Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 177 (2007) 5–33
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Narrowing and Rewriting Logic: from Foundations to Applications

Santiago Escobar[a,1] José Meseguer[b,2] Prasanna Thati[c,3]

[a] *Universidad Politécnica de Valencia, Spain.*
[b] *University of Illinois at Urbana-Champaign, USA.*
[c] *Carnegie-Mellon University, USA.*

Abstract

Narrowing was originally introduced to solve equational E-unification problems. It has also been recognized as a key mechanism to unify functional and logic programming. In both cases, narrowing supports equational reasoning and assumes confluent equations. The main goal of this work is to show that narrowing can be greatly generalized, so as to support a much wider range of applications, when it is performed with rewrite theories (Σ, E, R), where (Σ, E) is an equational theory, and R is a collection of rewrite rules with no restrictions. Such theories axiomatize concurrent systems, whose states are equivalence classes of terms modulo E, and whose transitions are specified by R. In this context, narrowing is generalized from an equational reasoning technique to a symbolic model checking technique for *reachability analysis* of a, typically infinite, concurrent system. We survey the foundations of this approach, suitable narrowing strategies, and various applications to security protocol verification, theorem proving, and programming languages.

Keywords: Narrowing, Rewriting Logic, Maude, Reachability, Equational Reasoning, Security protocols

1 Introduction

1.1 Why Rewriting Logic

Logic programming is a *parametric* idea: it is parameterized by the computational logic one chooses as the basis of one's programming language [42]. The more expressive the logic, the wider the range of applications one can naturally support without having to corrupt the language's declarative semantics. This poses the interesting challenge of finding more expressive computational logics without losing good efficiency; that is, without falling into the Turing tar pits of general theorem proving.

[1] Email:sescobar@dsic.upv.es
[2] Email:meseguer@cs.uiuc.edu
[3] Email:pthati@gmail.com

1571-0661
doi:10.1016/j.entcs.2007.01.004

Rewriting logic [43] is a computational logic that can be efficiently implemented and that widens quite substantially the range of applications naturally supported by declarative programming. It generalizes both equational logic and Horn logic, and furthermore supports a declarative programming style for object-oriented systems and for general distributed programming [44]. In fact, it is a very general *logical and semantic framework*, in which a wide range of logics and models of computation can be faithfully represented [40].

For the purposes of this paper it may be enough to sketch out two ideas: (i) how rewriting logic combines equational logic and traditional term rewriting; and (ii) what the intuitive meaning of a rewrite theory is all about. A rewrite theory is a triple $\mathcal{R} = (\Sigma, E, R)$ with Σ a signature of function symbols, E a set of Σ-equations of the form $t = t'$, and R a set of Σ-rewrite rules [4] of the form $l \to r$. Therefore, the logic's atomic sentences are of two kinds: equations, and rewrite rules. Equational theories and traditional term rewriting systems then appear as special cases. An equational theory (Σ, E) can be faithfully represented as the rewrite theory $(\Sigma, E, \emptyset)$; and a term rewriting system (Σ, R) can likewise be faithfully represented as the rewrite theory $(\Sigma, \emptyset, R)$.

Of course, if the equations of an equational theory (Σ, E) are *confluent*, there is another useful representation, namely, as the rewrite theory $(\tilde{\Sigma}, \emptyset, R_E)$, where $\tilde{\Sigma} = \Sigma \cup \{\approx, \mathtt{true}\}$, and $R_E = \overrightarrow{E} \cup \{\, x \approx x \to \mathtt{true}\}$, where $\overrightarrow{E}$ are the rewrite rules obtained by orienting the equations E. By confluence we then have the equivalence:

$$(\Sigma, E) \vdash t = t' \quad \Leftrightarrow \quad (\tilde{\Sigma}, \emptyset, R_E) \vdash t \approx t' \to^* \mathtt{true}$$

Much work in rewriting techniques and in functional logic programming has traditionally centered around this equivalence. But by implicitly suggesting that rewrite rules are just an efficient technique for equational reasoning, this equivalence can easily prevent us from seeing that rewrite rules can have a much more general *nonequational* semantics. This is the whole *raison d'etre* of rewriting logic. In rewriting logic a rewrite theory has two complementary readings: one computational, and the other logical. Computationally, a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ axiomatizes a *concurrent system*, whose states are E-equivalence classes, and whose *atomic transitions* are specified by the rules R. Logically, $\mathcal{R}$ axiomatizes a logical *inference system*, whose formulas are Σ-expressions satisfying structural axioms E, and whose inference rules are precisely the rules in R. The inference system of rewriting logic [43] then allows us to answer the same question in two complementary readings: (i) can we *reach* state $[t']_E$ from state $[t]_E$? and (ii) can we *derive* formula $[t']_E$ from formula $[t]_E$?

1.2 Narrowing as Symbolic Reachability Analysis

Of course, questions (i) and (ii) above are the same question, namely, the *reachability* question. Rewriting logic gives us a complete inference system [43] to derive

[4] In general, rewrite rules can be conditional [43], but we treat here the simpler, unconditional case.

for a given rewrite theory $\mathcal{R}$ all valid *universally quantified* reachability formulas $(\forall \overrightarrow{x})\ t \rightarrow^* t'$. But an equally important problem is being able to derive all valid *existentially quantified* reachability formulas $(\exists \overrightarrow{x})\ t \rightarrow^* t'$. Why is answering such existential reachability questions important? Because if we could, we would have a very powerful *symbolic model checking* technique, not in the limited BDD-based finite-state sense, but in the much more general sense of model checking infinite state systems. Indeed, in the formula $(\exists \overrightarrow{x})\ t \rightarrow^* t'$, t represents a typically infinite set of *initial states*, and t' represents a typically infinite set of *target states*. The model checking question is then whether from some initial state in t we can reach some state in t'. For example, t' may describe a set of attacks on a security protocol; then such attacks exist iff $(\exists \overrightarrow{x})\ t \rightarrow^* t'$ is valid.

Here is where narrowing comes in. Proposed originally as a method to solve equational goals $(\exists \overrightarrow{x})\ t = t'$, [23,33,35], it was soon recognized as a key mechanism to unify functional and logic programming [26,29]. But even in that original setting we can reinterpret narrowing as a technique to answer reachability questions. That is, narrowing allows us to convert the question $(\exists \overrightarrow{x})\ t = t'$ in (Σ, E) into the reachability question $(\exists \overrightarrow{x})\ t \approx t' \rightarrow^* \mathtt{true}$ in the rewrite theory $(\tilde{\Sigma}, \emptyset, R_E)$. But the converse is definitely not true: when we interpret rewrite rules as transitions in a system, reachability questions do *not* have an equational counterpart: we may be able to reach a state b from a state a, but it may be impossible to return to a from b. That is, reachability is definitely *not* symmetric. The whole point of a rewrite theory (Σ, E, R) is to distinguish equality between states by E, and reachability between states by R as totally different relations.

The goal, then, is to generalize narrowing from an equational solving technique for confluent equational theories (Σ, E) to a symbolic reachability analysis technique for arbitrary rewrite theories (Σ, E, R), whose rules R may typically fail to be confluent, and may often not terminate. In this way, we obtain a useful new technique, first suggested in [44,14], to model check infinite state systems. In this sense, narrowing complements other existing techniques for analyzing such systems, including model checking for suitable subclasses, e.g., [7,9,17,24], abstraction techniques, e.g., [10,38,28,36,54], tree-automata based reachability analyses, e.g., [25,50], and theorem proving, e.g. [52,51].

Note that narrowing now has to happen *modulo* the equations E. A particularly nice situation, on which we focus, is when the equations E decompose as a disjoint union $E = \Delta \uplus B$, with B having a finitary unification algorithm, and with Δ confluent and terminating modulo B. Under appropriate coherence [60] conditions discussed in Section 3, the theory (Σ, E, R) becomes semantically equivalent to the much more tractable theory $(\Sigma, B, \overrightarrow{\Delta} \cup R)$.

In the fullest possible generality, narrowing is a sound but not necessarily complete method to solve reachability goals $(\forall \overrightarrow{x})\ t \rightarrow^* t'$. However, in Sections 5 and 6 we show that: (i) it is complete in the weaker sense of finding all normalized solutions; (ii) it is complete in the strong sense for wide classes of theories of practical interest; and (iii) completeness in the solvability sense can be recovered for arbitrary rewrite systems using back-and-forth narrowing.

Efficiency of narrowing by means of clever *strategies* is another important concern. In Section 6.2 we report on two research directions we have been advancing in this area. On the one hand, our goal has been to generalize to arbitrary rewriting systems the extension from *lazy* rewriting strategies [55,4,5] to lazy narrowing strategies for functional logic programming provided by Antoy, Echahed, and Hanus with their *needed narrowing* [6,5]. This is an optimal demand-driven strategy that lazily narrows only those outermost positions that are strictly necessary while generating also optimal unifiers. Needed narrowing was improved by a more refined notion of demandedness by the *natural narrowing* strategy proposed by S. Escobar [18,19]. However, these lazy narrowing strategies are complete only under certain strong assumptions, such as that the rewrite rules are left-linear and constructor-based. These assumptions, while reasonable in a functional (logic) setting, are quite restrictive in our more general reachability setting that we are interested in. In recent work [21], we have proposed a generalization of natural narrowing to a reachability setting where the rewrite rules can be non-left-linear and non-constructor-based. This generalization is strictly more efficient than needed narrowing when specialized to the functional (logic) setting, and it is complete in the weak sense that it is guaranteed to find all R-normalized solutions. On the other hand, a second, quite different strategy idea, first suggested by C. Meadows [41], centers upon using term grammars to drastically cut down the narrowing search space. Although we illustrate this technique in the context of security protocol verification in which it first arose, and where we are further extending it in collaboration with Meadows [20], we believe that it will have a much wider applicability in practice to general narrowing-based symbolic model checking.

1.3 From Foundations to Applications

As already mentioned, the whole point of having a more general computational logic is to support a wider range of applications. In Section 7 we try to give a flavor for several new applications that our proposed generalization of narrowing to rewrite theories make possible, including: (i) new security protocol verification methods; (ii) more efficient theorem proving techniques; and (iii) more expressive and efficient programming language techniques.

2 Background

We assume some familiarity with term rewriting and narrowing, see [57,43] for missing definitions. Given a binary relation $\Rightarrow \subseteq T \times T$ on a set T of elements, we say that an element $a \in T$ is *$\Rightarrow$-irreducible* (or is a *normal form* w.r.t. $\Rightarrow$) if there is no element $b \in T$ such that $a \Rightarrow b$. We denote the transitive closure of $\Rightarrow$ by $\Rightarrow^+$, and the transitive and reflexive closure by $\Rightarrow^*$. We say that the relation $\Rightarrow$ is *terminating* if there is no infinite sequence $a_1 \Rightarrow a_2 \Rightarrow \cdots \Rightarrow \cdots$. We say that $\Rightarrow$ is *confluent* if whenever $a \Rightarrow^* b$ and $a \Rightarrow^* c$, there exists an element d such that $b \Rightarrow^* d$ and $c \Rightarrow^* d$. We say that $\Rightarrow$ is *convergent* if it is confluent and terminating.

An *order-sorted signature* Σ is defined by a set of sorts S, a partial order re-

lation of subsort inclusion $\leq$ on S, and an $(\mathsf{S}^* \times \mathsf{S})$-indexed family of operations $\{\Sigma_{\mathsf{w},\mathsf{s}}\}_{(\mathsf{w},\mathsf{s})\in \mathsf{S}^*\times\mathsf{S}}$. We denote $f \in \Sigma_{\mathsf{w},\mathsf{s}}$ by $f : \mathsf{w} \to \mathsf{s}$. We define a relation $\equiv$ on S as the smallest equivalence relation generated by the subsort inclusion relation $\leq$. We assume that each equivalence class of sorts contains a *top* sort that is a supersort of every other sort in the class. Formally, for each sort s we assume that there is a sort [5] $[\mathsf{s}]$ such that $\mathsf{s} \equiv \mathsf{s}'$ implies $\mathsf{s}' \leq [\mathsf{s}]$. Furthermore, for each $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ we assume that there is also an $f : [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n] \to [\mathsf{s}]$. We require the signature Σ to be *sensible*, i.e., whenever we have $f : \mathsf{w} \to \mathsf{s}$ and $f : \mathsf{w}' \to \mathsf{s}'$ with w, w' of equal length, then $\mathsf{w} \equiv \mathsf{w}'$ implies $\mathsf{s} \equiv \mathsf{s}'$.

A Σ-*algebra* is defined by an S-indexed family of sets $\mathsf{A} = \{\mathsf{A}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ such that $\mathsf{s} \leq \mathsf{s}'$ implies $\mathsf{A}_\mathsf{s} \subseteq \mathsf{A}_{\mathsf{s}'}$, and for each function $f : \mathsf{w} \to \mathsf{s}$ with $\mathsf{w} = \mathsf{s}_1 \times \ldots \times \mathsf{s}_n$ a function $f_{\mathsf{A}^{\mathsf{w},\mathsf{s}}} : \mathsf{A}_{\mathsf{s}_1} \times \ldots \times \mathsf{A}_{\mathsf{s}_n} \to \mathsf{A}_\mathsf{s}$. Further, we require that subsort overloaded operations agree, i.e., for each $f : \mathsf{w} \to \mathsf{s}$ and $(\mathsf{a}_1, \ldots, \mathsf{a}_n) \in \mathsf{A}^\mathsf{w}$ we require $f_{\mathsf{A}^{\mathsf{w},\mathsf{s}}}(\mathsf{a}_1, \ldots, \mathsf{a}_n) = f_{\mathsf{A}^{[\mathsf{w}],[\mathsf{s}]}}(\mathsf{a}_1, \ldots, \mathsf{a}_n)$, where if $\mathsf{w} = \mathsf{s}_1 \times \ldots \times \mathsf{s}_n$, then $[\mathsf{w}] = [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n]$. We assume a family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of infinite sets of variables such that $\mathsf{s} \not\equiv \mathsf{s}'$ implies $\mathcal{X}_\mathsf{s} \cap \mathcal{X}_{\mathsf{s}'} = \emptyset$, and all variables in $\mathcal{X}$ are different from any constant symbols in Σ. We use uppercase letters $X, Y, W, \ldots$ to denote variables in $\mathcal{X}$. We denote the set of ground Σ-terms and Σ-terms of sort s by $\mathcal{T}_{\Sigma\mathsf{s}}$ and $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$, respectively. More generally, we write $\mathcal{T}_\Sigma$ for the Σ-algebra of ground terms over Σ, and $\mathcal{T}_\Sigma(\mathcal{X})$ for the Σ-algebra of terms with variables from $\mathcal{X}$. We use lowercase letters $t, s, u, v, w, \ldots$ to denote terms in $\mathcal{T}_\Sigma(\mathcal{X})$. $Var(t)$ denotes the set of variables in $t \in \mathcal{T}_\Sigma(\mathcal{X})$. A term is *linear* if each variable in the term occurs at a single position. We denote the linearized version of a term t by $\overline{t}$.

We use a finite sequence of positive integers, called a *position*, to denote an access path in a term. We use lowercase letters p, q to denote positions in a term. For $t \in \mathcal{T}_\Sigma(\mathcal{X})$, $Pos(t)$ denotes the set of positions in t, and $Pos_\Sigma(t)$ denotes the set of non-variable positions in t. Given a position p and a set P of positions, we define $p.P = \{p.q \mid q \in P\}$ and just write $p.q$ for $p.\{q\}$. The root of a term is at the empty position ϵ. The subterm of t at position p is denoted by $t|_p$ and $t[s]_p$ is the term t with the subterm at position p replaced by s.

A *substitution* is an S-sorted mapping $\sigma : \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{X})$ which maps a variable of sort s to a term of sort s, and which is different from the identity only for a finite subset $Dom(\sigma)$ of $\mathcal{X}$. A substitution σ with $Dom(\sigma) = \{X_1, \ldots, X_n\}$ is usually denoted as $\sigma = [t_1/X_1, \ldots, t_n/X_n]$. The identity substitution is denoted by id, i.e., $Dom(id) = \emptyset$. We denote the homomorphic extension of σ to $\mathcal{T}_\Sigma(\mathcal{X})$ also by σ. The set of variables introduced by σ is $Ran(\sigma) = \cup_{X\in Dom(\sigma)} Var(\sigma(X))$. A substitution σ is called a *renaming* if it is a bijective mapping of variables to new variables that preserves the sorts strictly, i.e., for each $X \in \mathcal{X}_\mathsf{s}$, $\sigma(X) \in (\mathcal{X}_\mathsf{s} \setminus Dom(\sigma))$ and $\sigma(X) \neq \sigma(Y)$ for any two different variables $X, Y \in Dom(\sigma)$. A term t is called a *renamed version* of another term s if there is a renaming σ such that $t = \sigma(s)$. The restriction of a substitution σ to a set of variables V is defined as $\sigma|_V(X) = \sigma(X)$ if $X \in V$; and $\sigma|_V(X) = X$ otherwise. For substitutions σ, ρ such

[5] In the order-sorted specifications discussed in this paper we will sometimes leave this top sort and its associated operators implicit, in the sense that an order-sorted signature can always be conservatively completed to one satisfying our requirements.

that $Dom(\sigma) \cap Dom(\rho) = \emptyset$ we define their composition as $(\sigma \circ \rho)(X) = \rho(\sigma(X))$ for each variable X in $\mathcal{X}$. We say that a substitution σ is *away* from a set of variables V if $Ran(\sigma) \cap V = \emptyset$.

A Σ-*equation* is an expression of the form $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for an appropriate sort s. Order-sorted equational logic has a sound and complete inference system $E \vdash_\Sigma$ (see [45]) inducing a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$: $t =_E t'$ if and only if $E \vdash_\Sigma t = t'$; where under the assumption that all sorts S in Σ are non-empty, i.e., $\forall \mathsf{s} \in \mathsf{S} : \mathcal{T}_{\Sigma \mathsf{s}} \neq \emptyset$, the inference system $E \vdash_\Sigma$ can treat universal quantification in an implicit way.

The *E-subsumption* preorder $\ll_E$ holds between $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, denoted $t \ll_E t'$ (meaning that t' is more general than t), if there is a substitution σ such that $t =_E \sigma(t')$; such a substitution σ is said to be an *E-match* from t to t'. For substitutions σ, ρ and a set of variables V we define $\sigma|_V =_E \rho|_V$ if $\sigma(x) =_E \rho(x)$ for all $x \in V$, and $\sigma|_V \ll_E \rho|_V$ if there is a substitution η such that $\sigma|_V =_E (\rho \circ \eta)|_V$. We write $e \ll e'$ when E is empty, i.e., for $e \ll_\emptyset e'$.

An *E-unifier* for a Σ-equation $t = t'$ is a substitution σ such that $\sigma(t) =_E \sigma(t')$. For $Var(t) \cup Var(t') \subseteq W$, a set of substitutions $CSU_E(t = t', W)$ is said to be a *complete* set of unifiers of the equation $t =_E t'$ away from W if: (i) each $\sigma \in CSU_E(t = t', W)$ is an E-unifier of $t =_E t'$; (ii) for any E-unifier ρ of $t =_E t'$ there is a $\sigma \in CSU_E(t = t', W)$ such that $\rho|_V \ll_E \sigma|_V$ and $V = Var(t) \cup Var(t')$; (iii) for all $\sigma \in CSU_E(t = t', W)$, $Dom(\sigma) \subseteq (Var(t) \cup Var(t'))$ and $Ran(\sigma) \cap W = \emptyset$. An E-unification algorithm is *complete* if for any equation $t = t'$ it generates a complete set of E-unifiers. Note that this set needs not be finite. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of solutions.

A *rewrite rule* is an expression of the form $l \to r$, where $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for an appropriate sort s. The term l (resp. r) is called the *left-hand* side (resp. *right-hand* side) of the rule $l \to r$. In this paper we allow extra variables in right-hand sides, i.e., we do *not* impose the usual condition $Var(r) \subseteq Var(l)$. We will make explicit when extra variables are not allowed. An *(unconditional) order-sorted rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$ with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. We write R^{-1} for the reversed rules of R, i.e., $R^{-1} = \{r \to l \mid l \to r \in R\}$. We call $\mathcal{R}$ linear (resp. left-linear, right-linear) if for each rule $l \to r \in R$, l and r are linear (resp. l is linear, r is linear). Given $\mathcal{R} = (\Sigma, \emptyset, R)$, we might assume that Σ is defined as the disjoint union $\Sigma = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors*, and symbols $f \in \mathcal{D}$, called *defined symbols*, where $\mathcal{D} = \{root(l) \mid l \to r \in R\}$ and $\mathcal{C} = \Sigma - \mathcal{D}$. A pattern is a term $f(l_1, \ldots, l_k)$ where $f \in \mathcal{D}$ and $l_i \in T_\mathcal{C}(\mathcal{X})$, for $1 \leq i \leq k$. A rewrite system $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, \emptyset, R)$ is *constructor-based* if every left-hand side of a rule in R is a pattern.

We define the *one-step rewrite relation* $\to_R$ on $\mathcal{T}_\Sigma(\mathcal{X})$ as follows: $t \xrightarrow{p}_R t'$ (or $\to_R$ if p is not relevant) if there is a position $p \in Pos_\Sigma(t)$, a (possibly renamed) rule $l \to r$ in R such that $Var(t) \cap (Var(r) \cup Var(l)) = \emptyset$, and a substitution σ such that $t|_p = \sigma(l)$ and $t' = t[\sigma(r)]_p$. Note that during a rewrite step extra variables in the right-hand side of the rewrite rule being used may be automatically instantiated

with arbitrary substitutions. The relation $\rightarrow_{R/E}$ for rewriting modulo E is defined as $=_E \circ \rightarrow_R \circ =_E$, i.e., $t \rightarrow_{R/E} t'$ if there are $w, w' \in \mathcal{T}_\Sigma(\mathcal{X})$ such that $t =_E w$, $w \rightarrow_R w'$, and $w' =_E t'$. Note that $\rightarrow_{R/E}$ induces a relation on E-equivalence classes, namely, $[t]_E \rightarrow_{R/E} [t']_E$ iff $t \rightarrow_{R/E} t'$. We say $\mathcal{R} = (\Sigma, E, R)$ is terminating (resp. confluent, convergent) if the relation $\rightarrow_{R/E}$ is terminating (resp. confluent, convergent).

For substitutions σ, ρ and a set of variables V we define $\sigma|_V \rightarrow_R \rho|_V$ if there is $X \in V$ such that $\sigma(X) \rightarrow_R \rho(X)$ and for all other $Y \in V$ we have $\sigma(Y) = \rho(Y)$. The relation $\rightarrow_{R/E}$ on substitutions is defined as $=_E \circ \rightarrow_R \circ =_E$. A substitution σ is called *R/E-normalized* if $\sigma(X)$ is $\rightarrow_{R/E}$-irreducible for all X.

3 Narrowing

Since E-congruence classes can be infinite, $\rightarrow_{R/E}$-reducibility is undecidable in general. Therefore, we "implement" R/E-rewriting by a combination of rewriting using oriented equations and rules. We assume that E is split into a set of (oriented) equations Δ and a set of axioms B, i.e., $E = \Delta \uplus B$, and is such that:

(i) B is *regular*, i.e., for each $t = t'$ in B, we have $Var(t) = Var(t')$, and *sort-preserving*, i.e., for each substitution σ, we have $\sigma(t) \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ if and only if $\sigma(t') \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$.

(ii) B has a finite and complete unification algorithm and $\Delta \cup B$ has a complete (but not necessarily finite) unification algorithm.

(iii) For each $t = t'$ in Δ we have $Var(t') \subseteq Var(t)$.

(iv) Δ is *sort-decreasing*, i.e., for each $t = t'$ in Δ, each $\mathsf{s} \in \mathsf{S}$, and each substitution σ, $\sigma(t') \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ implies $\sigma(t) \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$.

(v) The rewrite rules $\overrightarrow{\Delta}$ obtained by orienting the equations in Δ are *confluent and terminating modulo B*, i.e., the relation $\rightarrow_{\overrightarrow{\Delta}/B}$ is convergent modulo B.

Definition 3.1 ($R \cup \Delta, B$-rewriting) *We define the relation $\rightarrow_{R,B}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ as $t \rightarrow_{R,B} t'$ if there is a $p \in Pos_\Sigma(t)$, $l \rightarrow r$ in R such that $Var(t) \cap (Var(r) \cup Var(l)) = \emptyset$, and a substitution σ such that $t|_p =_B \sigma(l)$ and $t' = t[\sigma(r)]_p$. The relation $\rightarrow_{\Delta,B}$ is similarly defined by considering the oriented rewrite rules $\overrightarrow{\Delta}$ obtained from the equations in Δ. We define $\rightarrow_{R \cup \Delta, B}$ as $\rightarrow_{R,B} \cup \rightarrow_{\Delta,B}$.*

Note that, since B-matching is decidable, $\rightarrow_{\Delta,B}$, $\rightarrow_{R,B}$, and $\rightarrow_{R\cup\Delta,B}$ are decidable. $R \cup \Delta, B$-normalized (and similarly R, B or Δ, B-normalized) substitutions are defined in a straightforward manner.

The idea is to implement $\rightarrow_{R/E}$ (on terms and goals) using $\rightarrow_{R\cup\Delta,B}$. For this to work, we need the following additional assumptions.

(vi) We assume that $\rightarrow_{\Delta,B}$ is *coherent with B* [35], i.e., $\forall t_1, t_2, t_3$ we have $t_1 \rightarrow^+_{\Delta,B} t_2$ and $t_1 =_B t_3$ implies $\exists t_4, t_5$ such that $t_2 \rightarrow^*_{\Delta,B} t_4$, $t_3 \rightarrow^+_{\Delta,B} t_5$ and $t_4 =_E t_5$.

(vii) We assume that

(a) $\rightarrow_{R,B}$ is *E-consistent with B*, i.e. $\forall t_1, t_2, t_3$ we have that $t_1 \rightarrow_{R,B} t_2$ and

$t_1 =_B t_3$ imply $\exists t_4$ such that $t_3 \to_{R,B} t_4$ and $t_2 =_E t_4$; and

(b) $\to_{R,B}$ is *E-consistent with* $\to_{\Delta,B}$, i.e. $\forall t_1, t_2, t_3$ we have that $t_1 \to_{R,B} t_2$ and $t_1 \to^*_{\Delta,B} t_3$ imply $\exists t_4, t_5$ such that $t_3 \to^*_{\Delta,B} t_4$ and $t_4 \to_{R,B} t_5$ and $t_5 =_E t_2$.

The following lemma links $\to_{R/E}$ with $\to_{\Delta,B}$ and $\to_{R,B}$.

Lemma 3.2 [47] *Let $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ be an order-sorted rewrite theory with properties (i)–(vii) assumed above. Then $t_1 \to_{R/E} t_2$ if and only if $t_1 \to^*_{\Delta,B} \to_{R,B} t_3$ for some $t_3 =_E t_2$.*

Thus $t_1 \to^*_{R/E} t_2$ if and only if $t_1 \to^*_{R\cup\Delta,B} t_3$ for some $t_3 =_E t_2$.

Narrowing generalizes rewriting by performing unification at non-variable positions instead of the usual matching. The essential idea behind narrowing is to *symbolically* represent the transition relation between terms as a narrowing relation between terms. Specifically, narrowing instantiates the variables in a term by a B-unifier that enables a rewrite modulo B with a given rule and a term position.

Definition 3.3 ($R \cup \Delta, B$-narrowing) *The $R\cup\Delta, B$-narrowing relation on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as $t \overset{\sigma}{\leadsto}_{R\cup\Delta,B} t'$ (or $\leadsto_\sigma$ if $R \cup \Delta, B$ is understood) if there is $p \in Pos_\Sigma(t)$, a rule $l \to r$ in $R \cup \Delta$ such that $Var(t) \cap (Var(l) \cup Var(r)) = \emptyset$, and $\sigma \in CSU_B(t|_p = l, V)$ for a set V of variables containing $Var(t)$, $Var(l)$, and $Var(r)$, such that $t' = \sigma(t[r]_p)$.*

Similarly, Δ, B-narrowing and R, B-narrowing relations are defined on terms as expected.

4 Narrowing Reachability Goals

First, we recall the definition of reachability goals provided in [47].

Definition 4.1 (Reachability goal) *Given an order-sorted rewrite theory $\mathcal{R} = (\Sigma, E, R)$, we define a reachability goal G as a conjunction of the form $t_1 \to^* t'_1 \wedge \ldots \wedge t_n \to^* t'_n$, where for $1 \le i \le n$, we have $t_i, t'_i \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s_i}}$ for appropriate sorts $\mathsf{s_i}$. The empty goal is denoted by Λ. We assume that conjunction $\wedge$ is* associative and commutative*, so that the order of conjuncts is irrelevant.*

We say that the t_i are the *sources* of the goal G, while the t'_i are the *targets*. We define $Var(G) = \bigcup_i (Var(t_i) \cup Var(t'_i))$. A substitution σ is an *$\mathcal{R}$-solution* of G (or just a solution for short, when $\mathcal{R}$ is clear from the context) if $\sigma(t_i) \to^*_{R/E} \sigma(t'_i)$ for $1 \le i \le n$.

Definition 4.2 (R/E-rewriting on goals) *We define the rewrite relation on goals as follows.*

(Reduce) $G \wedge t_1 \to^* t_2 \quad \to_{R/E} \quad G \wedge t'_1 \to^* t_2 \quad$ *if* $t_1 \to_{R/E} t'_1$

(Eliminate) $G \wedge t \to^* t \quad \to_{R/E} \quad G.$

The relations $\to_{R\cup\Delta,B}$, $\to_{R,B}$, and $\to_{\Delta,B}$ are lifted to goals and substitutions in a similar manner.

Lemma 4.3 [47] *σ is a solution of a reachability goal G if and only if $\sigma(G) \rightarrow^*_{R/E} \Lambda$.*

Given an (unconditional) order-sorted rewrite theory $\mathcal{R}$, we are interested in finding a complete set of $\mathcal{R}$-solutions for a given goal G.

Definition 4.4 (Complete set of solutions on goals) *A set Γ of substitutions is said to be a complete set of $\mathcal{R}$-solutions of G if*

- *Every substitution $\sigma \in \Gamma$ is an $\mathcal{R}$-solution of G, and*
- *For any $\mathcal{R}$-solution ρ of G there is a substitution $\sigma \in \Gamma$ such that $\sigma|_{Var(G)} \ll_E \rho|_{Var(G)}$.*

The narrowing relation on terms is extended to reachability goals by narrowing only the left-hand sides of the goals, while the right-hand sides only accumulate substitutions. The idea is to repeatedly narrow the left-hand sides until each left-hand side unifies with the corresponding right-hand side. The composition of the unifier with all the substitutions generated (in the reverse order) gives us a solution of the goal.

Definition 4.5 ($R \cup \Delta, B$-narrowing on goals) *We define the narrowing relation on goals as follows.*

(Narrow) $G \wedge t_1 \rightarrow^* t_2 \overset{\sigma}{\rightsquigarrow}_{R\cup\Delta,B} \sigma(G) \wedge t_1' \rightarrow^* \sigma(t_2)$ *if* $t_1 \overset{\sigma}{\rightsquigarrow}_{R\cup\Delta,B} t_1'$

(Unify) $G \wedge t_1 \rightarrow^* t_2 \overset{\sigma}{\rightsquigarrow}_{R\cup\Delta,B} \sigma(G)$ *if* $\sigma \in CSU_{\Delta\cup B}(t_1 = t_2, Var(G))$.

We write $G \overset{\sigma}{\rightsquigarrow}^*_R G'$ if there is a sequence of derivations $G \overset{\sigma_1}{\rightsquigarrow}_R \ldots \overset{\sigma_n}{\rightsquigarrow}_R G'$ such that $\sigma = \sigma_n \circ \sigma_{n-1} \circ \ldots \circ \sigma_1$. Similarly, Δ, B-narrowing and $R \cup \Delta, B$-narrowing relations are defined on goals as expected.

5 Soundness and Weak Completeness

Let us recall that Δ, B-narrowing is known to give a sound and complete procedure for $\Delta \cup B$-unification [35]. Soundness of narrowing as a solving reachability procedure is easy from [35].

Theorem 5.1 (Soundness) [47] *If $G \overset{\sigma}{\rightsquigarrow}^*_{R\cup\Delta,B} \Lambda$, then σ is a solution of G.*

The completeness of narrowing as a procedure for solving equational goals critically depends on the assumption that the equations are confluent, an assumption that is no longer reasonable in our more general reachability setting, where the meaning of a rewrite is changed from an oriented equality to a *transition* or an *inference*, so that we can specify and program with rewrite theories concurrent systems and logical inference systems. In this general setting, confluence and termination are not reasonable assumptions, and are therefore dropped. As a result of this, narrowing is no longer a complete procedure for solving reachability goals, in that it may fail to find certain solutions.

The idea behind proving weak completeness is to associate with each $R \cup \Delta, B$-rewriting derivation an $R \cup \Delta, B$-narrowing derivation. It is possible to establish such a correspondence only for R/E-normalized substitutions, and hence the weakness

in completeness.

Theorem 5.2 (Weak completeness) [47] *Let R be a set of rewrite rules with no extra-variables in the right-hand side, ρ be an R/E-normalized solution of a reachability goal G, and let V be a finite set of variables containing $Var(G)$. Then there is σ such that $G \overset{\sigma}{\rightsquigarrow}^*_{R\cup\Delta,B} \Lambda$ and $\sigma|_V \ll_E \rho|_V$.*

Narrowing is complete only with respect to R/E-normalized solutions, as shown by the following example.

Example 5.3 *Consider $\mathcal{R} = (\Sigma, \emptyset, R)$, where Σ has a single sort, and constants a, b, c, d, and a binary function symbol f, and R has the following three rules:*

$$a \to b \qquad a \to c \qquad f(b,c) \to d$$

The reachability goal $G : f(x,x) \to^* d$ has $\sigma = \{a/x\}$ as a solution. But G has neither a trivial solution nor a narrowing derivation starting from it.

We can provide a general procedure which builds a narrowing tree starting from G to find all R/E-normalized solutions.

Theorem 5.4 [47] *Let R be a set of rewrite rules with no extra-variables in the right-hand side. For a reachability goal G, let V be a finite set of variables containing $Var(G)$, and Γ be the set of all substitutions σ, where $G \overset{\sigma}{\rightsquigarrow}^*_{R\cup\Delta,B} \Lambda$. Then Γ is a complete set of solutions of G with respect to R/E-normalized solutions.*

Nodes in this tree correspond to goals, while edges correspond to one-step $R\cup\Delta, B$-narrowing derivations. There are two issues to be addressed here:

(i) Since there can be infinitely long narrowing derivations, the tree has to be traversed in a fair manner to cover all possible narrowing derivations.

(ii) The $\Delta \cup B$-unification algorithm invoked at each node of a tree for $\rightsquigarrow_{R/E}$ can in general return an infinite set of unifiers. By assumption (ii) in Section 4, $\Delta\cup B$-unification has a complete but not necessarily finite unification algorithm. However, the narrowing steps themselves of the general procedure above use only B-unification, which is finite, and thus the enumeration of $\Delta \cup B$-unifiers should be interleaved in a fair manner with the expansion of the narrowing tree. If the rhs's of G are *strongly $\to_{\Delta,B}$-irreducible*, i.e., all their instances by $\to_{\Delta,B}$-normalized substitutions are $\to_{\Delta,B}$-irreducible, then $\Delta \cup B$-unification can be replaced by B-unification.

While this general procedure gives us weak completeness, for it to be useful in practice we need effective narrowing strategies that drastically cut down the search space by expanding only relevant (or necessary) parts of the narrowing tree. We discuss this topic in Section 6.2.

6 Completeness and Computational Efficiency Issues

6.1 Completeness

The crucial reason for losing completeness is that, by definition, narrowing can be performed only at non-variable positions, and therefore cannot account for rewrites that occur within the solution (i.e. under variable positions) [6]. Such "under-the-feet" rewrites can have non-trivial effects if the rewrite rules or the reachability goal are non-linear, and the rules are not confluent.

A natural question to ask is whether the simple narrowing procedure described above is complete for specific classes of rewrite theories. In [47] we have identified several useful classes of rewrite theories for which the naive narrowing procedure can find *all* solutions, and have applied these results to verify safety properties of cryptographic protocols. One such class is that of so-called "topmost" rewrite theories, that includes: (i) most object-oriented systems; (ii) a wide range of Petri net models; and (iii) many reflective distributed systems [46]. Another such class is one where the rewrite rules are right linear and the reachability goal is linear.

In [58], we establish a completeness result of a much broader scope by generalizing narrowing to *back-and-forth-narrowing* for solving reachability goals. Back-and-forth narrowing is *complete* in the *solvability* sense, i.e., it is guaranteed to find a solution when there is one. The back-and-forth procedure is very general, in the sense that there are absolutely no assumptions on the given rewrite system $\mathcal{R} = (\Sigma, \emptyset, R)$. In particular, the rewrite rules in R need *not* be left or right linear, or confluent, or terminating, and can also have *extra variables* in the right-hand side.

In back-and-forth narrowing, we:

- generalize the basic narrowing step through *linearization* of the term being narrowed, and
- use a *combination of forward and backward narrowing* with this generalized relation.

Specifically, we account for under-the-feet rewrites by defining an *extended narrowing step* that is capable of "skipping" several such rewrites and capturing the first rewrite that occurs at a non-variable position. This is achieved by *linearizing* a term before narrowing it with a rule. The intermediate under-the-feet rewrites that have thus been skipped will be accounted for by extending the reachability goal with appropriate subgoals. For example, consider the reachability goal $\exists x.\ f(x,x) \rightarrow^* d$ in Example 5.3 again. We: (i) linearize the term $f(x,x)$ to, say, $f(x_1,x_2)$, (ii) narrow the linearized term with the rule $f(b,c) \rightarrow d$ and the unifier $\{b/x_1, c/x_2\}$, and (iii) extend the reachability goal with subgoals $x \rightarrow^* b$ and $x \rightarrow^* c$. This gives us the new reachability goal $\exists x.\ d \rightarrow^* d \wedge x \rightarrow^* b \wedge x \rightarrow^* c$.

However, linearization alone is not enough, in general, to regain completeness:

[6] One could of course generalize the definition of narrowing to allow narrowing steps at variable positions. But that would make the narrowing procedure very inefficient since, in general, we would have to perform *arbitrary* instantiations of variables.

we also need to use the "back-and-forth" idea. For example, consider a goal $\exists \overrightarrow{x}.t \to^* t'$, where the solution σ is such that any rewrite sequence $\sigma(t) \to^* \sigma(t')$ is such that none of the rewrites occur at non-variable positions of t. But observe that if at least one of these rewrites occurs at a non-variable position in t', then we can narrow the right side t' in the *backward* direction, i.e. using R^{-1}, to obtain a simpler goal. For instance, in the goal $\exists x.\ d \to^* d \wedge x \to^* b \wedge x \to^* c$ above, backward narrowing gives us the goal $\exists x.\ d \to^* d \wedge x \to^* a \wedge x \to^* a$, which has the unifier (solution) $\{a/x\}$.

In general, backward narrowing might in turn enable *forward* narrowing steps using R on the left-hand side, and so on, until we reach a point where all the rewrites occur under variable positions of both the left-hand and right-hand sides. In this case, however, the left-hand and right-hand sides are unifiable, and we are therefore done. For the simple example considered above, however, note that just backward narrowing with R^{-1}, even without any linearization, gives us the solution as follows: $d \rightsquigarrow_{id} f(b,c) \rightsquigarrow_{id} \rightsquigarrow_{id} f(a,a)$. But in [58] we present examples showing that a combination of forward and backward narrowing is indeed necessary, in that neither direction is complete by itself.

6.2 Narrowing Strategies

An important problem for narrowing to be effective in practice is to devise *strategies* that improve its efficiency. Otherwise, one could quickly face a combinatorial explosion in the number of possible narrowing sequences. When several narrowing derivations are possible for the same solution, the question is whether there is a preferred strategy and whether a standardization result is possible.

We have been working on two strategy approaches to explore the search space in a smart manner: (i) the natural narrowing strategy [21]; and (ii) the grammar-based narrowing strategy for cryptographic protocol verification [20].

6.2.1 Natural narrowing

In a recent work [21], we have proposed a narrowing strategy, called natural narrowing, which allows rewrite theories $\mathcal{R} = (\Sigma, \emptyset, R)$ that can be non-left-linear, non-constructor-based, non-terminating, and non-confluent. This strategy improves all the previous state-of-the-art narrowing strategies within the functional logic setting, and it is complete in the weak sense that it is guaranteed to find all R-normalized solutions. We give the reader an intuitive feeling for how natural narrowing works.

Example 6.1 [21] Consider the following rewrite system for proving equality ($\approx$) of arithmetic expressions built using modulus or remainder (%), subtraction ($-$), and minimum (`min`) operations on natural numbers.

```
(1) M % s(N) → (M−s(N)) % s(N)        (5) min(0, N) → 0
(2) (0 − s(M)) % s(N) → N − M         (6) min(s(N),0) → 0
(3) M − 0 → M                         (7) min(s(N),s(M)) → s(min(M,N))
(4) s(M) − s(N) → M−N                 (8) X ≈ X → true
```

Note that this rewrite system is not left-linear because of rule (8), and it is not constructor-based because of rule (2). Furthermore, note that it is neither terminating nor confluent due to rule (1).

Consider the term[7] $t = \infty \,\%\, \texttt{min(X,X-0)} \approx \infty \,\%\, \texttt{0}$ and the following two narrowing sequences. Only these two are relevant, while evaluation of subterm ∞ may run into problems. First, the following sequence leading to `true`, that starts by unifying subterm $t|_{1.2}$ with left-hand side (lhs) (5):

$$\infty \,\%\, \texttt{min(X,X-0)} \approx \infty \,\%\, \texttt{0} \leadsto_{[\texttt{X} \mapsto \texttt{0}]} \infty \,\%\, \texttt{0} \approx \infty \,\%\, \texttt{0} \leadsto_{id} \texttt{true}$$

Second, the following sequence not leading to `true`, that starts by reducing subterm $t|_{1.2.2}$ with lhs (3) and that early instantiates variable `X`:

$$\infty \,\%\, \texttt{min(X,X-0)} \approx \infty \,\%\, \texttt{0} \leadsto_{[\texttt{X} \mapsto \texttt{s(X')}]} \infty \,\%\, \texttt{min(s(X'),s(X'))} \approx \infty \,\%\, \texttt{0}$$
$$\leadsto_{id} \infty \,\%\, \texttt{s(min(X',X'))} \approx \infty \,\%\, \texttt{0}$$

The key points to achieve these optimal evaluations are:

(i) (*Demanded positions*). This notion is relative to a left-hand side (lhs) l and determines which positions in a term t should be narrowed in order to be able to match l at a root position. For the term $\infty \,\%\, \texttt{min(X,X-0)} \approx \infty \,\%\, \texttt{0}$ and lhs $\texttt{X} \approx \texttt{X}$, only subterm `min(X,X-0)` is demanded.

(ii) (*Failing term*). This notion is relative to a lhs l and stops further wasteful narrowing steps. Specifically, the last term $\infty \,\%\, \texttt{s(min(X',X'))} \approx \infty \,\%\, \texttt{0}$ of the second former sequence fails w.r.t. lhs (8), since the subterm `s(min(X',X'))` is demanded by (8) but there is a clash between symbols `s` and `0`.

(iii) (*Most frequently demanded positions*). This notion determines those demanded positions w.r.t. non-failing lhs's that are demanded by the maximum number of rules and that cover all such non-failing lhs's. It provides the optimality properties. If we look closely at lhs's (5), (6), and (7) defining `min`, we can see that the first argument is demanded by the three rules, whereas the second argument is demanded only by (6) and (7). Thus, subterm `X` at the first argument of `min` in term $\infty \,\%\, \texttt{min(X,X-0)} \approx \infty \,\%\, \texttt{0}$ is the most frequently demanded position. Note that this subterm is a variable; this motivates the last point.

(iv) (*Lazy instantiation*). This notion relates to an incremental construction of unifiers without the explicit use of a unification algorithm. This is necessary in the previous example, since subterm `min(X,X-0)` does not unify with lhs's l_6 and l_7. However, we can deduce that narrowing at subterm `X-0` is only necessary when substitution $[\texttt{X} \mapsto \texttt{s(X')}]$, inferred from l_6 and l_7, has been applied. Thus, we early construct the appropriate substitutions $[\texttt{X} \mapsto \texttt{0}]$ and $[\texttt{X} \mapsto \texttt{s(X')}]$ in order to reduce the search space.

6.2.2 A Grammar-based Strategy for Protocol Verification

In work of Escobar, Meadows, and Meseguer [20], the grammar techniques used by Meadows in her NRL Protocol Analyzer (NPA) [41] have been placed within the general narrowing setting proposed in this paper and have been implemented in Maude, in what we call the Maude-NPA tool. For narrowing to be a practical tool for such protocol analysis we need efficient strategies that drastically cut down the

[7] The subterm ∞ represents an expression that has a remarkably high computational cost.

search space, since protocols have typically an infinite search space and are highly non-deterministic.

The NRL Protocol Analyzer [41] is a tool for the formal specification and analysis of cryptographic protocols that has been used with great effect on a number of complex real-life protocols. One of the most interesting of its features is that it can be used, not only to prove or disprove authentication and secrecy properties using the standard Dolev-Yao model [16], but also to reason about security in face of attempted attacks on low-level algebraic properties of the functions used in a protocol. Maude-NPA's ability to reason well about these low-level functionalities is based on its combination of symbolic reachability analysis using narrowing modulo algebraic axioms E, together with its grammar-based techniques for reducing the size of the search space. On one hand, unification modulo algebraic properties (e.g., encryption and decryption, concatenation and deconcatenation) as narrowing using a finite convergent (i.e., confluent and terminating) set of rewrite rules where the right-hand side of each rule is either a subterm of the left-hand side or a ground term [15] allows the tool to represent behavior which is not captured by the usual Dolev-Yao free algebra model. On the other hand, techniques for reducing the size of the search space by using inductively defined co-invariants describing states unreachable to an intruder allow us to start with an infinite search space and reduce it in many cases to a finite one, thus freeing us from the requirement to put any a priori limits on the number of sessions.

Example 6.2 Consider a protocol $\mathcal{R} = (\Sigma, E, R)$ with only two operations, encryption, represented by $e(K, X)$, and decryption, represented by $d(K, X)$, where K is the key and X is the message. Suppose that each time an honest principal A receives a message X, it outputs $d(k, X)$, where k is a constant standing for a key shared by all honest principals. We can denote this by a protocol rule $X \to d(k, X)$ in R. Encryption and decryption usually satisfy the following cancellation properties E: $d(K, e(K, X)) = X$ and $e(K, d(K, X)) = X$. In order to keep the example simple, we assume that the intruder does not perform operations, so no extra intruder rules are added to R. Suppose now that we want to find out how an intruder can learn a term m that does not know initially. The NPA uses backwards search, so we ask what rules could produce m, and how. According to the honest principal rule $X \to d(k, X)$ and the property $d(K, e(K, X)) = X$, we have that the intruder can learn m only if it previously knows $e(k, m)$. That is, we consider the rule application $e(k, m) \to d(k, e(k, m))$, where $d(k, e(k, m)) =_E m$. We then ask the Maude-NPA how the intruder can learn $e(k, m)$, and we find that this can only happen if the intruder previously knows $e(k, e(k, m))$. We see a pattern emerging, which suggests a set of terms belonging to the following formal tree language $\mathcal{L}$:

$$L \mapsto m$$

$$L \mapsto e(k, L)$$

We can verify the co-invariant stating that intruder knowledge of any member of $\mathcal{L}$ implies previous knowledge of some member of $\mathcal{L}$, therefore being impossible for

an intruder to learn any member of $\mathcal{L}$ from an initial state in which does not know any messages.

This defines a backwards narrowing strategy where we discard any protocol state in which the intruder has to learn some term in the grammar $\mathcal{L}$, since this would lead to a useless backwards search path. For more details see [20] and Section 7.1.

7 Applications

In this section we show how narrowing can be used as a unified mechanism for programming and proving. We consider the following scenarios:

(i) The use of narrowing reachability analysis in security protocol verification (Section 7.1).

(ii) Various uses of narrowing in theorem proving, e.g., for improving equational unification, for inductive theorem proving, and for automatic, inductionless induction theorem proving (Section 7.2).

(iii) Within functional logic programming, the use of narrowing in broader classes of programs not supported by current functional logic programming languages, e.g., narrowing modulo AC axioms, removing left-linearity or constructor-based requirements, nonconfluent systems, etc. (Section 7.3).

(iv) Many formal techniques use some form of narrowing. For example, inductionless induction and partial evaluation do so. These techniques can yield better results when more efficient narrowing strategies are used (Sections 7.2.3 and 7.3.1).

The grammar-based strategy of Section 6.2.2 supports scenario (i), whereas the natural narrowing strategy of Section 6.2.1 supports scenarios (ii), (iii), and (iv).

7.1 Security Protocol Verification

Verification of many security protocol properties can be formulated as solving reachability problems. For instance, verifying the *secrecy* property of a protocol amounts to checking whether the protocol can reach a state where an intruder has discovered a data item that was meant to be a secret. In this section we show how the strong completeness of narrowing for topmost rewrite theories, together with the grammar-based strategy explained in Section 6.2.2, can be exploited to get a generic and complete procedure for the analysis of such security properties *modulo* algebraic properties of the cryptographic functions.

Example 7.1 Consider the well-known Needham-Schroeder protocol [48] that uses public keys to achieve authentication between two parties, Alice and Bob. The protocol is specified in Maude[8] according to the framework described in [20]:

[8] The Maude syntax is so close to the corresponding mathematical notation for defining rewrite theories as to be almost self-explanatory. The general point to keep in mind is that each item: a sort, a subsort, an operation, an equation, a rule, etc., is declared with an obvious keyword: `sort`, `subsort`, `op`, `eq` (or `ceq` for conditional equations), `rl` (or `crl` for conditional rules), etc., with each declaration ended by a space

```
fmod PROTOCOL-SYMBOLS is
  --- Importing Auxiliary Sorts: Msg, Fresh, Strands, ...
  protecting DEFINITION-PROTOCOL-RULES .

  --- Sort Information
  sorts Name Nonce Key Enc .
  subsort Name Nonce Enc Key < Msg .
  subsort Name < Key .

  --- Encoding operators for public/private encryption
  op pk : Key Msg -> Enc .
  op sk : Key Msg -> Enc .

  --- Nonce operator
  op n : Name Fresh -> Nonce .

  --- Intruder's name
  op i : -> Name .

 --- Associativity operator
  op _;_ : Msg  Msg  -> Msg .

  *** Encryption/Decryption Cancellation Algebraic Properties
  eq pk(Ke:Key,sk(Ke:Key,Z:Msg)) = Z:Msg .
  eq sk(Ke:Key,pk(Ke:Key,Z:Msg)) = Z:Msg .

endfm

mod PROTOCOL-STRANDS-RULES is
  protecting PROTOCOL-SYMBOLS .

  var SS : StrandSet .
  var K : IntruderKnowledge .
  vars L ML L1 L2 : SMsgList .
  vars M M1 M2 : Msg .
  vars A B : Name .
  var Ke : Key .
  var r : Fresh .
  var N : Nonce .

  *** General rule: Accept input message
  rl [ L1 | -(M), L2 ] & SS & {M inI, K} => [ L1, -(M) | L2 ] & SS & {M inI, K} .

  *** General rule: Accept output message
  rl [ L1 | +(M), L2 ] & SS & {M !inI, K} => [ L1, +(M) | L2 ] & SS {M inI, K} .

  *** General rule: Accept output message
  rl [ L1 | (+(M), L2) ]) & SS & K => [ L1, +(M) | L2 ] & SS & K .

  *** Dolev-Yao Intruder Rules
  rl ([ -(M1)), -(M2) | +(M1 ; M2) ] & SS & {(M1 ; M2) !inI, K} => SS & {(M1 ; M2) inI, K} .

  rl ([ -(M1 ; M2) | +(M1) , +(M2) ] & SS & {M1 !inI, K} => SS & {M1 inI, K} .

  rl ([ -(M1 ; M2) | +(M2) , +(M1) ] & SS & {M2 !inI, K} => SS & {M2 inI, K} .

  rl ([ -(M) | +(sk(i, M)) ] & SS & {sk(i, M) !inI, K} => SS & {sk(i, M) inI, K} .

  rl ([ -(M) | +(pk(Ke, M)) ] & SS & {pk(Ke, M) !inI, K} => SS & {pk(Ke, M) inI, K} .

  rl ([ nil | +(A) ] & SS & {A !inI, K} => SS & {A inI, K} .

  *** Initiator
  rl [ nil | +(pk(B, A ; n(A, r))), -(pk(A, n(A, r) ; N)), +(pk(B, N)) ] & SS
       & {pk(B, A ; n(A, r)) !inI, K}
  => SS & {pk(B, A ; n(A, r)) inI, K} .

  rl [ +(pk(B, A ; n(A, r))), -(pk(A, n(A, r) ; N)) | +(pk(B, N)) ] & SS & {pk(B,N) !inI, K}
  => SS & {(B,N) inI, K} .

  *** Responder
  rl [ -(pk(B,A ; N)) | +(pk(A,N ; n(B,r))), -(pk(B,n(B,r))) ] & SS
       & {pk(A,N ; n(B,r)) !inI, K}
  => SS & {pk(A,N ; n(B,r)) inI, K} .
```

and a period. Indeed, a rewrite theory $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ is defined with the signature Σ using keyword `op`, equations in Δ using keyword `eq`, axioms in B using keywords `assoc`, `comm` and `id:`, and rules in R using keyword `rl`. Another important point is the use of "mix-fix" user-definable syntax, with the argument positions specified by underbars; for example: `if_then_else_fi`.

```
endm
```

In this Maude specification, a nonce, i.e., a random number sent by one principal to another to ensure confidentially, is denoted by `n(A,r)`, where A is the name of the principal and r is the randomly generated number. Concatenation of two messages is denoted by the operator `_;_`, e.g., `n(A,r);n(B,r')`. Encryption of a message M with the public key of principal A is denoted by `pk(A,M)`, e.g., `pk(A,n(S,r);S)`. Encryption of a message with a private key is denoted by `sk(A,M)`, e.g., `sk(A,n(S,r);S)`. The name of the intruder is fixed and denoted by constant `i`. The only secret key operation the intruder can perform is `sk(i,m)` for a known message `m`.

The protocol is described using strands [22]. A *state* of the protocol is a set of strands (with & the associative and commutativity union operator) and the *intruder knowledge* at that point, which is enclosed within curly brackets and contains two kinds of facts: positive knowledge facts, denoted by (m `inI`), and negative knowledge facts, denoted by (m `!inI`). A *strand* denotes the sequence of input messages (denoted by $-(M)$ or M^-) and output messages (denoted by $+(M)$ or M^+) that a principal performs. Different sessions of the same protocol can be run in parallel just by having different strands in the set of strands. The protocol is described as the following set of strands [20]:

(i) $[pk(B, A; n(A, r))^+, pk(A, n(A, r); Z)^-, pk(B, Z)^+]$
This strand represents principal A initiating the protocol by sending his/her name and a nonce, both encrypted with B's public key, to B in the first message. Then A receives B's response and sends a final message consisting of the rest of the message received from B.

(ii) $[pk(B, A; W)^-, pk(A, W; n(B, r'))^+, pk(B, n(B, r'))^-]$
This strand represents principal B receiving A's first message, checking that it is the public key encryption of A's name concatenated with some value W, and then sending to A the concatenation of that value W with B's own nonce, encrypted with A's public key. Then, B receives the final message from A and verifies that the final message that it receives has B's nonce encrypted with B's public key.

Together with the intruder capabilities to concatenate, deconcatenate, encrypt and decrypt messages according to the Dolev-Yao attacker's capabilities [16]:

(iii) $[M_1^-, M_2^-, (M_1; M_2)^+]$ Concatenation of two messages into a message.
(iv) $[(M_1; M_2)^-, M_1^+, M_2^+]$ Extraction of two concatenated messages.
(v) $[M^-, pk(Y, M)^+]$ Encryption of a message with a public key.
(vi) $[M^-, sk(i, M)^+]$ Encryption of a message with the intruder's secret key.

All these strands give rise to backwards rewrite rules describing their effect on the intruder's knowledge as shown in the above Maude specification. There are also three general rules for accepting input and output messages. As explained in [20], we then perform a backwards narrowing reachability analysis, i.e., we provide a reachability goal from a final state pattern describing an attack, like

$$\cdots \ \& \ [m_1, m_2, \ldots, m_k \mid nil] \ \& \ \cdots \ \& \ \{(m'_1 \ \texttt{inI}), \ldots, (m'_n \ \texttt{inI})\}$$

to an initial state pattern of the form

$$\cdots \ \& \ [nil \mid m_1, m_2, \ldots, m_k] \ \& \ \cdots \ \& \ \{(m'_1 \ \texttt{!inI}), \ldots, (m'_n \ \texttt{!inI}), \ldots, (m'_{n+j} \ \texttt{!inI})\}$$

where the initial state may contain more strands and more terms m' to be known in the future (m' `!inI`) in the intruder knowledge than the final state, since they may have been added during the backwards narrowing process.

Consider, for example, the following final attack state pattern (where Z is a variable of sort Msg, and A, B are variables of sort Name):

$$[\ pk(B, A; Z)^-, \ pk(A, Z; n(B, r'))^+, \ pk(B, n(B, r'))^- \mid nil\ ] \ \& \ \{\ (n(B, r') \ \texttt{inI})\ \}$$

which represents a situation where B has completed the expected communication with someone (i.e., A) and the intruder has learned B's nonce. For this insecure goal state, the reachability analysis returns several possible solutions, including the following initial state corresponding to Lowe's attack [39] (note the new strands and the new terms in the intruder knowledge):

$$\begin{array}{l}
[\ nil \mid pk(i, A; n(A, r))^+, \ pk(A, n(A, r); n(B, r'))^-, \ pk(i, n(B, r'))^+\ ] \ \& \\
[\ nil \mid pk(i, A; n(A, r))^-, \ (A; n(A, r))^+\ ] \ \& \\
[\ nil \mid (A; n(A, r))^-, \ pk(B, (A; n(A, r))^+\ ] \ \& \\
[\ nil \mid pk(B, A; n(A, r))^-, \ pk(A, n(A, r); n(B, r'))^+, \ pk(B, n(B, r'))^-\ ] \ \& \\
[\ nil \mid pk(i, n(B, r'))^-, \ n(B, r')^+\ ] \ \& \\
[\ nil \mid n(B, r')^-, \ pk(B, n(B, r'))^+\ ] \ \& \\
\{\ (n(B, r') \ \texttt{!inI}), \ (pk(i, n(B, r')) \ \texttt{!inI}), \ (pk(A, n(A, r); n(B, r')) \ \texttt{!inI}), \\
\quad (pk(i, A; n(A, r)) \ \texttt{!inI}), \ ((A; n(A, r)) \ \texttt{!inI}), \ (pk(B, (A; n(A, r)) \ \texttt{!inI}), \\
\quad (pk(B, n(B, r')) \ \texttt{!inI}) \qquad \}
\end{array}$$

Note that, in order to define an effective mechanism to find the previous attack, we have to detect and avoid many irrelevant paths (usually infinite in depth). For instance, we should avoid the following infinite backwards narrowing sequence generated by the Dolev-Yao strand for deconcatenation shown above,

$$\begin{array}{l}
[\ldots, m^- \mid \ldots] \\
\rightsquigarrow [\ldots \mid m^-, \ldots] \ \& \ [(M_1; m)^- \mid m^+, M_1^+] \\
\rightsquigarrow [\ldots \mid m^-, \ldots] \ \& \ [nil \mid (M_1; m)^-, m^+, M_1^+] \ \& \ [(M_2; (M_1; m))^- \mid (M_1; m)^+, M_2^+] \\
\rightsquigarrow [\ldots \mid m^-, \ldots] \ \& \ [nil \mid (M_1; m)^-, m^+, M_1^+] \ \& \ [nil \mid (M_2; (M_1; m))^-, (M_1; m)^+, M_2^+] \\
\quad \& \ [(M_3; (M_2; (M_1; m)))^- \mid (M_2; (M_1; m))^+, M_3^+] \\
\rightsquigarrow \cdots
\end{array}$$

which shows that the intruder learnt a message $m_1; \cdots; m_n; m$ in a previous state that he/she is decomposing to learn m. Indeed, this useless search and many similar ones are avoided using a grammar-based strategy (see Section 6.2.2). Furthermore, for many protocols backwards narrowing with a grammar-based strategy terminates, allowing full verification of security properties.

7.2 Theorem Proving

We consider three aspects related to theorem proving where narrowing reachability analysis using a convergent equational theory E is relevant:

- equational unification problems, i.e., solving $(\exists \overrightarrow{x})\ t(\overrightarrow{x}) = t'(\overrightarrow{x})$,
- inductive theorem proving, i.e., solving $E \vdash_{ind} (\exists \overrightarrow{x})\ t = t'$, and
- automatic proof by inductionless induction of goals $E \vdash_{ind} (\forall \overrightarrow{x})\ t = t'$.

7.2.1 Equational unification

Narrowing was originally introduced as a complete method for generating all solutions of an equational unification problem, i.e., for goals F of the form

$$(\exists \overrightarrow{x})\ t_1(\overrightarrow{x}) = t'_1(\overrightarrow{x}) \wedge \ldots \wedge t_n(\overrightarrow{x}) = t'_n(\overrightarrow{x})$$

in free algebras modulo a set E of convergent equations [23,33,35]. As already pointed out in the Introduction, solving E-unification problems such as the goal F above, is equivalent to solving by narrowing the reachability goal $G = (\exists \overrightarrow{x})\ t_1 \approx t'_1 \rightarrow^* \mathtt{true} \wedge \ldots \wedge t_n \approx t'_n \rightarrow^* \mathtt{true}$ in the rewrite theory $\mathcal{R}_{\mathcal{E}} = (\tilde{\Sigma}, \emptyset, R_E)$, where $\tilde{\Sigma} = \Sigma \cup \{\approx, \mathtt{true}\}$ and $R_E = \overrightarrow{E} \cup \{\ x \approx x \rightarrow \mathtt{true}\}$, with $\overrightarrow{E}$ the rules obtained by orienting the equations E. Even in this traditional setting, our techniques can be useful. For example, many convergent equational theories fail to satisfy the left-linearity and constructor-based requirements; but no such restrictions apply to natural narrowing.

7.2.2 Inductive theorem proving

The just-described reduction of existential equality goals to reachability goals has important applications to *inductive theorem proving*. Specifically, it is useful in proving existentially quantified inductive theorems like $E \vdash_{ind} (\exists \overrightarrow{x})\ t = t'$ in the initial model, i.e., in the minimal Herbrand model of E satisfies $(\exists \overrightarrow{x})\ t = t'$. As it is well-known (see, e.g., [27])

$$E \vdash (\exists \overrightarrow{x})\ t = t' \quad \Leftrightarrow \quad E \vdash_{ind} (\exists \overrightarrow{x})\ t = t'$$

therefore, narrowing is an *inductive* inference method. An effective narrowing strategy, such as natural narrowing, can provide a very effective semidecision procedure for proving such inductive goals, because it will *detect failures* to unify, stopping with a counterexample instead of blindly expanding the narrowing tree. In particular, an effective narrowing strategy can be added to inductive provers such as Maude's ITP [11]. In cases when equational narrowing is ensured to terminate (see [49]), an effective narrowing strategy can also be used to prove *universal* inductive goals of the form $E \vdash_{ind} (\forall \overrightarrow{x})\ C \Rightarrow t = t'$, with C a conjunction of equations, because we can reduce proving such a goal to first solving $E \vdash_{ind} (\exists \overrightarrow{x})\ C$ by narrowing, and then proving $E \vdash_{ind} (\forall \overrightarrow{x})\ \sigma(t) = \sigma(t')$ for each of the solutions σ found for C.

7.2.3 Inductionless Induction

An effective narrowing strategy can also be useful in proving universal inductive theorems of the form $E \vdash_{ind} (\forall \overrightarrow{x})\ t = t'$, even in the case where equational narrowing is not guaranteed to terminate. Specifically, an effective narrowing strategy

can be fruitfully integrated in automatic techniques that use narrowing for proving or refuting such goals, such as inductionless induction [12].

Inductionless induction aims at automatically proving universal inductive theorems without using an explicit induction scheme, as in implicit induction techniques such as [8,53]. It simplifies the task by using classical first-order theorem provers which are refutation-complete and saturation-based, and a dedicated technique to ensure (in-)consistency. Given a set C of equations that are to be proved or refuted (also understood as conjectures) in the initial model for a set of equations E, the technique considers a (first-order) axiomatization $\mathcal{A}$ of the initial model that represents the "negative" information about inequalities in the model. The key fact that is exploited is that the conjectures C are inductive theorems if and only if $C \cup \mathcal{A} \cup E$ is consistent. This consistency check is in turn performed in two stages [9]: (i) first the logical consequences of C using E are computed, and (ii) the consistency of each such consequence with $\mathcal{A}$ is checked using a refutationally complete theorem-prover. The conjectures are true if and only if there are no logical consequences that are inconsistent with $\mathcal{A}$.

The relevant point in this inductionless induction technique is that deductions of C are computed by a (not very restricted) version of narrowing called *superposition*, with some additional tactics to eliminate *redundant* (or irrelevant) consequences. Specifically, the conjectures in C are narrowed using oriented equations E to obtain the logical consequences. The consistency of each such non-redundant consequence with $\mathcal{A}$ is checked as before, until the superposition does not return any more non-redundant consequences. The point is that a narrowing strategy can be used instead of superposition (under some restrictions not discussed here) to compute a smaller set of deductions, which can increase the chances of termination of the procedure above, without loss of soundness.

We briefly recall a few more details about the inductionless induction method (see [12] for additional details). We assume below that $\succeq$ is a reduction ordering which is total on ground terms, i.e. a relation $\succ$ which is irreflexive, transitive, well-founded, total, monotonic, and stable under substitutions. A well-known reduction ordering is the *recursive path ordering*, based on a total ordering $\succ_\Sigma$ (called *precedence*) on Σ. The following is the inference rule defining the *superposition* strategy (note that $l = r$ is symmetric).

$$\text{Superposition} \quad \frac{l = r \qquad c[s]_p}{\sigma(c[r]_p)} \quad \begin{array}{l} \text{if } \sigma = mgu(l, s), s \text{ is not a variable,} \\ \sigma(r) \not\succeq \sigma(l), l = r \in E, c[s]_p \in C. \end{array}$$

Given a ground equation c, $C^{\prec c}$ is the set of ground instances of equations in C that are strictly smaller than c in this ordering. A ground conjecture c is *redundant* in a set of conjectures C if $E \cup \mathcal{A} \cup C^{\prec c} \vdash c$. A non-ground conjecture is redundant if all its ground instances are. An inference is *redundant* if one of its premises or its conclusion are redundant in C.

[9] Under suitable assumptions about E and $\mathcal{A}$ (see [12]).

Example 7.2 Consider the problem of coding the rather simple inequality $x+y+z+w \leqslant h$ for natural numbers x, y, z, w, h, borrowed from [18], which is specified in the following Maude module.

```
mod SOLVE is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat .

  vars X Y Z W H : Nat .

  op _+_ : Nat Nat -> Nat .
  rl X + 0 => X .
  rl X + s(Y) => s(X) + Y .

  op _<+>_<+>_<+>_<=_ : Nat Nat Nat Nat Nat -> Bool .
  rl X <+> s(Y) <+> s(Z) <+> W <= s(H) => X <+> Y <+> s(Z) <+> W <= H .
  rl X <+> 0 <+> s(Z) <+> W <= s(H) => X <+> 0 <+> Z <+> W <= H .
  rl s(X) <+> Y <+> 0 <+> W <= s(H) => X <+> Y <+> 0 <+> W <= H .
  rl 0 <+> s(Y) <+> 0 <+> W <= s(H) => 0 <+> Y <+> 0 <+> W <= H .
  rl 0 <+> 0 <+> 0 <+> s(W) <= s(H) => 0 <+> 0 <+> 0 <+> W <= H .
  rl 0 <+> 0 <+> 0 <+> 0 <= H => true .
  rl 0 <+> 0 <+> 0 <+> s(W) <= 0 => false .
  rl s(X) <+> Y <+> 0 <+> 0 <= 0 => false .
  rl X <+> 0 <+> s(Z) <+> W <= 0 => false .
  rl 0 <+> s(Y) <+> Z <+> 0 <= 0 => false .
  rl 0 <+> s(Y) <+> 0 <+> s(W) <= 0 => false .
  rl s(X) <+> s(Y) <+> s(Z) <+> 0 <= 0 => false .
endm
```

Consider also the following conjecture to be proved

$$(c_1)\forall X, Y, Z : \texttt{(0 <+> Z <+> s(0) <+> (X + Y) <= Z) = (0 <+> Z <+> s(0) <+> (Y + X) <= Z)}$$

Inductionless induction with superposition cannot prove this conjecture, because it goes into a loop by generating infinitely many non-redundant logical consequences of the conjecture. These consequences are generated when the subterms `Y + X` and `X + Y` are narrowed by such not very restricted narrowing. However, the natural narrowing strategy never narrows the above subterms, resulting in only finitely many non-redundant consequences. In this way, inductionless induction can prove the conjecture.

The rules (which in this context are understood as equations and have to be oriented) are correctly oriented using the recursive path ordering and the precedence $(\texttt{\_<+>\_<+>\_<+>\_<=\_}) \succ_\Sigma + \succ_\Sigma \texttt{s} \succ_\Sigma \texttt{0} \succ_\Sigma \texttt{false} \succ_\Sigma \texttt{true}$. This means that for every rule $l \to r \in \mathcal{R}$ and every substitution σ, $\sigma(l) \succ \sigma(r)$. Note also that the rules are confluent, terminating, and sufficiently complete [10], which are conditions necessary for the inductionless induction technique. Consider the following axiomatization $\mathcal{A}$ of the initial model for the equations under consideration, also necessary for refutation in the inductionless induction technique:

$$\texttt{true} \neq \texttt{false} \qquad \texttt{s(X)} \neq \texttt{0} \qquad \texttt{s(X) = s(Y)} \Rightarrow \texttt{X = Y}$$

Now, we try to prove the conjecture. We have the following consequences if we perform superposition starting from the conjecture c_1:

[10] See [57] for details about sufficient completeness.

(c_2) `false = (0 <+> 0 <+> s(0) <+> (Y + X) <= 0)` at position ϵ in the left side with $[\mathtt{Z} \mapsto 0]$

(c_3) `(0 <+> 0 <+> s(0) <+> (X + Y) <= 0) = false` at position ϵ in the right side with $[\mathtt{Z} \mapsto 0]$

(c_4) `(0 <+> Z' <+> s(0) <+> (X + Y) <= Z') = (0 <+> s(Z') <+> s(0) <+> (Y + X) <= s(Z'))` at position ϵ in the left side with $[\mathtt{Z} \mapsto \mathtt{s(Z')}]$

(c_5) `(0 <+> s(Z') <+> s(0) <+> (X + Y) <= s(Z')) = (0 <+> Z' <+> s(0) <+> (Y + X) <= Z')` at position ϵ in the right side with $[\mathtt{Z} \mapsto \mathtt{s(Z')}]$

(c_6)`(0 <+> Z <+> s(0) <+> X <= Z) = (0 <+> Z <+> s(0) <+> (0 + X) <= Z)`

at position 4 in the left side with $[\mathtt{Y} \mapsto 0]$

(c_7)`(0 <+> Z <+> s(0) <+> (s(X) + Y') <= Z) = (0 <+> Z <+> s(0) <+> (s(Y') + X) <= Z)` at position 4 in the left side with $[\mathtt{Y} \mapsto \mathtt{s(Y')}]$

(c_8)`(0 <+> Z <+> s(0) <+> (0 + Y) <= Z) = (0 <+> Z <+> s(0) <+> Y <= Z)`

at position 4 in the right side with $[\mathtt{X} \mapsto 0]$

(c_9)`(0 <+> Z <+> s(0) <+> (s(X') + Y) <= Z) = (0 <+> Z <+> s(0) <+> (s(Y) + X') <= Z)` at position 4 in the right side with $[\mathtt{X} \mapsto \mathtt{s(X')}]$

Now, the conjectures c_2, c_3, c_4, c_5 are redundant, since they can be simplified to a term of the form $t = t$ by using the (oriented) equations and (oriented) conjecture c_1. Conjectures c_6, c_7, c_8, c_9 are not redundant but consistent w.r.t. $\mathcal{A}$. Narrowing the conjectures c_6, c_7, c_8, c_9 by instantiating the variable `Z` with 0 and `s(Z')`, will give us redundant consequences that are similar to c_2, c_3, c_4, c_5. Apart from these redundant consequences, we have the following consequences for c_6:

(c_{10})`(0 <+> Z <+> s(0) <+> 0 <= Z) = (0 <+> Z <+> s(0) <+> 0 <= Z)`
at position 4 in the right side with $[\mathtt{X} \mapsto 0]$

(c_{11})`(0 <+> Z <+> s(0) <+> s(X') <= Z) = (0 <+> Z <+> s(0) <+> (s(0) + X') <= Z)`
at position 4 in the right side with $[\mathtt{X} \mapsto \mathtt{s(X')}]$

where c_{10} is a trivial conjecture, and thus redundant. Furthermore, we have the following consequences for c_7:

(c_{12})`(0 <+> Z <+> s(0) <+> s(X) <= Z) = (0 <+> Z <+> s(0) <+> (s(0) + X) <= Z)`
at position 4 in the left side with $[\mathtt{Y'} \mapsto 0]$

(c_{13})`(0 <+> Z <+> s(0) <+> (s(s(X)) + Y'') <= Z) = (0 <+> Z <+> s(0) <+> (s(s(Y'')) + X) <= Z)`
at position 4 in the left side with $[\mathtt{Y'} \mapsto \mathtt{s(Y'')}]$

(c_{14})`(0 <+> Z <+> s(0) <+> (s(0) + Y') <= Z) = (0 <+> Z <+> s(0) <+> s(Y') <= Z)`
at position 4 in the right side with $[\mathtt{X} \mapsto 0]$

(c_{15})`(0 <+> Z <+> s(0) <+> (s(s(X')) + Y') <= Z) = (0 <+> Z <+> s(0) <+> (s(s(Y')) + X') <= Z)`
at position 4 in the right side with $[\mathtt{X} \mapsto \mathtt{s(X')}]$

where c_{12} is identical to c_{11} and c_{13} is identical to c_{15}, up to renaming of variables. We have similar consequences for c_8 and c_9, which have not been shown here for simplicity. At this point, the reader can realize that the inductionless induction process will loop forever producing the non-redundant conjectures:

$$\mathtt{(0 <+> Z <+> s(0) <+> s^k(X) <= Z) = (0 <+> Z <+> s(0) <+> (s^k(0) + X) <= Z)}$$
$$\mathtt{(0 <+> Z <+> s(0) <+> (s^k(0) + X) <= Z) = (0 <+> Z <+> s(0) <+> s^k(Y) <= Z)}$$
$$\mathtt{(0 <+> Z <+> s(0) <+> (s^k(X) + Y) <= Z) = (0 <+> Z <+> s(0) <+> (s^k(Y) + X) <= Z)}$$

However, natural narrowing produces only the consequences c_2, c_3, c_4, c_5, since it first lazily instantiates the most frequently demanded variable `Z` with `0` and `s(Z')`, and after either of these instantiations the conjecture c_1 fails with respect to the rules that demand the subterms `Y + X` and `X + Y`. Since all of the conjectures c_2, c_3, c_4, c_5 are redundant, the inductionless induction procedure terminates, thus proving the conjecture c_1.

Another relevant point is the observation that an effective narrowing strategy can be used as the basis of a much simpler inductionless induction procedure, where no inconsistency checks and no use of a first-order theorem prover based on saturation are needed, and only some tactics on top of narrowing to eliminate redundant deductions are necessary. The point is that logical consequences obtained by saturation and the inconsistency check with $\mathcal{A}$ can be implicitly performed by an effective narrowing strategy: for example, with the rule $X \approx X \to \texttt{true}$ added to the set of equations E, inconsistency can be detected as *failure* of an effective narrowing strategy to further narrow a conjecture of the form $t \approx t'$, as shown in the following example. Such an inductionless induction procedure would be applicable to a very broad class of equational theories, namely to theories that are terminating, confluent, and sufficiently complete with respect to constructor symbols $\mathcal{C}$.

Example 7.3 Consider again the rewrite theory of Example 7.2, but now with the conjecture `s(X + Y) = Y`. If we add the rule `X` $\approx$ `X` $\to$ `true` to the rewrite theory and execute the expression `s(X + Y)` $\approx$ `Y` with the natural narrowing strategy, the following sequence `s(X + Y)` $\approx$ `Y` $\leadsto_{[\texttt{Y}\mapsto\texttt{0}]}$ `s(X)` $\approx$ `0` ending in a (failing) expression different from `true` is obtained by the strategy, hence refuting the conjecture.

7.3 Programming Languages

Effective rewriting and narrowing strategies are useful in functional programming languages based on rewriting, such as Haskell [31], and functional logic programming languages based on narrowing, such as Curry [30]. Programs in these languages are commonly represented as left-linear constructor-based rewrite theories.

One of the main issues of concern in all previous approaches [6,5,4,32,55,18,19] has been to extend the class of rewrite theories where the known rewriting and narrowing strategies are applicable. This is crucial in a programming language setting, because the situations to which the basic strategy of the language is applicable determine the range of applications where the programming language can be effectively used. This is one of the main contributions of our work to the programming languages area, since supporting general classes of rewriting theories without left-linearity and constructor-based restrictions or modulo equational properties such as associativity-commutativity (AC) is highly desirable. Programs in equational programming languages such as OBJ, CafeOBJ, ASF+SDF, Maude, and ELAN usually fit into this larger class of rewrite theories. For instance, the program of Example 7.1 cannot be encoded into a programming language such as Curry, since it does not obey the left-linearity and constructor-based restrictions, and furthermore requires the use of equational properties such as AC axioms.

In what follows we focus on one particular technique in programming languages that can benefit from our work: partial evaluation.

7.3.1 Partial Evaluation

Various techniques in programming languages, such as partial evaluation, use some form of narrowing. These techniques can yield better results when a naive, unrestricted narrowing strategy is replaced by a more efficient version of narrowing.

Partial evaluation (PE) is a semantics-based program transformation which specializes a program with respect to known parts of its input [34]. PE has been widely applied in the fields of functional programming [34,56], logic programming [13], and functional logic programming [2,37,1]. The various PE techniques in all these contexts share the common idea of narrowing sequences. Indeed, "partially evaluating" a functional term for which only some of its inputs are known, while the others "unknowns" are represented by logical variables, is essentially the same as narrowing such a term with some added stopping criterion. This is sometimes implicit in the usual PE and supercompilation terminology, where such kind of narrowing is often described by other names such as *driving* [59]. However, the use of narrowing has already been explicitly considered as a unifying framework for PE in [2].

The core of PE is that a finite narrowing tree is produced from the input term t by using a narrowing strategy and an unfolding rule (also understood as a stopping criterion) [2]. Then, the specialized program is obtained from the leaves of this narrowing tree, i.e., informally speaking, we apply the computed substitutions to the initial term, which gives us the left-hand sides of the specialized rules, and take the leaves as the corresponding right-hand sides. Hence, the success of the partial evaluation lies in having a good narrowing strategy and a good unfolding rule. A naive (and not very interesting) version would be to consider unrestricted narrowing and an unfolding rule such as "stop constructing the narrowing tree when it reaches depth k".

We briefly recall the partial evaluation technique of [2]. Given a narrowing sequence $t \leadsto^*_\sigma s$, we call $\sigma(t) \to s$ its *resultant*. A narrowing tree for a term t in a rewrite theory $\mathcal{R}$ is a tree such that the root node is t, and for each term s in the tree we have $t \leadsto^*_\sigma s$. Given a narrowing tree for a term t in which no term with a constructor symbol at top is narrowed in any of the narrowing sequences, the set of resultants associated to the narrowing sequences of the tree is called a *pre-evaluation* of t in $\mathcal{R}$. A pre-evaluation can already be seen as the specialized version of the program for input term t. However, the partial evaluation technique also uses a notion of *S-closedness* of a term t w.r.t. a set of terms S, to ensure that any possible instance of the term t is covered by S. Specifically, for a set S of terms, we say that a term t is S-closed if any subterm $t|_p$ that is rooted by a defined symbol is an instance of a term $s \in S$, i.e. $\exists \sigma : \sigma(s) = t|_p$, and all terms $\sigma(x)$ for $x \in \mathcal{X}$ are also S-closed. We say that a pre-evaluation is S-closed if for all resultants $t_i \to s_i$ in the pre-evaluation, s_i is S-closed. Partial evaluation of a program $\mathcal{R}$ w.r.t. a set of terms S consists of producing an S-closed pre-evaluation of each term $s \in S$ and obtaining the specialized program from all the resultants.

Finally, the specialized program is obtained from the resultants using a mapping from terms to terms in order to simplify the resulting program (see [2] for details).

Our work can be quite useful for PE applications for several reasons. First, efficient narrowing strategies such as natural narrowing can avoid combinatorial explosions while constructing the narrowing tree, which is the main danger in PE algorithms. Second, it can be applied, much more generally than previous PE techniques, to rewrite theories having non-functional semantics, an area where, to the best of our knowledge, PE techniques have not yet been applied. Third, as illustrated by the PE example below, it can result in *smaller*, more effective specialized programs than those obtained using other narrowing strategies.

We show below how partially evaluating programs using more effective narrowing strategies can result in smaller and more effective specialized programs. This was recently illustrated in [3] using the needed narrowing strategy [6]. That is, any narrowing strategy performing better than others can give better specialized programs (under some restrictions not discussed here), as shown in the following example.

Example 7.4 Consider the rewrite theory of Example 7.2. Such program is left-linear and constructor-based. Indeed, it is also orthogonal but many rewriting and narrowing strategies behave badly and cannot provide a unique normal form. Now, consider the input t = `(X + Y) <+> 0 <+> s(s(s(0))) <+> W <= s(s(0))` for which the program above is to be specialized. Clearly, the specialized program should always evaluate to `false`, regardless of what the rest of the input is. Furthermore, t can be rewritten to `false` without ever instantiating the variables `X`, `Y` and `W`, as follows:

```
(X + Y) <+> 0 <+> s(s(s(0))) <+> W <= s(s(0))
  → (X + Y) <+> 0 <+> s(s(0)) <+> W <= s(0)
  →  (X + Y) <+> 0 <+> s(0) <+> W <= 0  →  false
```

Natural narrowing is able to provide this optimal computation, whereas weakly needed narrowing [5] yields an unnecessarily bigger narrowing tree. That is, when we consider the weakly needed narrowing strategy [5], we have the following narrowing sequences from t:

```
(X + Y) <+> 0 <+> s(s(s(0))) <+> W <= s(s(0))
 ⇝id (X + Y) <+> 0 <+> s(s(0)) <+> W <= s(0)
 ⇝id  (X + Y) <+> 0 <+> s(0) <+> W <= 0 ⇝id false
```

```
(X + Y) <+> 0 <+> s(s(s(0))) <+> W <= s(s(0))
 ⇝[Y↦0] X <+> 0 <+> s(s(s(0))) <+> W <= s(s(0))
 ⇝id X <+> 0 <+> s(s(0)) <+> W <= s(0)  ⇝id X <+> 0 <+> s(0) <+> W <= 0 ⇝id false
```

```
(X + Y) <+> 0 <+> s(s(s(0))) <+> W <= s(s(0))
 ⇝[Y↦s(Y')] (s(X) + Y') <+> 0 <+> s(s(s(0))) <+> W <= s(s(0))
```

Instead, natural narrowing returns only the first narrowing sequence, corresponding to the optimal one shown above. Hence, the specialized program that we obtain using natural narrowing in the partial evaluation framework of [2] would be (after the usual renaming stage)

```
solve(X,Y,W) → false
```

which is simpler than the one obtained by using weakly needed narrowing [5]:

```
solve'(X,Y,W) → false
solve'(X,0,W) → false
solve'(X,s(Y'),W) → solve'(s(X),Y',W)
```

8 Conclusions and Future Work

We have explained how narrowing can be generalized from its original equational setting to the much wider setting of rewriting logic. Of course, the equational and functional logic programming worlds are faithfully preserved; but now they can fruitfully interact with a much richer range of new applications.

Much work remains ahead in three main directions: foundations, implementations, and applications. At the foundational level, several natural generalizations seem highly desirable. For example, extending our results to conditional rewrite theories, and generalizing the order-sorted equational setting to membership equational logic [45]. Work on strategies should also be advanced. For example, we should generalize natural narrowing to work *modulo* equational axioms such as associativity and commutativity; and extend grammar-based strategies to a much wider range of theories. At the implementation level, our future work will focus on adding narrowing capabilities to the Maude rewriting logic language and on advancing the Maude-NPA tool. At the applications level, besides continuing our own work on protocol verification, model checking, and automated deduction, we hope that other researchers will become actively engaged with these ideas and will develop new application areas.

Acknowledgments. We cordially thank Catherine Meadows for her fundamental contributions to the security protocol verification ideas that we have summarized as one of the important applications of this work. We thank María Alpuente, Salvador Lucas, and Germán Vidal for their comments about partial evaluation. S. Escobar has been partially supported by the EU (FEDER) and Spanish MEC TIN-2004-7943-C04-02 project, the Generalitat Valenciana under grant GV06/285, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project. J. Meseguer's research has been supported in part by ONR Grant N00014-02-1-0715.

References

[1] Albert, E. and G. Vidal, *The Narrowing-driven Approach to Functional Logic Program Specialization*, New Generation Computing **20(1)** (2001), pp. 3–26.

[2] Alpuente, M., M. Falaschi and G. Vidal, *Partial Evaluation of Functional Logic Programs*, ACM Transactions on Programming Languages and Systems **20** (1998), pp. 768–844.

[3] Alpuente, M., S. Lucas, M. Hanus and G. Vidal, *Specialization of functional logic programs based on needed narrowing.*, Theory and Practice of Logic Programming **5** (2005), pp. 273–303.

[4] Antoy, S., *Definitional trees*, in: *Proc. of the 3rd International Conference on Algebraic and Logic Programming ALP'92*, Lecture Notes in Computer Science **632** (1992), pp. 143–157.

[5] Antoy, S., R. Echahed and M. Hanus, *Parallel evaluation strategies for functional logic languages*, in: *Proc. of the Fourteenth International Conference on Logic Programming (ICLP'97)* (1997), pp. 138–152.

[6] Antoy, S., R. Echahed and M. Hanus, *A needed narrowing strategy*, Journal of the ACM **47(4)** (2000), pp. 776–822.

[7] Bouajjani, A. and R. Mayr, *Model checking lossy vector addition systems*, in: C. Meinel and S. Tison, editors, 16^{th} *Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science **1563**, 1999, pp. 323–333.

[8] Bouhoula, A. and M. Rusinowitch, *Implicit induction in conditional theories*, Journal of Automated Reasoning **14** (1995), pp. 189–235.

[9] Burkart, O., D. Caucal, F. Moller and B. Steffen, *Verification over Infinite States*, in: J. Bergstra, A.Ponse and S. Smolka, editors, *Handbook of Process Algebra*, Elsevier Publishing, 2001 pp. 545–623.

[10] Clarke, E. M., O. Grumberg and D. E. Long, *Model checking and abstraction*, ACM Transactions on Programming Languages and Systems **16** (1994), pp. 1512–1542.

[11] Clavel, M., F. Durán, S. Eker and J. Meseguer, *Building equational proving tools by reflection in rewriting logic*, in: K. Futatsugi, A. T. Nakagawa and T. Tamai, editors, *Cafe: An Industrial-Strength Algebraic Formal Method*, Elsevier, 2000 pp. 1–31, http://maude.csl.sri.com/papers.

[12] Comon, H. and R. Nieuwenhuis, *Induction = I-Axiomatization + First-Order Consistency*, Information and Computation **159** (2000), pp. 151–186.

[13] De Schreye, D., R. Glück, J. Jørgensen, M. Leuschel, B. Martens and M. Sørensen, *Conjunctive Partial Deduction: Foundations, Control, Algorithms, and Experiments*, Journal of Logic Programming **41** (1999), pp. 231–277.

[14] Denker, G., J. Meseguer and C. L. Talcott, *Protocol specification and analysis in Maude*, in: N. Heintze and J. Wing, editors, *Proceedings of Workshop on Formal Methods and Security Protocols, June 25, 1998, Indianapolis, Indiana*, 1998, http://www.cs.bell-labs.com/who/nch/fmsp/index.html.

[15] Dershowitz, N., S. Mitra and G. Sivakumar, *Decidable matching for convergent systems (preliminary version).*, in: D. Kapur, editor, *11th International Conference on Automated Deduction (CADE-11)*, Lecture Notes in Computer Science **607** (1992), pp. 589–602.

[16] Dolev, D. and A. Yao, *On the security of public key protocols*, IEEE Transaction on Information Theory **29** (1983), pp. 198–208.

[17] Emerson, A. and K. Namjoshi, *On model checking for nondeterministic infinite state systems*, in: 13^{th} *IEEE Symposium on Logic in Computer Science*, 1998, pp. 70–80.

[18] Escobar, S., *Refining weakly outermost-needed rewriting and narrowing*, in: D. Miller, editor, *Proc. of 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'03* (2003), pp. 113–123.

[19] Escobar, S., *Implementing natural rewriting and narrowing efficiently*, in: Y. Kameyama and P. J. Stuckey, editors, *7th International Symposium on Functional and Logic Programming (FLOPS 2004)*, Lecture Notes in Computer Science **2998** (2004), pp. 147–162.

[20] Escobar, S., C. Meadows and J. Meseguer, *A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties*, Theoretical Computer Science **367(1-2)**, pp. 162–202.

[21] Escobar, S., J. Meseguer and P. Thati, *Natural narrowing for general term rewriting systems*, in: J. Giesl, editor, 16^{th} *International Conference on Rewriting Techniques and Applications*, Lecture notes in computer science **3467** (2005), pp. 279–293.

[22] Fabrega, F. J. T., J. C. Herzog and J. Guttman, *Strand spaces: Proving security protocols correct*, Journal of Computer Security **7** (1999), pp. 191–230.

[23] Fay, M., *First order unification in equational theories*, in: W. Bibel and R. Kowalski, editors, 4^{th} *Conference on Automated Deduction*, Lecture Notes in Computer Science **87** (1979), pp. 161–167.

[24] Finkel, A. and P. Schnoebelen, *Well-structured transition systems everywhere!*, Theoretical Computer Science **256** (2001), pp. 63–92.

[25] Genet, T. and V. V. T. Tong, *Reachability analysis of term rewriting systems with Timbuk*, in: 8^{th} *International Conference on Logic for Programming*, Lecture Notes in Computer Science **2250**, 2001.

[26] Goguen, J. and J. Meseguer, *Equality, types, modules and (why not?) generics for logic programming*, Journal of Logic Programming **1** (1984), pp. 179–210.

[27] Goguen, J. and J. Meseguer, *Models and Equality for Logical Programming*, in: G. L. H. Ehrig, R. Kowalski and U. Montanari, editors, *Proc. of TAPSOFT'87*, Lecture Notes in Computer Science **250** (1987), pp. 1–22.

[28] Graf, S. and H. Saidi, *Construction of abstract state graphs with PVS*, in: O. Grumberg, editor, *Computer Aided Verification. 9th International Conference, CAV'97, Haifa, Israel, June 22-25, 1997, Proceedings*, Lecture Notes in Computer Science **1254**, 1997, pp. 72–83.

[29] Hanus, M., *The integration of functions into logic programming: From theory to practice*, Journal of Logic Programming **19&20** (1994), pp. 583–628.

[30] Hanus, M., S. Antoy, H. Kuchen, F. López-Fraguas, W. Lux, J. Moreno Navarro and F. Steiner, *Curry: An Integrated Functional Logic Language (version 0.8)*, Available at: http://www.informatik.uni-kiel.de/~curry (2003).

[31] Hudak, P., P. L. Wadeler, Arvind, B. Boutel, J. Fairbairn, J. Fasel, K. Hammond, J. Hughes, T. Johnsson, R. Kieburtz, R. S. Nikhil, S. L. Peyton-Jones, M. Reeve, D. Wise and J. Young, *Report on the functional programming language haskell*, Technical report, Department of Computer Science, Glasgow University (1990).

[32] Huet, G. and J.-J. Lévy, *Computations in Orthogonal Term Rewriting Systems, Part I + II*, in: *Computational logic: Essays in honour of J. Alan Robinson* (1992), pp. 395–414 and 415–443.

[33] Hullot, J., *Canonical forms and unification*, in: W. Bibel and R. Kowalski, editors, 5^{th} *Conference on Automated Deduction*, Lecture Notes in Computer Science **87** (1980), pp. 318–334.

[34] Jones, N., C. Gomard and P. Sestoft, "Partial Evaluation and Automatic Program Generation," Prentice-Hall, Englewood Cliffs, NJ, 1993.

[35] Jouannaud, J.-P., C. Kirchner and H. Kirchner, *Incremental construction of unification algorithms in equational theories*, in: 10^{th} *International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **154** (1983), pp. 361–373.

[36] Kesten, Y. and A. Pnueli, *Control and data abstraction: The cornerstones of practical formal verification*, International Journal on Software Tools for Technology Transfer **4** (2000), pp. 328–342.

[37] Lafave, L. and J. Gallagher, *Constraint-based Partial Evaluation of Rewriting-based Functional Logic Programs*, in: *Proc. of 7th International Workshop on Logic-based Program Synthesis and Transformation, LOPSTR'97*, Lecture Notes in Computer Science **1463** (1997), pp. 168–188.

[38] Loiseaux, C., S. Graf, J. Sifakis, A. Bouajjani and S. Bensalem, *Property preserving abstractions for the verification of concurrent systems*, Formal Methods in System Design **6** (1995), pp. 1–36.

[39] Lowe, G., *Breaking and fixing the Needham-Schroeder public-key protocol using fdr*, in: *Tools and algorithms for construction and analysis of systems (TACAS '96)*, Lecture Notes in Computer Science **1055** (1996), pp. 147–166.

[40] Martí-Oliet, N. and J. Meseguer, *Rewriting logic as a logical and semantic framework*, in: D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd. Edition*, Kluwer Academic Publishers, 2002 pp. 1–87, first published as SRI Tech. Report SRI-CSL-93-05, August 1993.

[41] Meadows, C., *The NRL Protocol Analyzer: An overview*, The Journal of Logic Programming **26** (1996), pp. 113–131.

[42] Meseguer, J., *General logics*, in: H.-D. E. et al., editor, *Logic Colloquium'87* (1989), pp. 275–329.

[43] Meseguer, J., *Conditional rewriting logic as a unified model of concurrency*, Theoretical Computer Science **96** (1992), pp. 73–155.

[44] Meseguer, J., *Multiparadigm logic programming*, in: H. Kirchner and G. Levi, editors, *Proc. 3rd Intl. Conf. on Algebraic and Logic Programming* (1992), pp. 158–200.

[45] Meseguer, J., *Membership algebra as a logical framework for equational specification*, in: F. Parisi-Presicce, editor, *Proc. WADT'97* (1998), pp. 18–61.

[46] Meseguer, J. and C. Talcott, *Semantic models for distributed object reflection*, in: *Proceedings of ECOOP'02, Málaga, Spain, June 2002* (2002).

[47] Meseguer, J. and P. Thati, *Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols*, High-Order Symbolic Computation (2006), to appear.

[48] Needham, R. and M. Schroeder, *Using encryption for authentication in large networks of computers*, Communications of the ACM **21** (1978), pp. 993–999.

[49] Nieuwenhuis, R., *Basic paramodulation and decidable theories*, in: *Proceedings of the Eleventh Annual IEEE Symposium On Logic In Computer Science (LICS'96)* (1996), pp. 473–483.

[50] Ohsaki, H., H. Seki and T. Takai, *Recognizing boolean closed A-tree languages with membership conditional mechanism*, in: 14^{th} *International Conference on Rewriting Techniques and Applications*, Lecture notes in computer science **2706** (2003), pp. 483–498.

[51] Owre, S., N. Shankar, J. Rushby and D. Stringer-Calvert, "PVS system guide, PVS language reference, and PVS prover guide version 2.4," Computer Science Laboratory, SRI International (2001).

[52] Paulson, L., "Isabelle: A Generic Theorem Prover," Lecture Notes in Computer Science **828**, Springer Verlag, 1994.

[53] Reddy, U., *Term rewriting induction*, in: M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, Lecture Notes in Computer Science **449** (1990), pp. 162–177.

[54] Saïdi, H. and N. Shankar, *Abstract and model check while you prove*, in: N. Halbwachs and D. Peled, editors, *Computer Aided Verification. 11th International Conference, CAV'99, Trento, Italy, July 6-10, 1999, Proceedings*, Lecture Notes in Computer Science **1633**, 1999, pp. 443–454.

[55] Sekar, R. and I. Ramakrishnan, *Programming in equational logic: Beyond strong sequentiality*, Information and Computation **104** (1993), pp. 78–109.

[56] Sørensen, M. H., R. Glück and N. D. Jones, *A positive supercompiler*, Journal of Functional Programming **6** (1996), pp. 811–838.

[57] TeReSe, editor, "Term Rewriting Systems," Cambridge University Press, Cambridge, 2003.

[58] Thati, P. and J. Meseguer, *Complete symbolic reachability analysis using back-and-forth narrowing*, Theoretical Computer Science (2006), to appear.

[59] Turchin, V. F., *Program Transformation by Supercompilation*, in: H. Ganzinger and N.D. Jones, editors, *Proceedings of Programs as Data Objects*, Lecture Notes in Computer Science **217** (1986), pp. 257–281.

[60] Viry, P., *Equational rules for rewriting logic*, Theoretical Computer Science **285** (2002), pp. 487–517.