reasons: (i) often, using *equationally-defined equality predicates* [36], 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 [37].

2. *The (possibly conditional) 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, that is, sort-decreasing, strictly coherent, confluent, and operationally terminating as rewrite rules modulo B [38].*
3. *The rules R are* ground coherent *with the equations E modulo B [39].*

We refer to [5,38,39,40,37] for more details, but give here an intuitive high-level explanation of what the above conditions mean in practice. Conditions (1)–(2) ensure that the initial $(\Sigma, E \cup B)$-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 Σ-terms.

Define the *one-step R, B-rewrite relation* $t \to_{R,B} t'$ between ground terms as follows. For $t, t' \in T_{\Sigma_{[s]}}$, $[s] \in \widehat{S}$, $t \to_{R,B} t'$ holds iff there is a rewrite rule $l \to r \text{ if } \phi \in R$, a ground substitution $\sigma \in [Y \to T_\Sigma]$ with Y the rule's variables, and a term position p in t such that $t|_p =_B l\sigma$, $t' = t[r\sigma]_p$, and $T_{\Sigma/E\cup B} \models \phi\sigma$. Let us explain *convergence* in more detail: (i) *sort-decreasingness* means that whenever $t \to_{\vec{E},B} u$, the least sort of u is smaller than or equal to the least sort of t; (ii) *strict-B-coherence* means that if $t \to_{\vec{E},B} u$ and $t =_B t'$, then there is a rewrite step $t' \to_{\vec{E},B} u'$ such that $u =_B u'$; intuitively, this gives the effect of rewriting in B-equivalence classes; *confluence* modulo B means that for each term t, if we have sequences $t \to^*_{\vec{E},B} u$ and $t \to^*_{\vec{E},B} v$, then there are sequences $u \to^*_{\vec{E},B} w$ and $v \to^*_{\vec{E},B} w'$ such that $w =_B w'$; *operational termination* modulo B means that: (a) there are no infinite sequences of rewrites with $\to_{\vec{E},B}$, and (b) if the equations E are conditional, all attempts to evaluate a condition to perform a rewrite terminate in finite time with either success or failure. All together, convergence means that for any term t there is an $\vec{E}, B$-irreducible term, denoted $t!_{\vec{E},B}$ and abbreviated to $t!$, which is obtained from t by rewriting with $\to_{\vec{E},B}$ until termination that is *unique* up to B-equality and is called t's *canonical form*, or, equivalently, its *normal form*. It also means that for any terms t, t' we have $t =_{E\cup B} t'$ iff $t! =_B t'!$ (Church-Rosser Property). 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 $u = t!_{\vec{E},B}$, 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 $t'! =_B u'!$. Note that *$vars(r) \subseteq vars(l)$ is nowhere assumed* for rules $l \to r \text{ if } \phi \in R$. This means that $\mathcal{R}$ can specify an *open system*, in the sense of [41], that interacts with an external, non-deterministic environment.

Conditions (1)–(3) allow a simple and intuitive description of the *initial reachability model* $\mathcal{T}_\mathcal{R}$ [42] 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 $[u'!] = [v]$. Specifically, under conditions (1)–(3) we then have an *isomorphism* of reachability models [42] $\mathcal{T}_\mathcal{R} \cong \mathcal{C}_\mathcal{R}$, where $\mathcal{C}_\mathcal{R}$ has a much simpler description than $\mathcal{T}_\mathcal{R}$. Furthermore, if $u \to_{R,B} u'$ has been performed with a rewrite rule $l \to r \text{ if } \phi \in R$

and a ground substitution $\sigma \in [Y \to T_\Sigma]$, then, assuming B-equality is decidable, checking whether condition $T_{\Sigma/E\cup B} \models \phi\sigma$ holds is *decidable* by reducing the terms in $\phi\sigma$ to $\vec{E}, B$-canonical form and checking for B-equality.

The signature Σ on which $T_{\Sigma/E\cup B}$ is defined has often a natural decomposition as a disjoint union $\Sigma = \Omega \uplus \Delta$, where the elements of the canonical term algebra $C_{\Sigma/E,B}$ are Ω-terms, whereas the function symbols $f \in \Delta$ are viewed as *defined functions* which are *evaluated away* by $\vec{E}, B$-simplification. Ω (with same poset of sorts as Σ) is then called a *constructor subsignature* of Σ.

A *decomposition* of an order-sorted equational theory $(\Sigma, E \cup B)$ is a rewrite theory $(\Sigma, B, \vec{E})$ such that the rules $\vec{E}$ are convergent modulo B. $(\Sigma, B, \vec{E})$ is called *sufficiently complete* with respect to the *constructor subsignature* Ω (with same poset of sorts as Σ) iff for each $t \in T_\Sigma$ we have: (i) $t!_{\vec{E},B} \in T_\Omega$, and (ii) if $u \in T_\Omega$ and $u =_B v$, then $v \in T_\Omega$. This ensures that for each $[u]_B \in C_{\Sigma/E,B}$ we have $[u]_B \subseteq T_\Omega$. Sufficient completeness is closely related to the notion of a *protecting* inclusion of decompositions.

Definition 6 (Protection, Constructor Decomposition). *Consider theory inclusion* $(\Sigma_0, E_0 \cup B_0) \subseteq (\Sigma, E \cup B)$ *where* $(\Sigma_0, B_0, \vec{E_0})$ *and* $(\Sigma, B, \vec{E})$ *are respective decompositions of theories* $(\Sigma_0, E_0 \cup B_0)$ *and* $(\Sigma, E \cup B)$. *We then say that the decomposition* $(\Sigma, B, \vec{E})$ protects $(\Sigma_0, B_0, \vec{E_0})$ *iff (i) for all* $t, t' \in T_{\Sigma_0}(X)$ *we have: (i)* $t =_{B_0} t' \Leftrightarrow t =_B t'$, *(ii)* $t = t!_{\vec{E_0},B_0} \Leftrightarrow t = t!_{\vec{E},B}$, *and (iii)* $C_{\Sigma_0/E_0,B_0} = C_{\Sigma/E,B}|_{\Sigma_0}$.

$(\Omega, B_\Omega, \vec{E_\Omega})$ *is a* constructor decomposition *of* $(\Sigma, B, \vec{E})$ *iff (i)* $(\Sigma, B, \vec{E})$ *protects* $(\Omega, B_\Omega, \vec{E_\Omega})$, *and (ii)* $(\Sigma, B, \vec{E})$ *is sufficiently complete with respect to the constructor subsignature* Ω. *Furthermore,* Ω *is called a subsignature of* free constructors modulo B_Ω *iff* $E_\Omega = \emptyset$, *so that* $C_{\Omega/E_\Omega,B_\Omega} = T_{\Omega/,B_\Omega}$.

Initial Algebras and Reachability Models. In this work, we crucially exploit the relationships between several kinds of initial models induced by various theories, i.e., initial $\mathcal{R}$-reachability models, initial $(\Sigma, E\cup B)$-algebras, and initial constructor $(\Omega, E_\Omega \cup B_\Omega)$-algebras. In particular, these relationships will be systematically exploited by our *state predicates*, based on the notion of *constrained constructor pattern* (Section 3).

Let us first give an overview of the theories involved and their associated initial algebras. Assuming convergent theories $(\Sigma, E \cup B)$ and $(\Omega, E_\Omega \cup B_\Omega)$ such that $(\Omega, B_\Omega, \vec{E_\Omega})$ is a constructor decomposition of $(\Sigma, B, \vec{E})$, we obtain the diagram in Figure 1 below.

In the figure, the map of theory inclusions on the left induces the Ω-isomorphisms on the right. Indeed, the constructor decomposition ensures that: (i) the unique Ω-homomorphisms $T_{\Omega/E_\Omega\cup B_\Omega} \to T_{\Sigma/E\cup B}|_\Omega$, and $C_{\Omega/E_\Omega,B_\Omega} \to C_{\Sigma/E,B}|_\Omega$ guaranteed by initiality are both Ω-*isomorphisms*, and, (ii) furthermore, the Ω-isomorphism $C_{\Omega/E_\Omega,B_\Omega} \to C_{\Sigma/E,B}|_\Omega$ is actually the *identity*, so that $C_{\Omega/E_\Omega,B_\Omega} = C_{\Sigma/E,B}|_\Omega$. In particular, this means that for each $[t]_{E\cup B} \in T_{\Sigma/E\cup B}$, the map $[t]_{E\cup B} \mapsto [t!_{\vec{E},B}]_{B_\Omega}$ is an Ω-*isomorphism* $T_{\Sigma/E\cup B}|_\Omega \to C_{\Omega/E_\Omega,B_\Omega}$, denoted

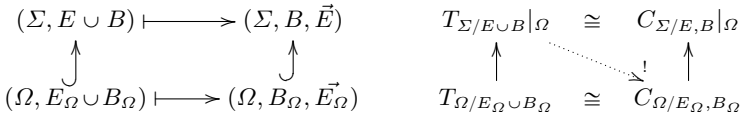

Fig. 1. Theory Inclusions (Left) and Initial Algebra Isomorphisms (Right)

as a dotted diagonal arrow on the right diagram. Such a diagonal arrow involves a *huge reduction* of the, in general very complex, initial algebra $T_{\Sigma/E\cup B}$ to the typically much simpler constructor canonical term algebra $C_{\Omega/E_\Omega,B_\Omega}$. In many practical cases, such an algebra is so simple that $E_\Omega = \varnothing$, so that $C_{\Omega/E_\Omega,B_\Omega} = T_{\Omega/B_\Omega}$. Furthermore, for B_Ω any combination of associativity and/or commutativity and/or identity axioms, except associativity without commutativity, satisfiability of quantifier-free Ω-formulas in T_{Ω/B_Ω} is *decidable* [30]. All this means that:

1. to check whether two Σ-terms are $E \cup B$-equal, we need only normalize by $\rightarrow^!_{\vec{E},B}$ and check B_Ω-equality;
2. checking whether two *constructor terms* (Ω-terms) are $E_\Omega \cup B_\Omega$-equal only requires normalizing by $\rightarrow^!_{\vec{E}_\Omega,B_\Omega}$ and checking B_Ω-equality;
3. in the very common case where $E_\Omega = \varnothing$ and we have a finitary B_Ω-unification algorithm, computing most general unifiers of constructor terms is *decidable* by B_Ω-unification.

Of course, all this lifts to the level of rewrite theories and their initial reachability models. If $\mathcal{R} = (\Sigma, E \cup B, R)$ is a rewrite theory satisfying the requirements in Definition 5 and, furthermore, $(\Omega, B_\Omega, \vec{E}_\Omega)$ is a constructor decomposition of $(\Sigma, B, \vec{E})$, then, the already-defined isomorphism of reachability models $\mathcal{T}_\mathcal{R} \cong \mathcal{C}_\mathcal{R}$ is in more detail an isomorphism $(T_{\Sigma/E\cup B}, \rightarrow_\mathcal{R}) \cong (C_{\Sigma/E,B}, \rightarrow_\mathcal{R})$. But since $C_{\Omega/E_\Omega,B_\Omega} = C_{\Sigma/E,B}|_\Omega$, the reachability model $\mathcal{C}_\mathcal{R}$ is much simpler than $\mathcal{T}_\mathcal{R}$, since its states (and therefore its state predicates) can be specified by *constructor terms*.

2.4 Variants and the Finite Variant Property

Given a convergent equational theory $(\Sigma, E \cup B)$, a *variant* [43,14,44] of a Σ-term t is a pair (u, θ) where θ is a substitution, and u is the $\vec{E}, B$-canonical form of the term instance $t\theta$. Intuitively, the variants of t are the fully simplified *patterns* to which the instances of t can be simplified with the oriented equations E modulo B. Some simplified instances are of course more general (as patterns) than others. $(\Sigma, E \cup B)$ has the *finite variant property* (FVP) in the Comon-Delaune sense [43] iff any Σ-term t has a *finite* set of most general variants. For example, the addition equations $E = \{x + 0 = x, x + s(y) = s(x + y)\}$ are *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 $x+y$. Instead, the

Boolean equations $G = \{x \vee \top = \top, x \vee \bot = x, x \wedge \top = x, x \wedge \bot = \bot\}$ *are* FVP. For example, the most general variants of $x \vee y$ are: $(x \vee y, id)$, $(x, \{y \mapsto \bot\})$, and $(\top, \{y \mapsto \top\})$. Let us define these notions more precisely.

Definition 7 (Variants, FVP). *Given a decomposition* $\mathcal{R} = (\Sigma, B, \vec{E})$ *and a* Σ*-term* t*, a* variant *[43,14] of* t *is a pair* (u, θ) *such that: (i)* $u =_B (t\theta)!_{\vec{E},B}$*, (ii)* $dom(\theta) = vars(t)$*, and (iii)* $\theta = \theta!_{\vec{E},B}$*, that is,* $x\theta = (x\theta)!_{\vec{E},B}$ *for all variables* x*.* (u, θ) *is called a* ground variant *iff, furthermore,* $u \in T_\Sigma$*. Note that if* (u, θ) *is a ground variant of some* t*, then* $[u]_B \in C_{\Sigma/\vec{E},B}$*. Given variants* (u, θ) *and* (v, γ) *of* t*,* (u, θ) *is called* more general *than* (v, γ)*, denoted* $(u, \theta) \sqsupseteq_B (v, \gamma)$*, iff there is a substitution* ρ *such that: (i)* $(\theta\rho)|_{vars(t)} =_B \gamma$*, and (ii)* $u\rho =_B v$*. Let* $[\![t]\!]_{\vec{E},B} = \{(u_i, \theta_i) \mid i \in I\}$ *denote a* complete set of variants *of* t*, that is, a set of variants such that for any variant* (v, γ) *of* t *there is an* $i \in I$*, such that* $(u_i, \theta_i) \sqsupseteq_B (v, \gamma)$*. A decomposition* $\mathcal{R} = (\Sigma, B, \vec{E})$ *of* $(\Sigma, E \uplus B)$ *has the* finite variant property *[43] (FVP) iff for each* Σ*-term* t *there is a* finite *complete set of variants* $[\![t]\!]_{\vec{E},B} = \{(u_1, \theta_1), \ldots, (u_n, \theta_n)\}$*.*

If B has a finitary unification algorithm and $\mathcal{R} = (\Sigma, B, \vec{E})$ is FVP, then for any term t the finite set $[\![t]\!]_{\vec{E},B}$ of its variants can be computed by *folding variant narrowing* [14]. Maude 2.7.1 supports the computation of $[\![t]\!]_{\vec{E},B}$ for B a combination of associative and/or commutative and/or identity axioms.

FVP theories $(\Sigma, B, \vec{E})$ are important for automating important aspects of reachability logic deduction because: (i) if B has a finitary unification algorithm, then $B \cup E$ also has a finitary unification algorithm; and (ii) if, furthermore, $(\Sigma, B, \vec{E})$ has a constructor decomposition $(\Omega, B_\Omega, \vec{E_\Omega})$, then, under a reasonable requirement on $T_{\Omega/E_\Omega \cup B_\Omega}$ called *OS-compactness* [30], satisfiability of quantifier-free formulas in the initial algebra $T_{\Sigma/E \cup B}$ is *decidable* [30,31]. See Section 6 for examples of rewrite theories where verification of their reachability logic properties was made easier by exploiting the fact that their equational part was FVP.

2.5 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 [28], where (Σ, B) defines the protocol's states, involving natural numbers, lists, and multisets over natural numbers. Σ has sorts $S = \{Nat, List, MSet, Conf, State, Pred\}$ with subsorts $Nat < List$ and $Nat < MSet$ and also the set of operators $F = \{0 : \to Nat,\ s\_ : Nat \to Nat,\ nil : \to List,\ \_;\_ : List\ List \to List,\ \varnothing : \to MSet,\ \_\_ : MSet\ MSet \to MSet,\ dupl : MSet \to Pred, tt : \to Pred,\ \_|\_|\_|\_ : MSet\ MSet\ MSet\ List \to Conf, < \_ > : Conf \to State\}$, 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 nil. The only equation in E is $dupl(s\ i\ i) = tt$. It defines the *dupl* predicate by detecting a duplicated element i in the multiset

$s\,i\,i$ (where s could be empty). The *states* of QLOCK are B-equivalence classes of ground terms of sort *State*.

QLOCK [28] 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.

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

where $\phi = dupl(n\,i\,w\,c) \neq tt$, 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 *dupl*—and that $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfies sub-requirements (1)–(3) of Definition 5. Note that *join* makes QLOCK an *open* system in the sense explained above.

3 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 4 will be pairs $u \mid \varphi$, called *constrained constructor patterns*, with u a term in a subsignature $\Omega \subseteq \Sigma$ of constructors, and φ a quantifier-free Σ-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 ρ 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 Σ-formulas.

Definition 8 (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* φ *a QF* Σ*-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 σ is a (sort-preserving) bijective renaming of variables we always have $[\![A]\!] = [\![A\sigma]\!]$.

Example 1 (Pattern Predicate Example). Recall that QLOCK states have the general 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 $< n \mid w \mid c \mid q > \to^* < n' \mid w' \mid c' \mid q' >$ 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 4 and 5. 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) [43,14] and therefore, assuming a B_Ω-unification algorithm, it has a $E_\Omega \cup B_\Omega$-*unification algorithm* computable by folding variant narrowing [14]; (iii) as we shall show in Lemma 4, 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 $l \to r$ *if* ϕ, with l and r Ω-terms, and ϕ a QF Σ-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 $l \to r$ *if* ϕ in $\hat{\mathcal{R}}$ using narrowing techniques [15] based on $E_\Omega \cup B_\Omega$-unification (for more on this, see the explanation of the $\textsc{Step}^\forall$ inference rule in Section 5); 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*.

3.1 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 $[\![B]\!] \supseteq ([\![u \mid \top]\!] \backslash [\![u \mid \phi]\!])$
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 $v \mid \psi$ with $vars(u \mid \varphi) \cap vars(v \mid \psi) = \varnothing$, we say that $u \mid \varphi$ *subsumes* $v \mid \psi$, denoted $v \mid \psi \sqsubseteq u \mid \varphi$, iff there is a substitution α such that: (i) $v =_{E_\Omega \cup B_\Omega} u\alpha$, and (ii) $T_{\Sigma/E \cup B} \models \psi \Rightarrow (\varphi\alpha)$. It then follows easily from the above definition of $[\![u \mid \varphi]\!]$ that if $u \mid \varphi$ subsumes $v \mid \psi$, then $[\![v \mid \psi]\!] \subseteq [\![u \mid \varphi]\!]$. Likewise, $\bigvee_{i \in I} u_i \mid \varphi_i$ *subsumes* $v \mid \psi$, denoted $v \mid \psi \sqsubseteq \bigvee_{i \in I} u_i \mid \varphi_i$, iff there is a $k \in I$ such that $u_k \mid \varphi_k$ subsumes $v \mid \psi$, and we then have $[\![v \mid \psi]\!] \subseteq [\![\bigvee_{i \in I} u_i \mid \varphi_i]\!]$.

Computationally, subsumption is a relatively cheap,[6] 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 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 Σ-formula φ 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 $vars(\varphi) \subseteq vars(u)$.

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

[6] 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_Ω 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_Ω are like that, and $E_\Omega \cup B_\Omega$ has the finite variant property [43,14]; but it will be more expensive. In general, the validity check $T_{\Sigma/E \cup B} \models \psi \Rightarrow (\varphi\alpha)$ 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 [30,31]; and (ii) even though the validity property $T_{\Sigma/E \cup B} \models \psi \Rightarrow (\varphi\alpha)$ 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.

parentheses for clarity, so that, e.g., the multiset a, b, b, c is denoted (a, b, b, c), we have:

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

Nevertheless, since we always have the set identity $[\![u \mid \varphi]\!] \cup [\![u \mid \neg\varphi]\!] = [\![u \mid \top]\!]$, we always also have the *set containment* $[\![u \mid \top]\!] \backslash [\![u \mid \varphi]\!] \subseteq [\![u \mid \neg\varphi]\!]$, giving us a cheap way to *over-approximate* the more complex set difference $[\![u \mid \top]\!] \backslash [\![u \mid \varphi]\!]$ by the simpler term pattern predicate $u \mid \neg\varphi$. Such computationally cheap over-approximations are taken advantage of in reachability logic's inference system (see Section 5).

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. 8 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 $vars(u \mid \varphi) \cap vars(v \mid \psi) = \varnothing$, 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 $\bot$, 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

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

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 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:*

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

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,

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

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 $v \mid \phi$ for which their shared variables $Y = vars(u \mid \varphi) \cap vars(v \mid \phi)$ are indeed parameters, so that the semantic meaning of their *set containment* is the containment of a parametric family of sets, i.e.,

$$\forall \rho \in [Y \to T_\Omega] \quad [\![(v \mid \phi)\rho]\!] \subseteq [\![(u \mid \varphi)\rho]\!].$$

To distinguish this notion of set containment from the standard one, where we may always rename $u \mid \varphi$ and $v \mid \phi$ so that $vars(u \mid \varphi) \cap vars(v \mid \phi) = \varnothing$, we write it as follows: $[\![v \mid \phi]\!] \subseteq_Y [\![u \mid \varphi]\!]$. Under these assumptions, there is a natural notion of *Y-parameterized subsumption* of $v \mid \phi$ by $u \mid \varphi$, denoted $v \mid \phi \sqsubseteq_Y u \mid \varphi$, namely, such subsumption holds iff there is a substitution α such that: (i) $(\forall y \in Y)\ \alpha(y) = y$, (ii) $v =_{E_\Omega \cup B_\Omega} u\alpha$, and (iii) $T_{\Sigma/E \cup B} \models \phi \Rightarrow (\varphi\alpha)$. As for the unparameterized case, a parameterized subsumption $v \mid \phi \sqsubseteq_Y u \mid \varphi$ provides a relatively efficient way of checking the parameterized inclusion $[\![v \mid \phi]\!] \subseteq_Y [\![u \mid \varphi]\!]$. The notions of parameterized intersection and parameterized containment will be used in Section 4.1 to reason about *parameterized* invariants and co-invariants, and in Section 5 to perform inferences in reachability logic.

4 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., [15]). Besides satisfying the requirements in Definition 5, $\mathcal{R}$ should also satisfy the requirements in Definition 9 below.

Definition 9 (Suitable Rewrite Theories). *We say a rewrite theory* $\mathcal{R} = (\Sigma, E \cup B, R)$ *satisfying the requirements in Definition 5 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*[7] *and (ii) the axioms* B_Ω *are linear.*
2. Σ *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,[8] *i.e. they satisfy* $vars(l) \backslash (vars(r) \cup vars(\phi)) = \varnothing$*.*

Requirements (1)–(3) in Definition 9 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 9, 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 10 (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 8. 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 9. 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 [3] 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

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

[8] 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 $M \to null \text{ if } M = M$.

the most general and expressive, and the one-path semantics applies mostly to sequential systems, in this work we focus on the all-paths semantics.

The reachability logic in [2,3] 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 4.1, 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 $\mathcal{R} \models^{\forall} A \rightarrow^{\circledast} B$ to a *relativized* satisfaction relation $\mathcal{R} \models^{\forall}_{T} A \rightarrow^{\circledast} B$, where T is a constrained pattern predicate such that $[\![T]\!]$ is a subset of the total 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 $\rightarrow_{\mathcal{R}}$-transitions *maximal* iff it cannot be extended. 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 11 (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 pattern predicate T we use does in fact define the standard relation, 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 [45] 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 12 (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

$u \mid \varphi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ with parameters Y, $\mathcal{R} \models^{\forall}_T u \mid \varphi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ holds if $\mathcal{R} \models^{\forall}_T (u \mid \varphi)\rho \to^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\rho$ holds for each $\rho \in [Y \to T_\Omega]$.

Since a constrained pattern predicate is equivalent to a disjunction of atomic ones, we can define satisfaction on general reachability logic formulas as follows: $\mathcal{R} \models^{\forall}_T \bigvee_{1\leqslant i\leqslant n} u_i \mid \varphi_i \to^{\circledast} A$ iff $\bigwedge_{1\leqslant i\leqslant n} \mathcal{R} \models^{\forall}_T u_i \mid \varphi_i \to^{\circledast} A$, assuming same parameters $Y_i = vars(u_i \mid \varphi_i) \cap vars(A)$, i.e., $Y_i = Y_{i'}$ for $1 \leqslant i < i' \leqslant n$.

$\mathcal{R} \models^{\forall}_T A \to^{\circledast} B$ 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 4.2 for a generalized Hoare logic), we reserve the term *postcondition* for a midcondition B in a reachability formula $A \to^{\circledast} B$ 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 $u \mid \varphi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$. 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 $\mathcal{R} \models^{\forall}_T u \mid \varphi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ holds iff:

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

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

Lemma 2. *(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 α such that $dom(\alpha) \subseteq U$ and $ran(\alpha) \cap (U \cup Z) = \varnothing$, the formula $(u \mid \varphi \to^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j)\alpha$ has parameters $vars(\alpha(Y))$.*

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

Definition 13 (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 $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$*, 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:*

$$(\forall i)\ (\forall \alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u, t_i))\ \ [\![(u \mid \varphi \wedge \chi_i)\alpha]\!] \subseteq_{vars(\alpha(Y))} [\![(\bigvee_{j\in J} v_j \mid \phi_j)\alpha]\!].$$

A trivial reachability formula $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ is called so because all states in its precondition have already reached the midcondition, and therefore $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ 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: $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ will be T*-inconsistent* iff there is a ground substitution ρ 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 3. *If* $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$ *with parameters* Y *is* T*-consistent and* $[\![u \mid \varphi]\!] \subseteq \mathit{Term}_{\mathcal{R}}$*, then* $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j$.

Recall that in requirement (3) for a suitable rewrite theory $\mathcal{R}$ we assumed unforgetful topmost rewrite rules of the form $l \rightarrow r$ *if* ϕ with $l \notin T_\Omega(X)$. For symbolic reasoning purposes it will be very useful to also require that $r \notin T_\Omega(X)$. This can be done without any real loss of generality by means of a theory transformation[9] $\mathcal{R} \mapsto \hat{\mathcal{R}}$ 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 $l \rightarrow r$ *if* ϕ in R into the rule $l \rightarrow r'$ *if* $\phi \wedge \hat{\theta}$, where: (i) r' is the Ω*-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 4. *The canonical reachability models* $\mathcal{C}_{\mathcal{R}}$ *and* $\mathcal{C}_{\hat{\mathcal{R}}}$ *are* identical.

4.1 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 $\mathcal{S} = (S, \rightarrow_{\mathcal{S}})$ with S its set of *states* and $\rightarrow_{\mathcal{S}} \subseteq S \times S$ its *transition*

[9] 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 6, is presented in [37,46].

relation. Given a set of "initial states" $S_0 \subseteq S$, the set $Reach(S_0)$ of states *reachable* from S_0 is defined as $Reach(S_0) = \{s \in S \mid (\exists s_0 \in S_0)\ s_0 \rightarrow_{\mathcal{S}}^* s\}$, where $\rightarrow_{\mathcal{S}}^*$ denotes the reflexive-transitive closure of $\rightarrow_{\mathcal{S}}$. 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 $Reach(S_0) \subseteq P$, or (ii) as a "bad" property $Q \subseteq S$, the *co-invariant*, that *never holds* from S_0, i.e., such that $Reach(S_0) \cap Q = \varnothing$. 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 $S \backslash P$.

All this is particularly relevant for the transition system $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ 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 $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$. Can we specify such invariant or co-invariant by means of *reachability formulas* and use the inference system of Section 5 to try to prove such formulas?

Suppose $\mathcal{R}$ specifies a *never-terminating system*, i.e., a system such that $Term_{\mathcal{R}} = \varnothing$. Many distributed systems are never-terminating. For example, QLOCK and other mutual exclusion protocols are never-terminating. Then the set $Term_{\mathcal{R}} = \varnothing$, and $\mathcal{R} \models_T^{\forall} A \rightarrow^{\circledast} B$ holds vacuously for *all* reachability formulas $A \rightarrow^{\circledast} B$ 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. We just need to first perform a simple *theory transformation*. To ease the exposition, we explain the transformation in case Ω has a single state constructor, say, $\langle \_, \ldots, \_ \rangle : s_1, \ldots, s_n \rightarrow State$. 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 $[\_, \ldots, \_] : s_1, \ldots, s_n \rightarrow State$ to Ω, and (2) a new rewrite rule $stop : \langle x_1{:}s_1, \ldots, x_n{:}s_n \rangle \rightarrow [x_1{:}s_1, \ldots, x_n{:}s_n]$ 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 $\langle \_, \ldots, \_ \rangle : s_1, \ldots, s_n \rightarrow State$ 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 3 (Invariants). *For $S_0, P \in PatPred(\Omega,\Sigma)$ constrained pattern predicates in standard form with $vars(S_0) \cap vars(P) = \emptyset$, $[\![P]\!]$ is an invariant of $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$ iff $\mathcal{R}_{stop} \models^{\forall}_{[]} S_0 \rightarrow^{\circledast} [P]$.*

The notion of a *parametric invariant* can be reduced to the unparameterized one: if $Y = vars(S_0) \cap vars(P)$, then $[\![P]\!]$ is an *invariant* of $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$ *with parameters* Y iff $\mathcal{R}_{stop} \models^{\forall}_{[]} S_0 \rightarrow^{\circledast} [P]$. That is, iff $[\![P\rho]\!]$ is an (unparameterized) invariant of $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0\rho]\!]$ for each $\rho \in [Y \rightarrow T_\Omega]$. In this way, Theorem 3 extends seamlessly to parametric invariants.

Example 2 (Specifying Invariants for QLOCK). As an example, we consider how to specify invariants as reachability formulas using the QLOCK specification from Sections 2 and 3. 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 3, we can specify invariants by reachability formulas. Define the set of initial states containing only normal processes by the pattern predicate $S_0 = < n' \mid \emptyset \mid \emptyset \mid nil > \mid dupl(n') \neq tt$. 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 $< n \mid w \mid \emptyset \mid q > \vee < n \mid w \mid i \mid i \ ; \ q >$. We need also to ensure our multisets are actually *sets*. Thus, we define the constructor pattern predicates $P_1 = \big(< n \mid w \mid \emptyset \mid q > \mid dupl(n\ w) \neq tt\big)$ 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 3, QLOCK ensures mutual exclusion from $[\![S_0]\!]$ iff $\mathcal{R}_{stop} \models^{\forall}_{[]} S_0 \rightarrow^{\circledast} [P]$ where here $[P]$ is $[P_1] \vee [P_2]$, i.e. $\big([\ n \mid w \mid \emptyset \mid q\ ] \mid dupl(n\ w) \neq tt \vee [\ n \mid w \mid i \mid i \ ; \ q\ ] \mid dupl(n\ w\ i) \neq tt\big)$.

The following easy corollary can be very helpful in proving invariants. It can, for example, be applied to prove the mutual exclusion of QLOCK.

Corollary 1. *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 an invariant of $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$ with parameters Y if: (i) $[\![S_0]\!] \subseteq_Y [\![P]\!]$ (see Section 3.1), and (ii) $\mathcal{R}_{stop} \models^{\forall}_{[]} P \rightarrow^{\circledast} [P\sigma]$, where σ is a sort-preserving bijective renaming of variables such that σ is the identity on Y and $vars(P) \cap vars(P\sigma) = Y$.*

Let us now turn to the case of co-invariants. Suppose we have specified constrained pattern predicates S_0 and Q and we want to prove that $[\![Q]\!]$ is a *co-invariant* of the system $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$. Can this be expressed by some reachability formula or formulas? The answer is yes! By using the rules of $\mathcal{R}$ *backwards*. Assume without loss of generality 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[10] with the equations E modulo

[10] Ground coherence of R^{-1} may be problematic due to the fact that, while a rule's the lefthand side l is assumed to be a constructor term, its righthand side r could

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 key result.

Theorem 4. *Under the above assumptions on $\mathcal{R}^{-1}$, if $S_0, Q \in PatPred(\Omega, \Sigma)$ are constrained pattern predicates in standard form with $vars(S_0) \cap vars(Q) = \varnothing$, then $[\![Q]\!]$ is a co-invariant of $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![S_0]\!]$ if: (i) $[\![Q]\!] \cap [\![S_0]\!] = \varnothing$, and (ii) $(\mathcal{R}^{-1})_{stop} \models^{\forall}_{[]} Q \rightarrow^{\circledast} [Q\sigma]$, where σ is a bijective renaming of variables such that $vars(Q) \cap vars(Q\sigma) = \varnothing$.*

The reduction of parametric invariants to unparameterized ones applies *mutatis mutandis*, to parametric co-invariants. For example, in the parametric version of the above theorem, where $Y = vars(S_0) \cap vars(Q)$, the checking of (i) now becomes checking $[\![Q]\!] \cap_Y [\![S_0]\!] = \varnothing$ (see Section 3.1); and for (ii), proving the reachability formula $Q \rightarrow^{\circledast} [Q\sigma]$, where σ is a bijective renaming of variables such that $vars(Q) \cap vars(Q\sigma) = Y$ and $\sigma(y) = y$ for each $y \in Y$.

4.2 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 [47].

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

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

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 $[u] \rightarrow_{\mathcal{R}}! [v]$, $[v] \in [\![B]\!]$. 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 $\rho \in [X \rightarrow T_\Omega]$.

---

be an arbitrary Σ-term involving defined function symbols that will typically give rise to coherence-type critical pairs between such a rule and the theory's equations E modulo B [39]. However, as explained right before Lemma 4 and proved in such lemma, there is no real loss of generality in assuming that r is also a constructor term, because this can always be achieved by a semantics-preserving transformation $\mathcal{R} \mapsto \hat{\mathcal{R}}$. Coherence-type critical pairs almost never arise in practice between a constructor-based rule and equations E. Therefore, the ground coherence assumption for R^{-1} is very reasonable assuming our theory is already of the form $\hat{\mathcal{R}}$.

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 $\{\langle p : \mathit{init}\rangle \mid \widetilde{\varphi}\}\ \mathcal{R}_{\mathcal{L}}\ \{\langle \mathit{skip} : S\rangle \mid \widetilde{\psi}\}$, 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 φ and ψ is essentially straightforward, but becomes complicated by the regrettable systematic confusion of *program* variables with *mathematical* variables in φ and ψ. 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 $\{\langle$`x := n ; factp` $: \mathit{init}\rangle \mid n \geqslant 0\}\ \mathcal{R}_{\mathcal{L}}\ \{\langle \mathit{skip} : S\rangle \mid S[\mathtt{y}] = n!\}$, 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, a Hoare triple $\{A\}\ \mathcal{R}\ \{B\}$ is just syntactic sugar for the reachability formula[11] $A \rightarrow^{\circledast} (B \wedge T)$, where $[\![T]\!] = \mathit{Term}_{\mathcal{R}}$, and we assume $\mathit{vars}(T) \cap (\mathit{vars}(A) \cup \mathit{vars}(B)) = \varnothing$. Indeed, we then have:

$$\mathcal{T}_{\mathcal{R}} \models \{A\}\ \mathcal{R}\ \{B\} \quad \Leftrightarrow \quad \mathcal{R} \models^{\forall} A \rightarrow^{\circledast} (B \wedge T).$$

When the above comparison is applied to programming languages (see also [48]), 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 [49] 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:[12] only reachability logic's few inference rules (see Section 5), which are rewrite-

[11] Admittedly, the pattern predicate $B \wedge T$ is not a disjunction of constrained patterns. However, by handling substitutions α in disjoint unifiers as extra conjunctions of equalities $\hat{\alpha}$, we can transform $B \wedge T$ into a disjunction of constrained patterns without affecting the original parameters Y of $\{A\}\ \mathcal{R}\ \{B\}$. Alternatively, by the technique used in (3) of Definition 13 we could expand out $A \rightarrow^{\circledast} (B \wedge T)$ into a conjunction of unifier-instantiated reachability formulas.

[12] See [3] for strong evidence about the advantages of the language-generic nature of reachability logic applied to programming languages within the $\mathbb{K}$ framework.

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 12, 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 $\mathcal{L}$ [50,6].

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 [51] 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:

$$C_{\Sigma/E,B,\mathit{State}} \ni [u] \mapsto \{A \in \mathit{PatPred}(\Omega, \Sigma)_{\mathit{State}} \mid [u] \in [\![A]\!]\} \in \mathcal{P}(\mathit{PatPred}(\Omega, \Sigma)_{\mathit{State}}).$$

Note the useful fact that $\mathcal{K}_{\mathcal{R}}$ can give semantics not only to *propositional* LTL formulas φ, but also to *universal quantifications* $(\forall Y)\, \varphi$ of propositional LTL formulas φ, where Y is a (possibly empty) finite set of variables typed in the signature Σ of $\mathcal{R}$. Indeed, we can *define*, for each $[u] \in C_{\Sigma/E,B,\mathit{State}}$,

$$\mathcal{K}_{\mathcal{R}}, [u] \models_{\mathit{LTL}} (\forall Y)\, \varphi \;\; \Leftrightarrow \;\; \forall \rho \in [Y \rightarrow T_{\Omega}]\; \mathcal{K}_{\mathcal{R}}, [u] \models_{\mathit{LTL}} \varphi\rho.$$

The comparison with LTL then also 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 $(\forall Y)\; A \rightarrow \mathit{en}_{\mathcal{R}}\, \mathcal{W}\, B$, where $\mathcal{W}$ is the "weak until" operator, and if $R = \{l_i \rightarrow r_i \;\mathit{if}\; \varphi_i\}_{i \in I}$, then $\mathit{en}_{\mathcal{R}}$ is the "enabledness" pattern predicate $\mathit{en}_{\mathcal{R}} = \bigvee_{i \in I} l_i \mid \varphi_i$. Indeed, we have:

$$\mathcal{R} \models^{\forall} \; A \rightarrow^{\circledast} B \;\; \Leftrightarrow \;\; \mathcal{K}_{\mathcal{R}} \models_{\mathit{LTL}} (\forall Y)\; A \rightarrow \mathit{en}_{\mathcal{R}}\, \mathcal{W}\, B.$$

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 $(\forall Y)\; A \rightarrow (\neg T)\, \mathcal{W}\, B$.

Note, finally, that thanks to the results in Section 4.1, reachability logic can also express universal LTL *safety formulas* of the form: $(\forall Y)\ A \rightarrow \Box B$ (with $A, B \in PatPred(\Omega, \Sigma)_{State}$ and $Y = vars(A) \cap vars(B)$), since we have:

$$\mathcal{R}_{stop} \models^{\forall}_{[]} A \rightarrow^{\circledast} [B] \ \Leftrightarrow\ \mathcal{K}_{\mathcal{R}} \models_{LTL} (\forall Y)\ A \rightarrow \Box B.$$

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, 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 paper cannot be used as generic (quantified) LTL proof methods. This illustrates the usual tension between the generality of a logic and the effectiveness of its mechanization.

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 $(\forall Y)\ A \rightarrow \Box B$ supported, but so are also *parametric eventuality properties* such as $(\forall Y)\ A \rightarrow en_{\mathcal{R}}\, \mathcal{W}\, B$, stating that all terminating paths starting at A eventually reach B for each ground instantiation of the parameters Y.

5 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 $R = \{l_j \rightarrow r_j \text{ if } \phi_j\}_{j \in J}$ 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 13) 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 $[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$, 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 $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ 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 $\models^{\forall}_T$ this means that any variable in ψ_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 $\{[\mathcal{L},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i \ \mid\ (u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i) \in \mathcal{C}\}$. 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 $\textsc{Step}^{\forall}$ rule.

A very useful feature of the inference system is that sequents $[\mathcal{L},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$, whose formulas $\mathcal{C}$ have been *postulated* (as the conjectures we want to prove) but not yet justified, are transformed by $\textsc{Step}^{\forall}$ into sequents of the form $[\mathcal{L} \cup \mathcal{C},\ \varnothing]\ \vdash_T\ u' \mid \varphi' \rightarrow^{\circledast} \bigvee_i v'_i \mid \psi'_i$, where now the formulas in $\mathcal{C}$ *can be assumed valid*, and can be used in derivations with the $\textsc{Axiom}$ rule.

Example 3 (Conjectures for QLOCK's Mutual Exclusion). By Corollary 1, mutual exclusion of QLOCK can be verified by: (i) using pattern subsumption to check the trivial inclusion $[\![S_0]\!] \subseteq [\![P]\!]$, and (ii) proving $\mathcal{R}_{stop} \models^{\forall}_{[\,]} P \rightarrow^{\circledast} [P\sigma]$, where σ is a sort-preserving bijective renaming of variables such that $vars(P) \cap vars(P\sigma) = \varnothing$. For QLOCK, we had the following 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 $P \rightarrow^{\circledast} [P\sigma]$ naturally splits into two corresponding reachability formulas $P_1 \rightarrow^{\circledast} [P\sigma]$ and $P_2 \rightarrow^{\circledast} [P\sigma]$ shown below:

$$< n \mid w \mid i \mid i\ ;\ q > \mid \varphi \rightarrow^{\circledast} [< n' \mid w' \mid i' \mid i'\ ;\ q' > \mid \varphi' \vee < n' \mid w' \mid \varnothing \mid q' > \mid \psi']$$

$$< n \mid w \mid \varnothing \mid q > \mid \psi \rightarrow^{\circledast} [< n' \mid w' \mid i' \mid i'\ ;\ q' > \mid \varphi' \vee < n' \mid w' \mid \varnothing \mid q' > \mid \psi']$$

where $\varphi = dupl(n\ w\ i) \neq tt$, $\psi = dupl(n\ w) \neq tt$, and φ', ψ' 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 $\mathcal{C} = \{P_i \rightarrow^{\circledast} \bigvee_{i \in I} [P_i\sigma]\}_{i \in I}$

Recall from Definition 13 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 3, 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.
2. As shown by an example in Section 5.2, the AXIOM inference rule is so powerful that, if unwisely used, it can generate invalid, and indeed T-inconsistent,

subgoals *from* valid ones.[13] This is a further reason to always check that all goals are T-consistent.

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 13, and therefore valid. It is a conditional rule of the form:

SUBSUMPTION

$$\frac{}{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i}$$

subject to the condition of showing that $u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \psi_j$ is a *trivial* formula by either: (i) showing that φ is *unsatisfiable*[14] 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 4, 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 φ 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 6. 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 5.1 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 $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ with parameters

[13] 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.

[14] 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 [30]. In general, however, we can only assume $\vec{E}$ convergent modulo B, so that satisfiability of a QF formula φ 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.

Y. We define:

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

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 β 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 $(i, [\beta]_{E_\Omega \cup B_\Omega})$.

Let $R = \{l_j \to r_j \text{ if } \phi_j\}_{j\in J}$. We likewise define:

$$\text{UNIFY}(u \mid \varphi',\ R) \equiv \{(j,\alpha) \mid \alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u, l_j)\}$$

a complete set of $E_\Omega \cup B_\Omega$-unifiers[15] of a pattern $u \mid \varphi'$ with the lefthand-sides of the rules in R.

Consider now the rule:

$\text{STEP}^\forall$

$$\frac{\bigwedge_{(j,\alpha)\in \text{UNIFY}(u\mid\varphi',\ R)} [\mathcal{A} \cup \mathcal{C},\ \varnothing]\ \vdash_T\ (r_j \mid \varphi' \wedge \phi_j)\alpha \to^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha}{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i}$$

where the above conjunction symbol abbreviates a *set of goals*[16] that need to be proved as hypotheses, and where $\varphi' \equiv \varphi \wedge \bigwedge_{(i,\beta)\in \text{MATCH}(u,\ \{v_i\}, Y)} \neg(\psi_i\beta)$. 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:

1. Any state in the set $[\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!]$ automatically satisfies the formula $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$. Therefore, $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ holds iff it does for states in $[\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$.
2. Furthermore, since $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ has been checked T-consistent, it holds iff it does for all T-terminating sequences *of length 1 or more* starting in a state in $[u_0] \in [\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$. That is, $[u_0]$ has an $\mathcal{R}$-successor $[u_1]$ in such a sequence.
3. Using the definition of over-approximating complement in Sec. 3.1 as well as the arguments in Theorem 5, the subset $[\![u \mid \varphi']\!] \subseteq [\![u \mid \varphi]\!]$ *over-approximates* the set of states $[\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$. Thus, this inference rule

[15] 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.

[16] 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.

unifies $u \mid \varphi'$ with the lefthand sides of rules in R, so that all $\mathcal{R}$-successors of states $[u_0] \in [\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$ are over-approximated by one of the pattern predicates $(r_j \mid \varphi' \wedge \phi_j)\alpha$.

Note that formulas in $\mathcal{C}$ are added to $\mathcal{A}$, so that from now on they can be used by AXIOM. By using $E_\Omega \cup B_\Omega$-unification, this inference rule is actually performing one step of *constrained narrowing* of $u \mid \varphi'$ with the (possibly conditional) rules R modulo $E_\Omega \cup B_\Omega$, in the sense of [46], to symbolically compute the new set of preconditions $\{(r_j \mid \varphi' \wedge \phi_j)\alpha \mid (j, \alpha) \in \text{UNIFY}(u \mid \varphi',\ R)\}$ obtained by one-step transitions with the rules R.

AXIOM

$$\frac{\bigwedge_j [\mathcal{A},\ \mathcal{C}]\ \vdash_T\ v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i}{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i}$$

where $(u' \mid \varphi' \rightarrow^{\circledast} \bigvee_j v'_j \mid \psi'_j) \in \mathcal{A}$ has parameters Y', and the substitution α 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 α 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 $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ and $u' \mid \varphi' \rightarrow^{\circledast} \bigvee_j v'_j \mid \psi'_j$ 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. AXIOM allows us to use a trusted formula in $\mathcal{A}$ to summarize multiple transition steps. Since φ is stronger than $\varphi'\alpha$, for each j we add φ to $(v'_j \mid \psi'_j)\alpha$ (the result of using axiom $u' \mid \varphi' \rightarrow^{\circledast} \bigvee_j v'_j \mid \psi'_j$). To find the matching substitution α more easily, in automatic applications of 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 5.2 for an example). The fully general application of AXIOM can be performed by a user command that provides the required matching substitution α 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 given theory $\mathcal{R}$, a T-consistent sequent $[\mathcal{A},\ \mathcal{C}']\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ 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 (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 $\mathcal{R} \models^\forall_T \mathcal{L}$, then $\mathcal{R} \models^\forall_T \mathcal{C}$.*

5.1 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:

$$\frac{\mathcal{G}}{\mathcal{G}'}$$

where $\mathcal{G}$ and $\mathcal{G}'$ are *semantically equivalent* sets of goals, in the sense that $\mathcal{R} \models^\forall_T \mathcal{G} \ \Leftrightarrow\ \mathcal{R} \models^\forall_T \mathcal{G}'$. 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* [52] capabilities. That is, we can view the semantic equivalence $\mathcal{R} \models^\forall_T \mathcal{G} \ \Leftrightarrow\ \mathcal{R} \models^\forall_T \mathcal{G}'$ 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., [52,53,54]). 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 Σ-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 $\mathcal{R} \models^\forall_T \mathcal{G} \ \Leftrightarrow\ \mathcal{R} \models^\forall_T \mathcal{G}'$. This means that SPLIT *does not affect soundness.*

SPLIT

$$\frac{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \psi \rightarrow^{\circledast} A \qquad [\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \phi \rightarrow^{\circledast} A}{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} A}$$

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. *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 ϕ 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 ϕ. 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 α 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 $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ into $u \mid \varphi \wedge (\varphi'\alpha) \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ and $u \mid \varphi \wedge \neg(\varphi'\alpha) \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$, 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 $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \phi_i$ by the equivalent goal $u \mid \psi \rightarrow^{\circledast} \bigvee_i v_i \mid \phi_i$. 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 $T_{\Omega,s} = \bigcup_{1\leqslant i\leqslant k}\{u_i\rho \mid \rho \in [X \rightarrow T_\Omega]\}$. 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

$$\frac{\bigwedge_{1\leqslant i\leqslant k} [\mathcal{A},\ \mathcal{C}]\ \vdash_T\ (u \mid \varphi)\{x{:}s \mapsto u_i\} \rightarrow^{\circledast} A\{x{:}s \mapsto u_i\}}{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} A}$$

where $x{:}s \in vars(u)$ and $\{u_1, \ldots, u_k\}$ is a pattern set for s.

Lemma 6. *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 $T_{\Sigma/E\cup B}|_{\Sigma_1} \cong T_{\Sigma/E\cup B}$ and $\bigwedge_i w_i = w'_i$ is a conjunction of Σ_1-equations. Substitution is the conditional inference rule:

Substitution

$$\frac{\bigwedge_{\alpha \in Unif_{E_1\cup B_1}(\bigwedge_i w_i = w'_i)} [\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u\alpha \mid \varphi\alpha \wedge \widehat{\alpha} \rightarrow^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\alpha}{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \bigwedge_i w_i = w'_i \wedge \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j}$$

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 $\widehat{\alpha} = x_1 = \alpha(x_1) \wedge, \ldots \wedge x_k = \alpha(x_k)$, with the same freshness assumptions as in Footnote 15 for the unifiers $\alpha \in Unif_{E_1\cup B_1}(\bigwedge_i w_i = w'_i)$.

Lemma 7. *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 7, given in Appendix A, uses the following lemma, which is of general interest and is also proved in Appendix A, as an auxiliary lemma:

Lemma 8. *(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* β *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:

$$[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \rightarrow^{\circledast} \bigvee_{j\in J} v_j \mid \phi_j \quad [\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u\beta \mid \psi \rightarrow^{\circledast} (\bigvee_{j\in J} v_j \mid \phi_j)\beta$$

appear in two different *leaves* of the partial proof tree, with, say, Y the parameters of the more general subgoal, β satisfying the conditions in the Instance Lemma 8, $vars(u\beta \mid \psi) \cap vars((\bigvee_{j\in J} v_j \mid \phi_j)\beta) = vars((u \mid \varphi)\beta) \cap vars((\bigvee_{j\in J} v_j \mid \phi_j)\beta) = vars(\beta(Y))$, 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 $(\mathcal{R} \models^{\forall}_{T} (u \mid \varphi)\beta \rightarrow^{\circledast} \bigvee_{j\in J}(v_j \mid \phi_j)\beta) \Rightarrow (\mathcal{R} \models^{\forall}_{T} u\beta \mid \psi \rightarrow^{\circledast} \bigvee_{j\in J}(v_j \mid \phi_j)\beta)$.

5.2 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 Ω with constants $0, 1$ and binary operator $\_ + \_$ of sort *Nat*, constants $\top, \bot$ of sort *Bool*, and a cell constructor $\langle \_ \rangle : \mathit{Nat} \rightarrow \mathit{Counter}$. The signature Σ extends Ω 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 (Ω, B_Ω) is decidable by variant satisfiability, satisfiability of QF Σ-formulas in $T_{\Sigma/E\cup B_\Omega}$ is also decidable [30]. The rewrite rules R defining the semantics of this simple counter system are: $\langle n + 1 \rangle \rightarrow \langle n \rangle$ and $\langle n + 1 \rangle \rightarrow \langle n + 1 + 1 \rangle$. 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 $\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$, which is actually in the Hoare logic fragment and can be proved as follows. We start with the sequent

$$[\varnothing,\ \{\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top\}]\ \vdash_T\ \langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$$

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:

$$[\{\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top\},\ \varnothing]\ \vdash_T\ \langle n' \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$$

$$[\{\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top\},\ \varnothing]\ \vdash_T\ \langle n'' + 1 + 1 \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$$

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

$$[\{\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top\},\ \varnothing]\ \vdash_T\ \langle 0 \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$$

which can be immediately closed by the SUBSUMPTION rule, thus proving the partial correctness property $\mathcal{R} \models^{\forall}_{T} \langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$.

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 $\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$, parametric on m, which can be proved as follows. We start with the sequent

$$[\varnothing,\ \{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\}]\ \vdash_{T}\ \langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$$

Applying the STEP$^{\forall}$ rule we get the sequents:

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle n' \rangle \mid n'+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$$

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle n''+1+1 \rangle \mid n''+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$$

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:

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle 0 \rangle \mid 0+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$$

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle k+1 \rangle \mid k+1+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$$

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:

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle 0 \rangle \mid m = 0 \rightarrow^{\circledast} \langle m \rangle \mid \top$$

Then SUBSTITUTION lets us solve the constraint $m = 0$ with $m \mapsto 0$, giving:

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle 0 \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$$

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:

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle k+1 \rangle \mid m = k+1 \rightarrow^{\circledast} \langle m \rangle \mid \top$$

$$[\{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\},\ \varnothing]\ \vdash_{T}\ \langle k+1 \rangle \mid k+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$$

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 $\mathcal{R} \models^{\forall}_{T} \langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$.

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 $A \rightarrow^{\circledast} B$ such that $[\![C]\!] \subseteq [\![A\alpha]\!]$ to a goal $C \rightarrow^{\circledast} D$ with $C \equiv u \mid \phi$ to "fast forward" and reduce the proof of $C \rightarrow^{\circledast} D$ to that of $(B\alpha) \wedge \varphi \rightarrow^{\circledast} D$. 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 $\mathcal{R} \models^{\forall}_{T} \langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$, we will be better off using the already proved partial correctness property $\mathcal{R} \models^{\forall}_{T} \langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top$ as an axiom. That is, we start with the sequent:

$$\begin{array}{c}\left[\{\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top\}, \{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\}\right] \\ \vdash_{T} \\ \langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\end{array}$$

Then, one application of AXIOM yields the sequent:

$$\begin{array}{c}\left[\{\langle n \rangle \mid \top \rightarrow^{\circledast} \langle 0 \rangle \mid \top\}, \{\langle n+1 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\}\right] \\ \vdash_{T} \\ \langle 0 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top\end{array}$$

But $\mathcal{R} \not\models^{\forall}_{T} \langle 0 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$, since this formula is parameterized by m, and, therefore, if it were valid, we should in particular have for $\rho = \{m \mapsto 1\}$ that $\mathcal{R} \models^{\forall}_{T} \langle 0 \rangle \mid n+1 > 1 \rightarrow^{\circledast} \langle 1 \rangle \mid \top$. 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 $\langle 0 \rangle \mid n+1 > m \rightarrow^{\circledast} \langle m \rangle \mid \top$ is *T-inconsistent*.

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.

5.3 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 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:

$$< n \mid w \mid \varnothing \mid q > | \psi \rightarrow^{\circledast} [< n' \mid w' \mid i' \mid i';\ q' > | \varphi' \vee < n' \mid w' \mid \varnothing \mid q' > | \psi']$$

$$< n \mid w \mid i \mid i\ ;\ q > | \varphi \rightarrow^{\circledast} [< n' \mid w' \mid i' \mid i';\ q' > | \varphi' \vee < n' \mid w' \mid \varnothing \mid q' > | \psi']$$

where $\varphi = \mathit{dupl}(n\ w\ i) \neq tt$, $\psi = \mathit{dupl}(n\ w) \neq tt$, and φ', ψ' 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:

$$c2n : < n \mid w \mid c\ i \mid i\ ;\ q > \ \rightarrow\ < n\ i \mid w \mid c \mid q >$$

The precondition of goal P_2 is $< n \mid w \mid i \mid i\ ;\ q > \mid dupl(n\ w\ i) \neq tt$. 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 $K = < n\ i \mid w \mid \varnothing \mid q' > \mid dupl(n\ w\ i) \neq tt$ (in our implementation, this unification proceeds slightly differently due to A/U unification; see Sec. 6.2).

At this point, our sequent will be of the form $[\mathcal{C},\ \varnothing]\ \vdash_T\ K \rightarrow^{\circledast} \bigvee_{j \in I} P'_j$, and we can apply the AXIOM rule with $P_1 \in \mathcal{C}$. To see this, note that the precondition of P_1, $< n \mid w \mid \varnothing \mid q > \mid dupl(n\ w) \neq tt$ covers our goal by the substitution $n \mapsto n\ i, q \mapsto q'$, where we have to prove the validity of the implication $dupl(n\ w\ i) \neq tt \Rightarrow dupl(n\ w\ i) \neq tt$, which is a tautology.

6 Prototype Implementation and Experiments

We have implemented the reachability logic proof system in Maude [34]. 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 [34]. Our prototype tool takes as input (i) a reflected rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R, \phi)$ and (ii) a set of reachability formulas $\mathcal{C} = \{A_i \rightarrow^{\circledast} B_i\}_{i \in I}$ 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, plus some global state information (for example, the theory $\mathcal{R}$). Given goal set $\mathcal{C}$, the initial proof state will be $\left\{[\varnothing,\ \mathcal{C}]\ \vdash_T\ A_i \rightarrow^{\circledast} B_i\right\}_{i \in I}$, 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_Ω-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 φ a QF Σ-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 Σ_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 Σ-formulas in $T_{\Sigma/E\cup B}$ is decidable by variant satisfiability, we use a variant satisfiability-based backend using techniques in [30,31]. 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 [43] and protects a constructor subtheory which we assume consists only of axioms B_Ω 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 [26]. The only exception is the case when B_Ω 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_Ω 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 (Ω, B_Ω) 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 Σ_1-formulas. Outside the Σ_1-formula case, it applies various heuristics based on clause simplification to try to answer inductive validity questions about QF Σ-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 Ω-abstracted as $\hat{\mathcal{R}}$. Therefore, we have automated the Ω-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 [55] 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 B.

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 $\mathcal{C} = \{A_i \rightarrow^{\circledast} B_i\}_{i \in I}$ 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.

6.1 Counter Proof Example

Though mathematically and syntactically simple, the counter example shown in subsection 5.2 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.

The reader may recall that the Maude syntax is quite similar to the formal notation used in the preliminaries in Section 2; this is no accident. Nevertheless, before continuing, we gloss over a few keywords in the Maude syntax and describe their associated concepts. 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 an equational theory whose equations are assumed (ground) convergent modulo the specified axioms B (specification of axioms is explained below), while the latter specifies a *system* module, which is a rewrite theory and therefore may contain both equations of the above form as well as potentially non-deterministic, non-terminating rewrite rules. Generally, functional modules specify data structures which are *imported* by potentially several different system modules, each specifying a different system of interest.

In this case, the system of interest is a counter (`COUNTER`) that may nondeterministically increase or decrease by one unit. 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` imports the functional module `PRES-NAT` in a semantics-preserving way that is precisely described in Definition 6.

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` (or `ceq` and `crl` in the conditional case), respectively.[17] The keyword `var` is used

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

to declare variables that will be used in later rules and equations. Constructor operators, the particular symbols that form the constructor subsignature Ω of the module's signature Σ, are specified using the `ctor` attribute, which appears within optional square brackets (`[]`) following an operator declaration.

```
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
```

Fig. 2. Theory specification for Counter.

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* [26,56].

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 counter 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 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 and/or commutative and/or having a unit element using the `assoc`, `comm`, and `id:` attributes, respectively. These declarations are not just descriptive: any equation or rule in which, for example, an associative-commutative operator appears, is actually applied *modulo* associativity and commutativity. This combination of features allows Maude to very naturally specify recursive functions and transition rewrite rules over lists, sets, multisets, etc., in a highly expressive and remarkably succinct way.

Now that we have described the rewrite theory to be analyzed as a Maude specification, we can 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 3 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.
```

Fig. 3. 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 8. 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 4. 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 5.2. 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

```
(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.
```

Fig. 4. Eventual Decrease of Counter Proof Script.

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 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.

6.2 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 Figure 5 and then show the proof script (without inline output) in Figure 6. 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 Σ and rewrite rules R to ensure that Maude's associative unification algorithm will work.[18] Thus, the signature Σ' of the Maude module `QLOCK` is identical to the signature Σ *except* for the list constructor which has an extra sort *NeList* with subsorts $Nat < NeList < List$, two constructors $nil : \rightarrow List$, $\_;\_ : NeList\ NeList \rightarrow NeList$, and where: (a) the original list concatenation operator $\_;\_ : List\ List \rightarrow List$ 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 6. 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.

[18] 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 [57]).

```
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

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
```

Fig. 5. 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 .)
```

Fig. 6. QLOCK Mutual Exclusion Proof Script

6.3 Other Examples

To validate the feasibility of our approach we also verified properties for a collection of examples. Table 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 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.*[19] 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.

[19] 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*.

We defined two order-preserving communication protocols assuming a sigle 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 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 4.1 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 5.2. 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.

7 Related Work and Conclusions

7.1 Related Work

Reachability logic [48,58,2,3] 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 [48,58]. This work, based on our earlier work in [29], 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 [2,3] 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 3, 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 6, 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 [7]. 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 [7] 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 [7] apply seem more restricted than the ones presented here. Roughly, (see their **Assumption 3**) [7] assumes restrictions akin to those imposed in [41] to allow "rewriting modulo SMT," which limits the equational theories (Σ, E) that can be handled. (3) Matching is used throughout in [7] 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 [7]; 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 [59]; (2) the coinductive approach by Lucanu et al. in [60]; and (3) the coinductive approach to reachability logic by Ciobâcă and Lucanu in [61]. The closest to our work are the approaches in [60] and, even more so, [61]. The approach in [60] adopts a semantic framework for models similar to the already-discussed work in [2,3], 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 [2,3], as well as the associated advantages and disadvantages, seem to be essentially the same as for [60]. However, an important contribution of the work in [60] is its coinductive semantics and justification for circular co-inductive reasoning. The relationship to our work is that the circular coinduction inference rule in [60] 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 [61]. 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 [61] 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, 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 [61] 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 $u \mid \varphi \rightarrow^{\circledast} v \mid \psi$, 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 combined effect of our STEP$^{\forall}$ and SUBSUMPTION rules (which in [29] 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 φ' 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 [62] and Maude [34]. 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, [63]) as in, e.g., [64], 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., [65,66]. 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 [67,68,69]. 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 [64], 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 [14]; (ii) decidable satisfiability (and validity) of quantifier-free formulas in initial algebras [16,17,18,19,20,21,22,23,24,25,26]; (iii) narrowing-based reachability analysis [15,46]; (iv) narrowing-based proof of safety properties [67,69]; (v) patterns and constrained patterns [21,46]; and (vi) rewriting modulo SMT [41]. Exploiting such connections, particularly with [14,26,56,46], has been essential to achieve the goals of this work.

7.2 Conclusions

In conclusion, this work 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 [64]; and (iii) how to exploit modularity and parameterization (in the sense of parameterized rewrite theories) aspects of reachability logic.

All this, together with reaching a mature tool implementation, are among our current goals for the near future.

Acknowledgments. We thank Grigore Roşu, Dorel Lucanu and Vlad Rusu for their very helpful comments on an earlier draft of this work, and the anonymous referees for numerous judicious comments that have also helped to substantially improve the paper. This research has been partially supported by NSF Grants CNS 13-19109 and CNS 14-09416, AFOSR Contract FA8750-11-2-0084, and NRL under contract N00173-17-1-G002.

References

1. Rosu G, Serbanuta T. An overview of the K semantic framework. *J. Log. Algebr. Program.*, 2010. **79**(6):397–434.
2. Stefanescu A, Ştefan Ciobâcă, Mereuta R, Moore BM, Serbanuta T, Rosu G. All-Path Reachability Logic. In: Proc. RTA-TLCA 2014, volume 8560. Springer LNCS, 2014 pp. 425–440.
3. Stefanescu A, Park D, Yuwen S, Li Y, Rosu G. Semantics-based program verifiers for all languages. In: Proc. OOPSLA 2016. ACM, 2016 pp. 74–91.
4. Meseguer J. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 1992. **96**(1):73–155.
5. Meseguer J. Twenty years of rewriting logic. *J. Algebraic and Logic Programming*, 2012. **81**:721–781.
6. Meseguer J, Rosu G. The rewriting logic semantics project: A progress report. *Inf. Comput.*, 2013. **231**:38–69.
7. Lucanu D, Rusu V, Arusoaie A, Nowak D. 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, volume 9200. Springer LNCS, 2015 pp. 451–474.
8. Siekmann JH. Unification Theory. *J. Symb. Comput.*, 1989. **7**(3/4):207–274.
9. Jouannaud JP, Kirchner C. 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.
10. Baader F, Siekmann JH. Unification theory. In: Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2, pp. 41–126. Oxford University Press, 1994.

11. Baader F, Snyder W. Unification Theory. In: Handbook of Automated Reasoning (in 2 volumes), pp. 445–532. Elsevier and MIT Press, 2001.
12. Hullot JM. Canonical Forms and Unification. In: Proc. Fifth Conference on Automated Deduction, volume 87 of *LNCS*, pp. 318–334. Springer, 1980.
13. Jouannaud JP, Kirchner C, Kirchner H. Incremental construction of unification algorithms in equational theories. In: Proc. ICALP'83. Springer LNCS 154, 1983 pp. 361–373.
14. Escobar S, Sasse R, Meseguer J. Folding variant narrowing and optimal variant termination. *J. Algebraic and Logic Programming*, 2012. **81**:898–928.
15. Meseguer J, Thati P. Symbolic reachability analysis using narrowing and its application to the verification of cryptographic protocols. *J. Higher-Order and Symbolic Computation*, 2007. **20**(1–2):123–160.
16. Maher MJ. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In: Proc. LICS '88. IEEE Computer Society, 1988 pp. 348–357.
17. Comon H, Lescanne P. Equational Problems and Disunification. *Journal of Symbolic Computation*, 1989. **7**:371–425.
18. Comon H. Complete Axiomatizations of Some Quotient Term Algebras. *Theor. Comput. Sci.*, 1993. **118**(2):167–191.
19. Baader F, Schulz KU. Combination Techniques and Decision Problems for Disunification. *Theor. Comput. Sci.*, 1995. **142**(2):229–255.
20. Comon H, Delor C. Equational Formulae with Membership Constraints. *Inf. Comput.*, 1994. **112**(2):167–216.
21. Meseguer J, Skeirik S. Equational Formulas and Pattern Operations in Initial Order-Sorted Algebras. In: Falaschi M (ed.), Proc. LOPSTR 2015, volume 9527. Springer LNCS, 2015 pp. 36—53.
22. Giesl J, Kapur D. Decidable Classes of Inductive Theorems. In: Proc. IJCAR 2001, volume 2083. Springer LNCS, 2001 pp. 469–484.
23. Giesl J, Kapur D. Deciding Inductive Validity of Equations. In: Proc. CADE 2003, volume 2741. Springer LNCS, 2003 pp. 17–31.
24. Falke S, Kapur D. Rewriting Induction + Linear Arithmetic = Decision Procedure. In: Proc. IJCAR 2012, volume 7364. Springer LNCS, 2012 pp. 241–255.
25. Aoto T, Stratulat S. Decision Procedures for Proving Inductive Theorems without Induction. In: Proc. PPDP2014. ACM, 2014 pp. 237–248.
26. Meseguer J. Variant-Based Satisfiability in Initial Algebras. In: Artho C, Ölveczky P (eds.), Proc. FTSCS 2015. Springer CCIS 596, 2016 pp. 1—32.
27. Bae K, Meseguer J. Model checking linear temporal logic of rewriting formulas under localized fairness. *Sci. Comput. Program.*, 2015. **99**:193–234.
28. Futatsugi K. Fostering Proof Scores in CafeOBJ. In: Proc. ICFEM 2010, volume 6447. Springer LNCS, 2010 pp. 1–20.
29. Skeirik S, Stefanescu A, Meseguer J. A Constructor-Based Reachability Logic for Rewrite Theories. In: Proc. Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017, volume 10855 of *Lecture Notes in Computer Science*. Springer, 2017 pp. 201–217.
30. Meseguer J. Variant-based satisfiability in initial algebras. *Sci. Comput. Program.*, 2018. **154**:3–41.
31. Skeirik S, Meseguer J. Metalevel algorithms for variant satisfiability. *J. Log. Algebr. Meth. Program.*, 2018. **96**:81–110.
32. Meseguer J. Membership algebra as a logical framework for equational specification. In: Proc. WADT'97. Springer LNCS 1376, 1998 pp. 18–61.

33. Goguen J, Meseguer J. Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theoretical Computer Science*, 1992. **105**:217–273.
34. Clavel M, Durán F, Eker S, Meseguer J, Lincoln P, Martí-Oliet N, Talcott C. All About Maude – A High-Performance Logical Framework. Springer LNCS Vol. 4350, 2007.
35. Dershowitz N, Jouannaud JP. Rewrite Systems. In: van Leeuwen J (ed.), Handbook of Theoretical Computer Science, Vol. B, pp. 243–320. North-Holland, 1990.
36. Gutiérrez R, Meseguer J, Rocha C. Order-sorted equality enrichments modulo axioms. *Sci. Comput. Program.*, 2015. **99**:235–261.
37. Meseguer J. Generalized Rewrite Theories and Coherence Completion. In: Rusu V (ed.), Proc. Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018, volume 11152 of *Lecture Notes in Computer Science*. Springer, 2018 pp. 164–183.
38. Lucas S, Meseguer J. Normal forms and normal theories in conditional rewriting. *J. Log. Algebr. Meth. Program.*, 2016. **85**(1):67–97.
39. Durán F, Meseguer J. On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories. *J. Log. Algebr. Program.*, 2012. **81**(7-8):816–850.
40. Meseguer J. Symbolic Reasoning Methods in Rewriting Logic and Maude. In: Moss LS, de Queiroz RJGB, Martínez M (eds.), Logic, Language, Information, and Computation - 25th International Workshop, WoLLIC 2018, Bogotá, Colombia, July 24–27, 2018, Proceedings, volume 10944 of *Lecture Notes in Computer Science*. Springer, 2018 pp. 25–60.
41. Rocha C, Meseguer J, Muñoz CA. Rewriting Modulo SMT and Open System Analysis. *Journal of Logic and Algebraic Methods in Programming*, 2017. **86**:269–297.
42. Bruni R, Meseguer J. Semantic foundations for generalized rewrite theories. *Theor. Comput. Sci.*, 2006. **360**(1-3):386–414.
43. Comon-Lundth H, Delaune S. The finite variant property: how to get rid of some algebraic properties. In Proc *RTA'05*, Springer LNCS 3467, 294–307, 2005.
44. Cholewa A, Meseguer J, Escobar S. Variants of Variants and the Finite Variant Property. Technical report, CS Dept. University of Illinois at Urbana-Champaign, 2014. Available at `http://hdl.handle.net/2142/47117`.
45. Rocha C, Meseguer J. Constructors, Sufficient Completeness, and Deadlock Freedom of Rewrite Theories. In: Proc. LPAR 2010, volume 6397 of *Lecture Notes in Computer Science*. Springer, 2010 pp. 594–609.
46. Meseguer J. Generalized Rewrite Theories, Coherence Completion, and Symbolic Methods, 2019. To appear in *Journal of Logical and Algebraic Methods in Programming*.
47. Hoare CAR. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 1969. **12**(10):576–580.
48. Rosu G, Stefanescu A. From Hoare Logic to Matching Logic Reachability. In: Giannakopoulou D, Méry D (eds.), FM, volume 7436 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-32758-2, 2012 pp. 387–402.
49. Reynolds JC. Separation Logic: A Logic for Shared Mutable Data Structures. In: LICS 2002. IEEE, 2002 pp. 55–74.
50. Meseguer J, Roşu G. The Rewriting Logic Semantics Project. *Theoretical Computer Science*, 2007. **373**:213–237.
51. Ellison C, Rosu G. An executable formal semantics of C with applications. In: Field J, Hicks M (eds.), POPL. ACM. ISBN 978-1-4503-1083-3, 2012 pp. 533–544.

52. Dowek G, Hardin T, Kirchner C. Theorem Proving Modulo. *J. Autom. Reasoning*, 2003. **31**(1):33–72.
53. Viry P. Adventures in sequent calculus modulo equations. *Electr. Notes Theor. Comput. Sci.*, 1998. **15**:21–32. doi:10.1016/S1571-0661(05)82550-2. URL `http://dx.doi.org/10.1016/S1571-0661(05)82550-2`.
54. Rocha C, Meseguer J. Theorem Proving Modulo Based on Boolean Equational Procedures. In: Proc. RelMiCS 2008, volume 4988. Springer LNCS, 2008 pp. 337–351.
55. Durán F, Ölveczky PC. A Guide to Extending Full Maude Illustrated with the Implementation of Real-Time Maude. *Electronic Notes in Theoretical Computer Science*, 2009. **238**(3):83 – 102.
56. Skeirik S, Meseguer J. Metalevel Algorithms for Variant-Based Satisfiability. In: Lucanu D (ed.), Proc. WRLA 2016, volume 9942. Springer LNCS, 2016 pp. 167–184.
57. Durán F, Eker S, Escobar S, Martí-Oliet N, Meseguer J, Talcott CL. Associative Unification and Symbolic Reasoning Modulo Associativity in Maude. In: Rusu V (ed.), Proc. Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018, volume 11152 of *Lecture Notes in Computer Science*. Springer, 2018 pp. 98–114.
58. Rosu G, Stefanescu A. Checking reachability using matching logic. In: Proc. OOPSLA 2012. ACM, 2012 pp. 555–574.
59. Moore B. Coinductive Program Verification. Ph.D. thesis, University of Illinois at Urbana-Champaign, 2016. `http://hdl.handle.net/2142/95372`.
60. Lucanu D, Rusu V, Arusoaie A. A generic framework for symbolic execution: A coinductive approach. *J. Symb. Comput.*, 2017. **80**:125–163.
61. Ştefan Ciobâcă, Lucanu D. A Coinductive Approach to Proving Reachability Properties in Logically Constrained Term Rewriting Systems. In: Proc. IJCAR 2018, volume 10900 of *Lecture Notes in Computer Science*. Springer, 2018 pp. 295–311.
62. Futatsugi K, Diaconescu R. CafeOBJ Report. World Scientific, 1998.
63. Goguen J. OBJ as a Theorem Prover with Application to Hardware Verification. In: Subramanyam P, Birtwistle G (eds.), Current Trends in Hardware Verification and Automated Theorem Proving, pp. 218–267. Springer-Verlag, 1989.
64. Futatsugi K. Generate & Check Method for Verifying Transition Systems in CafeOBJ. In: De Nicola R, Hennicker R (eds.), Software, Services, and Systems - Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering, volume 8950 of *Lecture Notes in Computer Science*. Springer, 2015 pp. 171–192.
65. Gâinâ D, Lucanu D, Ogata K, Futatsugi K. On Automation of OTS/CafeOBJ Method. In: Iida S, Meseguer J, Ogata K (eds.), Specification, Algebra, and Software - Essays Dedicated to Kokichi Futatsugi, volume 8373 of *Lecture Notes in Computer Science*. Springer, 2014 pp. 578–602.
66. Riesco A, Ogata K. Prove it! Inferring Formal Proof Scripts from CafeOBJ Proof Scores. *ACM Trans. Softw. Eng. Methodol.*, 2018. **27**(2):6:1–6:32.
67. Rocha C, Meseguer J. Proving Safety Properties of Rewrite Theories, 2011. In Proc. CALCO 2011, Springer LNCS 6859, 314-328.
68. Rocha C. Symbolic Reachability Analysis for Rewrite Theories. Ph.D. thesis, University of Illinois at Urbana-Champaign, 2012.
69. Rocha C, Meseguer J. 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, volume 8373 of *Lecture Notes in Computer Science*. Springer, 2014 pp. 603–629.

A Proofs of Lemmas and Theorems

Proof of Lemma 1

Proof. 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 $[u\alpha\tau!] \in [\![(u \mid \varphi \wedge \phi)\alpha]\!]$. Then, for $\rho = (\alpha\tau)|_Y$ we have $[u\alpha\tau!] \in [\![(u \mid \varphi)\rho]\!] \cap [\![(v \mid \phi)\rho]\!]$, as desired.

To show the $\subseteq$ part, let $[w] \in [\![(u \mid \varphi)\rho]\!] \cap [\![(v \mid \phi)\rho]\!]$ 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 $[w] = [u\alpha\delta!] \in [\![(u \mid \varphi \wedge \phi)\alpha]\!]$, as desired. □

Proof of Lemma 2

Proof. 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 3

Proof. Since $[\![u \mid \varphi]\!] \subseteq \mathit{Term}_{\mathcal{R}}$ and $[\![T]\!] \subseteq \mathit{Term}_{\mathcal{R}}$, $\mathcal{R} \models^{\forall}_{T} u \mid \varphi \to^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ iff for each $\rho \in [Y \to T_\Omega]$ and each $[w] \in [\![(u \mid \varphi)\rho]\!]$, if $[w] \in [\![T]\!]$ then $[w] \in [\![(\bigvee_{j \in J} v_j \mid \phi_j)\rho]\!]$. 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 4.

Proof. 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 σ 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 3.

Proof. A state $[\langle u_1, \ldots, u_n \rangle]_{B_\Omega} \in \mathcal{C}_{\mathcal{R},State}$ is reachable from $[\![S_0]\!]$ iff the state $[[u_1, \ldots, u_n]]_{B_\Omega}$ 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 Corollary 1.

Proof. Suppose (i) and (ii) hold. Then, since $S_0 \subseteq_Y P$, for each $\rho \in [Y \rightarrow T_\Omega]$ we have $[\![S_0\rho]\!] \subseteq [\![P\rho]\!]$, and, of course, since $P\rho$ is a variable renaming of $P\sigma\rho$, we also have $[\![P\rho]\!] = [\![P\sigma\rho]\!]$. But by Theorem 3 this shows that $[\![P\sigma\rho]\!] = [\![P\rho]\!]$ is an invariant of $(\mathcal{C}_{\mathcal{R},State}, \rightarrow_{\mathcal{R}})$ from $[\![P\rho]\!]$ and, *a fortiori*, from $[\![S_0\rho]\!]$. □

Proof of Theorem 4.

Proof. Suppose $[\![Q]\!]$ is not a co-invariant from $[\![S_0]\!]$. This exactly means that there are $[u] \in [\![S_0]\!]$ and $[v] \in [\![Q]\!]$ such that $[u] \rightarrow^*_{\mathcal{R}} [v]$, or, equivalently, $[v] \rightarrow^*_{\mathcal{R}^{-1}} [u]$. But since, by (ii) and Theorem 3, $[\![Q\sigma]\!]$ is an invariant of transition system $(\mathcal{C}_{\mathcal{R}^{-1},State}, \rightarrow_{\mathcal{R}^{-1}})$ from $[\![Q]\!] = [\![Q\sigma]\!]$, we must have $[u] \in [\![Q\sigma]\!]$, which is impossible by (i). □

Proof of Theorem 5.

Proof. We begin by introducing the following auxiliary notation

Definition 15. *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, $0 \leqslant j \leqslant m$, τ 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 9. *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. 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) $\mathcal{R} \not\models^{\forall,n_{min}} u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$, but (a) the Lemma's hypotheses hold for $n = n_{min}$; and (b) for any $n < n_{min}$ $\mathcal{R} \models^{\forall,n} u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$. We then show that, for any $[u_0] = [u\rho!] \in [\![u \mid \varphi]\!]$ and for any T-terminating path, $[u_0] \rightarrow_{\mathcal{R}} [u_1] \rightarrow_{\mathcal{R}} \ldots \rightarrow_{\mathcal{R}} [u_{n_{min}}]$ there exists a k, $0 \leqslant k \leqslant n_{min}$, τ and i with $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$, so

that, since $\mathcal{R} \models^{\forall,n}_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ for any $n < n_{min}$, we get $\mathcal{R} \models^{\forall,n_{min}}_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, 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, $\mathcal{R} \models^{\forall,n_{min}}_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$.

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

$$\varphi \Leftrightarrow (\varphi \wedge \bigvee_{(i,\beta)\in \text{MATCH}(u,\ \{v_i\},Y)} \psi_i\beta) \vee \varphi'.$$

Therefore, $[\![u \mid \varphi]\!] = [\![u \mid \varphi \wedge \bigvee_{(i,\beta)\in \text{MATCH}(u,\ \{v_i\},Y)} \psi_i\beta)]\!] \cup [\![u \mid \varphi']\!]$. But, since for 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 Ω-substitution $\gamma \uplus \tau$ with $dom(\gamma) = Y$ the goal's parameters, and for each $[w] = [(u(\gamma \uplus \tau))!] \in [\![u \mid \varphi \wedge \bigvee_{(i,\beta)\in \text{MATCH}(u,\ \{v_i\},Y)} \psi_i\beta)]\!]$ 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 $[w] \in [\![(v_i \mid \psi_i)\beta\gamma]\!]$, and, since γ and β have disjoint domains, so that $\gamma\beta = \beta\gamma$, a fortiori, $[w] \in [\![(\bigvee_i v_i \mid \psi_i)\gamma]\!]$. But this means that $[\![u \mid \varphi \wedge \bigvee_{(i,\beta)\in \text{MATCH}(u,\ \{v_i\},Y)} \psi_i\beta)]\!] \subseteq [\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!]$. Therefore, since we have: (i) $[\![u \mid \varphi]\!] = [\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!]) \uplus ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$, (ii) $[\![u \mid \varphi \wedge \bigvee_{(i,\beta)\in \text{MATCH}(u,\ \{v_i\},Y)} \psi_i\beta)]\!] \subseteq [\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!]$, and (iii) $[\![u \mid \varphi]\!] = [\![u \mid \varphi \wedge \bigvee_{(i,\beta)\in \text{MATCH}(u,\ \{v_i\},Y)} \psi_i\beta)]\!] \cup [\![u \mid \varphi']\!]$, the set-theoretic equalities (i)–(iii) force the containment $[\![u \mid \varphi']\!] \supseteq [\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$, 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 $[\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!]$ automatically satisfies the formula $u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, and $\mathcal{R} \models^{\forall,n_{min}-1}_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$, to reach the desired contradiction $\mathcal{R} \models^{\forall,n_{min}}_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ 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 $[u_0] = [u\rho!] \in [\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$. Note that if such a path is going to satisfy $\mathcal{R} \models^{\forall,n_{min}}_T u \mid \varphi \to^{\circledast} \bigvee_i v_i \mid \psi_i$ by means of some k, $0 \leqslant k \leqslant n_{min}$, with $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$, 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 $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$. But, since we have the over-approximation $[\![u \mid \varphi']\!] \supseteq [\![u \mid \varphi]\!] \backslash ([\![u \mid \varphi]\!] \cap_Y [\![\bigvee_i v_i \mid \psi_i]\!])$, and, by the assumptions on the minimal proof tree $\mathcal{P}$, we have $\mathcal{R} \models^{\forall,n_{min}}_T (r_j \mid \varphi' \wedge \phi_j)\alpha \to^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha$ for each $(j,\alpha) \in \text{UNIFY}(u \mid \varphi',\ R)$, it will be enough to show that: (a) $[u_1] = [(r_j\alpha\delta)!] \in [\![(r_j \mid \varphi' \wedge \phi_j)\alpha]\!]$ 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 δ, 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 $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$. First of all, since $[u_0] \rightarrow_{\mathcal{R}} [u_1]$, there is an unforgetful rule $l_j \rightarrow r_j$ *if* ϕ_j in R such that $[u_0] = [(u\rho)!] = [(l_j\gamma)!] \rightarrow_{\mathcal{R}} [(r_j\gamma)!] = [u_1]$, and $T_{\Sigma/E\cup B} \models \phi_j\gamma$. But since the variables of all sequents and those of $l_j \rightarrow r_j$ *if* ϕ_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 $(j, \alpha) \in \text{UNIFY}(u \mid \varphi',\ R)$ and a ground substitution δ 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 α are fresh, we have $vars((v_i \mid \psi_i)\alpha) = Z\backslash Y \uplus vars(\alpha(Y))$. Furthermore, since $u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ satisfies the invariant $vars(\psi_i) \subseteq vars(v_i) \cup vars(u \mid \varphi)$ for each i, and for each $(i, \beta) \in \text{MATCH}(u,\ \{v_i\}, Y)$ $u\beta = u =_{E_\Omega\cup B_\Omega} v_i\beta$, 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 $l_j \rightarrow r_j$ *if* ϕ_j is unforgetful, we have $vars(l_j) \subseteq vars(r_j) \cup vars(\phi_j) = W$, and therefore, by the freshness assumption on α 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 α, 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 $[u_1] \rightarrow_{\mathcal{R}} \ldots \rightarrow_{\mathcal{R}} [u_{n_{min}}]$ there is a k, $1 \leqslant k \leqslant n_{min}$, such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$, using the fact that $\mathcal{R} \models_T^{\forall, n_{min}} (r_j \mid \varphi' \wedge \phi_j)\alpha \rightarrow^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha$ for each $(j, \alpha) \in \text{UNIFY}(u \mid \varphi',\ R)$. 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 $\mathcal{R} \models_T^{\forall, n_{min}} (r_j \mid \varphi' \wedge \phi_j)\alpha \rightarrow^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha$ and (c), this ensures that there is a k, $1 \leqslant k \leqslant n_{min}$ such that $[u_k] \in [\![((v_i \mid \psi_i)\alpha)\delta|_{vars(\alpha(Y))}]\!] = [\![((v_i \mid \psi_i)\alpha|_Y)\delta|_{vars(\alpha(Y))}]\!] = [\![(v_i \mid \psi_i)(\alpha\delta)|_Y]\!] = [\![(v_i \mid \psi_i)\rho|_Y]\!]$. Therefore, there is a ground substitution τ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$, as desired.

AXIOM. 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 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 $\rho \in [U \rightarrow T_\Omega]$ such that $[u_0] = [(u\rho)!] \in [\![u \mid \varphi]\!]$ and any T-terminating path, $[u_0] \rightarrow_{\mathcal{R}} [u_1] \rightarrow_{\mathcal{R}} \ldots \rightarrow_{\mathcal{R}} [u_{n_{min}}]$ there exists a k, $0 \leqslant k \leqslant n_{min}$, an i, and a ground substitution τ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$.

Let $u' \mid \varphi' \rightarrow^{\circledast} \bigvee_j v'_j \mid \psi'_j$ 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, $[u_0] = [(u'\alpha\rho)!] \in$

$[\![u' \mid \varphi']\!]$. Since $\mathcal{R} \models^{\forall, n_{min}}_{T} u' \mid \varphi' \rightarrow^{\circledast} \bigvee_j v'_j \mid \psi'_j$, there exists j and $0 \leqslant k' \leqslant n_{min}$ such that $[u_{k'}] \in [\![(v'_j \mid \psi'_j)(\alpha\rho)|_{Y'}]\!]$. But by $(v'_j \mid \psi'_j)\alpha = (v'_j \mid \psi'_j)\alpha|_{Y'}$ and $Y = vars(\alpha(Y')$, we have $[\![(v'_j \mid \psi'_j)(\alpha\rho)|_{Y'}]\!] = [\![(v'_j\alpha \mid \psi'_j\alpha)\rho|_Y]\!]$. 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 $\rho \in [U \rightarrow T_\Omega]$ such that $T_{\Sigma/E \cup B} \models \varphi\rho$, $[\![(v'_j\alpha \mid \psi'_j\alpha)\rho|_Y]\!] = [\![(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi)\rho|_Y]\!]$.

Indeed, since $v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ is a closed subgoal in $\mathcal{P}$, we must have $\mathcal{R} \models^{\forall, n_{min}}_{T} v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$. But, by (1), $v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ has parameters Y and, by (2), $[u_{k'}] \in [\![(v'_j\alpha \mid \psi'_j\alpha \wedge \varphi)\rho|_Y]\!]$. But since the sequence $[u_{k'}] \rightarrow_{\mathcal{R}} \ldots \rightarrow_{\mathcal{R}} [u_{n_{min}}]$ has length $n \leqslant n_{min}$, there exist a k, $k' \leqslant k \leqslant n_{min}$, an i, and a ground substitution τ such that $[u_k] = [(v_i(\rho|_Y \uplus \tau))!] \in [\![(v_i \mid \psi_i)\rho|_Y]\!]$, 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 $U_0 \cap vars(\bigvee_i v_i \mid \psi_i) = \varnothing$. 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) using Lemma 9. Indeed, assume by contradiction that the theorem does not hold. Then, there must be a closed goal $(u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i) \in \mathcal{C}$ such that $[\mathcal{L},\ \mathcal{C}] \ \vdash_T \ u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ is a closed subgoal derived by our inference system for $\mathcal{R}$, but $\mathcal{R} \not\models u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i \in \mathcal{C}$. 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 9, we have $\mathcal{R} \models^{\forall, n_{min}}_{T} \varphi \rightarrow^{\circledast} \bigvee_i \psi_i$. This contradicts the assumption $\mathcal{R} \not\models^{\forall}_{n_{min}} u \mid \varphi \rightarrow^{\circledast} \bigvee_i v_i \mid \psi_i$ and completes the proof. □

Proof of Lemma 5

Proof. Since φ 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 12, using the parameter preservation condition. □

Proof of Lemma 6

Proof. Let Y be the parameters in $[\mathcal{A},\ \mathcal{C}] \ \vdash_T \ u \mid \varphi \rightarrow^{\circledast} A$. 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 $[\mathcal{A},\ \mathcal{C}] \ \vdash_T$

$u \mid \varphi \rightarrow^{\circledast} A$ 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 $[\mathcal{A},\ \mathcal{C}] \vdash_T (u \mid \varphi)\{x{:}s \mapsto u_i\} \rightarrow^{\circledast} A\{x{:}s \mapsto u_i\}$ are $(Y - \{x:s\}) \cup vars(u_i)$. Observe that, by the definition of pattern set for s, $[Y{\rightarrow}T_\Omega] = \bigcup_{1 \leqslant i \leqslant k} \{\{x:s \mapsto u_i\}\tau_i \mid \tau_i \in [(Y - \{x{:}s\}) \cup vars(u_i){\rightarrow}T_\Omega]\}$. Therefore, $\mathcal{R} \models^{\forall}_T [\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \rightarrow^{\circledast} A$ iff $\forall\ \rho \in [Y{\rightarrow}T_\Omega]\ \mathcal{R} \models^{\forall}_T ([\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \rightarrow^{\circledast} A)\rho$ iff $(\forall\ i,\ 1 \leqslant i \leqslant k)\ (\forall\ \tau_i \in [(Y - \{x{:}s\}) \cup vars(u_i){\rightarrow}T_\Omega])\ \mathcal{R} \models^{\forall}_T ([\mathcal{A},\ \mathcal{C}] \vdash_T u \mid \varphi \rightarrow^{\circledast} A)\{x{:}s \mapsto u_i\}\tau_i$ iff $\bigwedge_{1 \leqslant i \leqslant k} [\mathcal{A},\ \mathcal{C}] \vdash_T (u \mid \varphi)\{x{:}s \mapsto u_i\} \rightarrow^{\circledast} A\{x{:}s \mapsto u_i\}$, as desired. □

Proof of Lemma 7

Proof. Suppose the SUBSTITUTION rule is applied to $u \mid \bigwedge_i w_i = w'_i \wedge \varphi \rightarrow^{\circledast} \bigvee_{j \in J} v_j \mid \phi_j$ 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 $\rho \in [(U \backslash U_0 \uplus ran(\alpha)){\rightarrow}T_\Omega]$ 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:

$$\mathcal{R} \models^{\forall}_T (u \mid \bigwedge_i w_i = w'_i \wedge \varphi)\alpha \rightarrow^{\circledast} (\bigvee_{j \in J} v_j \mid \phi_j)\alpha$$
$$\Leftrightarrow$$
$$\mathcal{R} \models^{\forall}_T u\alpha \mid \varphi\alpha \wedge \widehat{\alpha} \rightarrow^{\circledast} (\bigvee_{j \in J} v_j \mid \phi_j)\alpha.$$

The ($\Leftarrow$) implication in the Lemma's proof now follows immediately from the above equivalence and the following *Instance Lemma*, where ψ is chosen to be the formula $\bigwedge_i w_i = w'_i \wedge \varphi$.

Lemma 8 (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* β *be a substitution whose domain* V *is contained in* $vars(u \mid \psi)$ *and where the variables in* $ran(\beta)$ *are all fresh. Then* $\mathcal{R} \models^{\forall}_T (u \mid \psi)\beta \rightarrow^{\circledast} (\bigvee_{j \in J} v_j \mid \phi_j)\beta$.

Proof. Let $U = vars(u \mid \psi)$. Note that, by the freshness assumption on β and $V \subseteq U$, the formula $(u \mid \psi)\beta \rightarrow^{\circledast} (\bigvee_{j \in J} v_j \mid \phi_j)\beta$ has parameters $vars(\beta(Y))$. We then need to show that for each $\delta \in [(U \backslash V) \uplus ran(\beta){\rightarrow}T_\Omega]$, $[u_0] = [(u\beta\delta)!] \in [\![(u \mid$

$\psi)(\beta\delta)|_{vars(\beta(Y))}]\!]$ and T-terminating sequence $[u_0] \to_{\mathcal{R}} [u_1] \dots [u_{n-1}] \to_{\mathcal{R}} [u_n]$ there is a $0 \leqslant k \leqslant n$ such that $[u_k] \in [\![(\bigvee_{j\in J} v_j \mid \phi_j)(\beta\delta)|_{vars(\beta(Y))}]\!]$. But $(\beta\delta)|_{vars(\beta(Y))} = \delta|_{Y\setminus V} \uplus (\beta\delta)|_{vars(\beta(Y\cap V))} = \delta|_{Y\setminus V} \uplus (\beta|_{Y\cap V})(\delta|_{vars(\beta(Y\cap V))}) = \delta|_{Y\setminus V} \uplus (\beta\delta)|_{Y\cap V} = (\beta\delta)|_Y$. Therefore, $[u_0] = [(u\beta\delta)!] \in [\![(u \mid \psi)(\beta\delta)|_Y]\!]$, 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 $[u_k] \in [\![(\bigvee_{j\in J} v_j \mid \phi_j)(\beta\delta)|_Y]\!] = [\![(\bigvee_{j\in J} v_j \mid \phi_j)(\beta\delta)|_{vars(\beta(Y))}]\!]$, as desired. □

We now resume the proof of the ($\Rightarrow$) implication for Lemma 7. 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 $[u_0] = [(u\gamma)!] \in [\![u \mid \bigwedge_i w_i = w'_i \wedge \varphi]\!]$ and each T-terminating sequence $[u_0] \to_{\mathcal{R}} [u_1] \dots [u_{n-1}] \to_{\mathcal{R}} [u_n]$ there is a $0 \leqslant k \leqslant n$, a $j \in J$, and a ground substitution $\tau \in [Z\setminus Y \to T_\Omega]$ such that $[u_k] = [(v_j(\rho|_Y \uplus \tau))!] \in [\![(\bigvee_{j\in J} v_j \mid \phi_j)\gamma|_Y]\!]$.

This can be shown as follows. Let $U_0 = vars(\bigwedge_i w_i = w'_i)$. Since γ 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 $\delta \in [(U\setminus U_0) \uplus ran(\alpha) \to T_\Omega]$ such that $\gamma =_{E_\Omega \cup B_\Omega} \alpha\delta$. Therefore, by our earlier Fact (1), $[u_0] = [(u\gamma)!] = [(u\alpha\delta)!] \in [\![u \mid \bigwedge_i w_i = w'_i \wedge \varphi \wedge \widehat{\alpha}]\!]$. 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 $\tau \in [Z\setminus Y \to T_\Omega] = [(vars(\alpha(Z))\setminus vars(\alpha(Y))) \to T_\Omega]$ such that $[u_k] = [(v_j\alpha(\delta|_{vars(\alpha(Y))} \uplus \tau))!] \in [\![(\bigvee_{j\in J} v_j \mid \phi_j)\alpha\delta|_{vars(\alpha(Y))}]\!]$. 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 $[u_k] = [(v_j(\gamma|_Y \uplus \tau))!] \in [\![(\bigvee_{j\in J} v_j \mid \phi_j)\gamma|_Y]\!]$, as desired. □

B Command Grammar

Here we provide a BNF grammar of the commands that can be given as inputs to our prototype Maude tool. In the grammar below, **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 cate-

gories 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⟩

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