REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 14, número 2
Págs. 98 - 121

Automatic Proof-Search Heuristics in the Maude Invariant Analyzer Tool

Camilo Rochaz[*][‡]

Fecha de Recibido: 09/10/2013 Fecha de Aprobación: 09/11/2013

Abstract

The Invariant Analyzer Tool is an interactive tool that mechanizes an inference system for proving safety properties of concurrent systems, which may be infinite-state or whose set of initial states may be infinite. This paper presents the automatic proof-search heuristics at the core of the Maude Invariant Analyzer Tool, which provide a substantial degree of automation and can automatically discharge many proof obligations without user intervention.

These heuristics can take advantage of equationally defined equality predicates and include rewriting, narrowing, and SMT-based proof-search techniques.

Keywords: *software engineering, proof-search heuristics, rewriting logic, Maude Invariant Analyzer, rewriting logic, satisfiability modulo theories.*

*Assistant Professor at Escuela Colombiana de Ingeniería Julio Garavito, AK 45 No. 205-59, Bogotá, D.C., Colombia. Correo electrónico: camilo.rocha@escuelaing.edu.co.
‡ Se concede autorización para copiar gratuitamente parte o todo el material publicado en la Revista Colombiana de Computación siempre y cuando las copias no sean usadas para fines comerciales, y que se especifique que la copia se realiza con el consentimiento de la Revista Colombiana de Computación.

1. Introduction

Safety properties of concurrent systems are among the most important properties to verify. They have received extensive attention in many different formal approaches, both algorithmic and deductive. Algorithmic approaches such as model checking are quite attractive because they are automatic. However, they cannot always be applied as a system can be infinite-state, so that no model checking algorithm which assumes a finite-state system can directly be used. Even if an abstraction can be found to make the system finite-state, an additional difficulty may arise: although for each initial state the set of states reachable from it is finite, the set of initial states may still be infinite, so that model checking verification may not be possible. For example, a mutual exclusion protocol should be verified for an arbitrary number of clients in its initial state, even if the states have been abstracted away so that the set of states reachable from each initial state is always finite.

This paper presents the automatic proof-search heuristics at the core of the Maude Invariant Analyzer Tool (InvA). The InvA tool mechanizes the inference system in [13,14] for proving safety properties of concurrent systems, which may be infinitestate or whose set of initial states may be infinite. The mechanization of the above inference system in the InvA tool provides a substantial degree of automation and can automatically discharge many proof obligations without user intervention. The development of the InvA tool is part of a broader effort in the Maude Formal Environment [6] to develop generic automatic and semi-automatic tool-support for different reasoning methods. The expression "generic" means that the verification methods and their associated tools are not tied to a specific programming language. The advantage of generic verification methods and tools is that the costly tool development effort can be amortized across a much wider range of applications, whereas a language-specific verification tool can only be applied to systems programmed in that specific language.

Any such generic approach requires a logical framework general enough to encompass many different models and languages. In this case, the use of the rewriting logic framework [9] is justified by its ability to express very naturally many different models of concurrent computation and many concurrent languages. It also has good properties as a general semantic framework for giving executable semantics to a wide range of languages and models of concurrency. In particular, it supports very well concurrent object-oriented computation. The same reasons making rewriting logic a good semantic framework make it also a good logical framework, that is, a metalogic in which many other logics can be naturally represented and executed. Furthermore, rewriting logic is a

reflective logic so that important aspects of its metatheory can be represented at the object level in a consistent way, so that the object-level representation correctly simulates the relevant metatheoretic aspects. The Maude [3] system is an implementation of rewriting logic, with efficient support for rewriting, both at the object level and at the metalevel, and narrowing modulo axioms.

In the rewriting logic framework, a concurrent system, such as, for example, a network protocol or an entire concurrent programming language such as Java, is specified as a *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$, with (Σ, E) an equational theory specifying the system's states as elements of the initial algebra $T_{\Sigma/E}$ and R a collection of (non-equational) rewrite rules specifying the system's *concurrent transitions*. Safety properties are a special type of *inductive* properties. That is, they do not hold for just any model of the given rewrite theory $\mathcal{R}$, but for its *initial reachability model* $\mathcal{T}_{\mathcal{R}}$. Concretely, for $\mathcal{R} = (\Sigma, E, R)$, this means that the states of such an initial model are precisely elements of the initial algebra $T_{\Sigma/E}$, and that its one-step transitions are *provable* rewrite steps between such states by means of the rules R. Therefore, given any safety property φ, the interest is on checking the model-theoretic satisfaction relation $\mathcal{T}_{\mathcal{R}} \models \varphi$, which is approximated deductively by means of the inductive inference relation $\mathcal{R} \Vdash$ mechanized in the InvA tool.

The inference system mechanized in the InvA tool is transformational in the sense that the rules of inference transform pairs of the form $\mathcal{R} \Vdash$ into other such pairs $\mathcal{R}' \Vdash \varphi'$. It is also *reductionistic* in the sense that: (i) all safety formulas in temporal logic eventually disappear and are replaced by *purely equational formulas* and (ii) the rewrite *theory* $\mathcal{R} = (\Sigma, E, R)$ is eventually replaced by its underlying *equational theory* (Σ, E). That is, in the end *all formal reasoning about safety properties is reduced to inductive reasoning about equational properties* in the underlying equational theory (Σ, E). This allows for these generic safety verification methods to take advantage of the existing wealth of equational reasoning techniques and tools already available.

The Maude Invariant Analyzer Tool supporting the transformational and reductionistic inference, at the level of deduction and heuristics for discharging proof obligations makes systematic use of:

- *Equatplification with* the equations defining both system states and state predicates to reduce proof obligations to simpler forms;

- *Boolean equality enrichments* [7] and its combination by means of Boolean operations, giving more teeth to the other proof-search heuristics because firstorder equality is made available to the object level;

- *Narrowing modulo axioms* with the equations defining state predicates to greatly simplify the equational proof obligations to which all proofs of safety formulas are ultimately reduced; and

- *Satisfiability modulo theories solving* (i.e., SMT-solving) with built-in predicates over the integers to automatically check for proof obligations that are tautologiional simes (or are unsatisfiable) when these or their subformulae correspond to integer linear arithmetic constraints.

The InvA tool, together with documentation and more examples, is available at http://camilorocha.info/software. The exposition on the automatic proofsearch heuristics currently available from the InvA tool presented in this paper is based on unpublished work in [13, Sec. 4.4].

2. Preliminaries

Notation and terminology from [10] for order-sorted equational logic and from [1] for rewriting logic is followed. An *order sorted signature* _ is a tuple å = (S; ≤F) with finite poset of sorts (S; ≤) and a finite set of function symbols *F*. It is assumed that: (i) each connected component of a sort s ˛ S in the poset ordering has a top sort, denoted by *ks*, and (ii) for each operator declaration f ˛ $F_{sl...sn,s}$ there is also a declaration f ˛ $F_{Ks1...ksn;\, ks}$. The collection $X = \{X_s\}_{s ˛ S}$ is an S-sorted family of disjoint sets of variables with each X_s countably infinite. The set of terms of sort s is denoted by $T_å(X)_s$ and the set of ground terms of sort s is denoted by $T_{å,s}$, which are assumed nonempty for each S. The expressions $T_å(X)$ and $T_å$ denote the respective term algebras. The set of variables of a term t is written *vars*(t) and is extended to sets of terms in the natural way. A *substitution* q is a sorted map from a finite subset dom(q) ˝ X to $T_å(X)$ and extends homomorphically in the natural way; ran(q) denotes the set of variables introduced by q and tq the application of q to a term t. Substitution q_1q_2 is the composition of substitutions q_1 and q_2 . Asubstitution q is called *ground* if *ran*(q) = ˘ .

A _-equation is a Horn clause t = u if g, where t = u is a S-*equality* with *t, u* ˛ T_S *(X)*$_s$ for some sort *s* ˛ *S*, and the *condition* g is a finite conjunction of S-equalities L$_{i ˛ I}$ $t_i = u_i$. An *equational theory* is a tuple (S;E) with order-sorted signature S and finite set of S-equations E. For ' a S-equation, (S;E) |-j iff j can be proved from (S;E) by the deduction rules in [10] iff j is valid in all models of (S;E); (S;E) induces the congruence relation $=_E$ on $T_S(X)$ defined for any *t; u* ˛ $T_S(X)$ *by* $t =_E u$ iff (S, E) |- *t* = *u*. The expressions T $T_{S/E}(X)$ and ?$T_{S/E}$ denote the quotient algebras induced by $=_E$ over the algebras *T (X)* and T_S, respectively; $T_{S/E}$ is the *initial algebra* of (S;*E*). An *E-unifier* for a S-equality *t* = *u* is a substitution q such that tq =E uq. A complete set of *E*-unifiers for a S-equality t = u is written CSU_E(t = u) and it is called *finitary* if it contains a finite number of *E-unifiers*. The expression GU_E(t = u) denotes the set of ground *E-unifiers* of a S-equality t = u. A theory inclusion (S;E) ˝ (S´;E´) is *protecting* iff the unique S-homomorphism $T_{S/E}$ fi $T_{S´/E´}$ ‰S to the S-reduct of the initial algebra $T_{S´/E´}$ is an isomorphism.

A S-*rule* is a sentence *t* fi *u* if g, where *t* fi *u* is a S-*sequent* with *t, u* ˛ $T_S(X)_s$ for some sort *s* ˛ *S* and the *condition* g is a finite conjunction of S-equalities. A *rewrite theory* is a tuple *R* = (S;E;R) with equational theory e*R* = (S;E) and a finite set of S-rules *R*. A *topmost rewrite theory* is a rewrite theory R = (S;E;R) such that for some top sort S = [S] and for each *t* fi u if g ˛ R, the terms t, u satisfy t, u ˛ $T_S(X)_s$ and t ˇ X, and no operator in S has S as argument sort. For *R* = (S;E;R) and ' a S-rule, R |- ' iff ' can be obtained from R by the deduction rules in [1] iff ' is valid in all models of R . For ' a S-equation, *R* |- iff eR |- . A rewrite theory R = (S;E;R) induces the rewrite relation fiR on $T_{S/E}$(X) defined for every *t, u* ˛ T_S(X) by $[t]_E$ fiR $[u]_E$ iff there is a *one-step* rewrite proof *R* |- t fi *u*. The expressions R |- t fi *u* and R |- t fi u respectively denote a *one-step* rewrite proof and an arbitrary length (but finite) rewrite proof in R from *t* to *u*. The expression T_R = ($T_{S/E}$, fi$^*_R$) denotes the *initial reachability* model of *R* = (S;E;R) [1]. A S-sequent j is an *inductive consequence* of *R* iff *R* |- j iff (" q: X fi T_S) *R* |- j q iff T_R |- j .

State predicates. A set of *state predicates II* for *R* = (S;*E;R*) can be equationally-defined by an equational theory e$_{II}$=(S$_{II}$;E], *E*$_{II}$). Signature S$_{II}$ contains S, two sorts Bool £ ^ [*Bool*] with constants ⊤ and ⊥ of sort

Bool , predicate symbols p : S fi [*Bool*]for each p ¸ *II*, and optionally some auxiliary function symbols.

Equations in E_{II} define the predicate symbols in S_{II} and auxiliary function symbols, if any; they protect (S;E) and the equational theory specifying sort *Bool* , constants ⊤ and ⊥, and the Boolean operations. It is easy to define a state predicate p ¸ *II* as a Boolean combination of other already-defined state predicates {*p1, . . . pn*} in S_{II}. The reason why p: S fi [*Bool*] instead of p : S fi *Bool* , is to allow partial definitions of p with equations that only define *the positive* case by equations $p(t)$ = ⊤ *if* j , and and either leave the *negative* case implicit or may only define some negative cases with equations $p(t')$ = ⊥ if g´ without necessarily covering all the cases.

LTL semantics. For p ¸ *II* and $[t]_E$ ¸ $T_{S/E;S}$, e_{II} defines the semantics of p in T_R as follows: it is said that $p([t]_E)$ holds in T_R iff e_{II} ⊢ p(t) = ⊤. This defines a Kripke structure K_R^{II} = ($T_{S/E;S}$; fi R; L_{II}) with labeling function L_{II} such that, for each $[t]_E$ ¸ $T_{S/E;S}$ the semantic equivalence p ¸ $L_{II}([t]_E)$ iff p $([t]_E)$ holds in T_R. Then, all of LTL can be interpreted in K_R^{II} R in the standard way [2], including the "always" ( ), "next" (O), and "strong implication" (Þ) operators.

Executability conditions. It is assumed that the set of equations of a rewrite theory R can be decomposed into a disjoint union E] A, with A a collection of axioms (such as associativity, and/or commutativity, and/or identity) for which there exists a *matching algorithm modulo A* producing a finite number of A-matching substitutions, or failing otherwise. It is also assumed that the equations E can be oriented into a set of *ground sort-decreasing, ground confluent, and ground terminating* rules $\vec{E}$ modulo A. The expression t$\downarrow_{S,E/A}$ ¸ $T_{S,s}(X)$ denotes the E/A-*canonical form* of t ¸ T_S(X). The rules R in R are assumed to be *ground coherent* relative to the equations E modulo A [17].

Free constructors. For R =(S;E ∔ A;R), the signature W ˝ S is a signature of *free constructors* modulo A iff for each sort s in S and t ¸ $T_{S,s}$ there is u ¸ T_{Ws} satisfying $t = E_{∔A}$ u, and $u\downarrow_{S,E/A} =_A u$ for any u ¸ $T_{W,s}$. For the development in this paper it is required that t ¸ $T_W(X)$ for each t fi u if g ¸ R.

3. The Maude Invariant Analyzer Tool: An Overview

The Maude Invariant Analyzer Tool (InvA) is a tool designed for *interactively* proving two key safety properties of executable Maude specifications, namely, inductive stability and inductive invariance, plus their combination by strengthening techniques. The tool mechanizes an inference system that, without assuming finiteness of the set of initial or reachable states, uses rewriting and narrowing-based reasoning techniques, in which all temporal logic formulas eventually disappear and are replaced bypurely equational conditional sentences. The InvA tool provides a substantial degree of mechanization and can automatically discharge many proof obligations without user intervention. It is implemented in the Maude language and exploits rewriting logic's reflection capabilities, i.e., it is a Maude specification that takes, as part of its input, a meta-representation of a Maude specification.

The concept of inductive stability for R = (S, E, R) is intimately related with the notion of the set of states $t \in T_{S,s}$ of T_R that satisfy a state predicate $p \in II$ being closed under fi R. More precisely, for $p \in II$ and $x \in X$, the property p being *inductively* stable for R is the safety property:

$$K_{R}^{II} \models p(x) \,)\quad p(x) \qquad (1)$$

meaning that if $I(t)$ holds in a state $t \in T_{S,s}$, then $p(u)$ holds in any state $u \in T_{S,s}$, that is reachable from t.

Invariants are among the most important safety properties. Given a set of initial states characterized by $I \in II$, a state predicate $p \in II$ being *inductively invariant* for R from the set of initial states I is the safety property

$$K_{R}^{II} \models I(x) \,)\quad p(x) \qquad (2)$$

meaning that if I(t) holds in a state $t \in T_{S;s}$, then p(u) holds in any state u $\in T_{S;s}$ reachable from t. In other words, the invariant p holds for all states reachable from I. Since the set of initial states is defined in e_{II} as a state

predicate $I \subseteq \Pi$, an equational definition of I can of course capture an infinite set of initial states.

3.1. Inference System Mechanized in the InvA Tool

Given a ground stability or ground invariance property φ, the InvA tool generates equational proof obligations such that, if they hold, then $\mathcal{T}_{\mathcal{R}} \models \varphi$. For a topmost rewrite theory $\mathcal{R}$ and of a set of state predicates Π specified in Maude, the InvA tool mechanizes inference rules ST, INV, STR1, STR2, C⇒, NR1, and NR2 depicted in Figure 1. Soundness proofs for each one of these inference rules can be found in [13]. The application of inference rules ST, INV, STR1, and STR2 to a given inductive stability or invariance LTL verification goal ultimately reduces such a goal to simpler inductive equational reasoning that can be handled by applying rules C⇒, NR1, and NR2.

Inference rule ST reduces the verification task of *p*-stability to the simpler condition $p \Rightarrow \bigcirc p$, which only involves 1-step search instead of arbitrary depth search. Inference rule INV reduces the verification task of inductive invariance to equational implication and inductive stability. Inference rules STR1 and STR2 are strengthening rules. Inference rule C⇒ handles equational implications, while rules NR1 and NR2 use 1-step narrowing modulo axioms to handle the symbolic 1-step search, forthe temporal next operator, in formulae of the form $p \Rightarrow \bigcirc p$. Note that any inductive stability and invariance formula is ultimately reduced to equational reasoning. Thanks to the availability in Maude 2.6 of unification modulo commutativity (C), associativity and commutativity (AC), and modulo these theories plus identities (U), and to the narrowing modulo infrastructure, the InvA tool can handle modules with operators declared C, CU, AC, and ACU. Furthermore, since unification modulo the above theory combinations is decidable, and each one yields a finite set of complete unifiers, the set of proof obligations is always finite.

3.2. Methodology and Commands Available to the User

The approach for proving inductive stability and invariance properties in the InvA tool is depicted in Figure 2.

Given a topmost rewrite theory $\mathcal{R}$, an equational specification e Π for the state predicates Π, and an inductive safety property φ the InvA tool internally generates equational proof obligations according to the

inference system in Figure 1 and tries to discharge as many of them as possible by using the heuristics described in Section 4. Any proof proof obligation that cannot be automatically discharged is output to the user so it can be handled interactively in an external tool such as Maude's Inductive Theorem Prover (ITP) [4, 8] (an experimental interactive tool for proving properties of the initial algebra $T_{\mathcal{E}}$ of an order-sorted equational theory $\mathcal{E}$ written in Maude).

$$\frac{\mathcal{R} \Vdash p(x) \Rightarrow \bigcirc p(x)}{\mathcal{R} \Vdash p(x) \Rightarrow \Box p(x)} \text{ ST}$$

$$\frac{\mathcal{R} \Vdash I(x) \Rightarrow p(x) \quad \mathcal{R} \Vdash p(x) \Rightarrow \Box p(x)}{\mathcal{R} \Vdash I(x) \Rightarrow \Box p(x)} \text{ INV}$$

$$\frac{\mathcal{R} \Vdash I(x) \Rightarrow J(x) \quad \mathcal{R} \Vdash J(x) \Rightarrow \Box q(x) \quad \mathcal{R} \Vdash q(x) \Rightarrow p(x)}{\mathcal{R} \Vdash I \Rightarrow \Box p} \text{ STR1}$$

$$\frac{\mathcal{R} \Vdash I(x) \Rightarrow p(x) \quad \mathcal{R} \Vdash I(x) \Rightarrow \Box q(x) \quad \mathcal{R} \Vdash q(x) \wedge p(x) \Rightarrow \bigcirc p(x)}{\mathcal{R} \quad I(x) \Rightarrow \Box p(x)} \text{ STR2}$$

$$\frac{\varepsilon_{II} \Vdash \bigwedge_{(q(v) = w \text{ if } \gamma') \in E_{II}} p(v) = \mathsf{T} \text{ if } \gamma' \wedge w = T}{\mathcal{R} \Vdash q(x) \Rightarrow p(x)} \text{ C}\Rightarrow$$

$$\frac{\bigwedge_{\substack{(l \to r \text{ if } \gamma) \in \mathcal{R} \\ (\theta, w, \gamma') \in \theta^p_l}} \varepsilon II \Vdash p(r\theta) = \mathsf{T} \text{ if } \gamma\theta \wedge \gamma'\theta \quad w\theta = \mathsf{T}}{\mathcal{R} \Vdash q(x) \Rightarrow \bigcirc p(x)} \text{ C}\Rightarrow$$

$$\frac{\bigwedge_{\substack{(l \to r \text{ if } \gamma) \in \mathcal{R} \\ (\theta, w, \gamma') \in \theta^p_l}} \varepsilon II \Vdash p(r\theta) = \mathsf{T} \text{ if } \gamma\theta \wedge \gamma'\theta \wedge w\theta = \mathsf{T} \wedge q(l)\theta = \mathsf{T}}{\mathcal{R} \Vdash q(x) \Rightarrow p(x)}$$

where $\theta^p_l = \cup_{(p(v) = w \text{ if } \gamma') \in E_{II}} \{(\theta, w, \gamma') \mid \theta \in CSUA(l=v)\}$

Fig. 1. Inference rules mechanized in the InvA tool.

The user interacts with the InvA tool via the following commands:

- (help .) shows the list of commands available.
- (analyze-stable <pred> in <module> <module>.) generates the proof obligations for inference ST with inference NR1, for the given predicate. The first module equationally specifies the state predicate and the second one the topmost rewrite theory. This command tries to eagerly discharge the proof obligations; those that cannot be discharged are shown to the user.
- (analyze-stable <pred> in <module> <module> assuming <pred>.) generates the proof obligations for proving the third premise of inference STR2 with inference NR2, for the given predicate and the given modules. The first module equationally specifies the state predicates and the second one the topmost rewrite theory. This command tries to eagerly discharge the proof obligations; those that cannot be discharged are shown to the user.
- (analyze <pred> implies <pred> in <module>.) generates the proof obligations for proving the given implication in the given module, according to inference *C* Þ . This command tries to eagerly discharge the proof obligations; those that cannot be discharged are shown to the user.
- (show pos.) shows the proof obligations computed by the last analyze command that could not be discharged; those that are discharged are not shown.
- (show-all pos .) shows all the proof obligations computed by the last analyze command.

Observe that the analysis commands in InvA give direct tool support for deductive reasoning with *some* of the inference rules presented in this chapter, but not for all of them. For example, there is no command in InvA directly supporting deduction with inference rule INV. Nevertheless, deduction with *all* inference rules in this chapter is supported by InvA via *combination of commands*. For example, deduction with inference rule INV can be achieved by combining the analyze and analyze-stable commands.

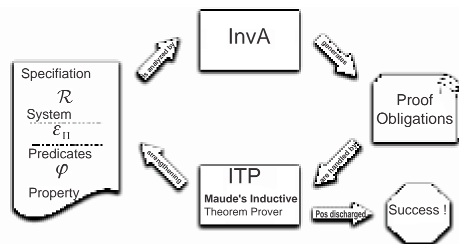

Fig. 2. Approach for checking inductive stability and invariance properties for rewrite theories.

4. Proof-Search Heuristics in InvA

The InvA tool has been successfully used in the formal verification of safety properties for concurrent systems in different domains. The InvA case studies include the formal verification of safety properties for the Illinois Browser Operating System (IBOS) [16], a modern, security-conscious web browser designed at the University of Illinois which could be integrated into a secure operating system. For this case study, the largest one considered so far, the InvA tool generated approximately 22755 proof obligations and, thanks to its proof-search heuristics, it was able to automatically discharge all but 48 proof obligations, a success rate of about 99:8% (see [13, Ch. 6] for details). This section presents a description of the proof-search heuristics used in the InvA tool and summarizes how comparatively useful in practice they are.

After applying rules ST, INV, STR1, STR2, CÞ , NR1, and NR2 according to the user commands, the InvA tool uses rewriting-based reasoning and narrowing procedures, and SMT decision procedures for automatically discharging as many of the generated equational proof obligations as possible. For an executable equational specification e_{II} = (S_{II}, E_{II} ¨ *A)* and a conditional proof obligation j of the form

$t = u$ if g;

the InvA tool applies a proof-search strategy such that, if it succeeds, then the Kripke structure K^{II}_{R} associated to the initial reachability model TR satisfies j . Otherwise, if the proof-search fails, the proof obligation j (or a logically equivalent variant) is output to the user.

4.1. Boolean Transformations

For the proof-search process, the InvA tool first tries to simplify Boolean expressions in j . During the simplification process, the tool assumes that any operator '~' is an equationally defined equality predicate, i.e., an *equality enrichment*. Given an order-sorted signature S = (S, £, F) and an order-sorted equational theory e=(S,E) with initial algebra Te , an equality enrichment [11] of e is an equational theory e˜ that extends e by defining a Boolean-valued equality function symbol '~' that coincides with '=' in Te .

Definition 1 An *equational theory* e˜ = (S˜,E˜) *is called an equality* enrichment of e = (S,*E*), *with* S˜ = (S˜,£˜, F˜) and S = (S,£,F), *iff*

- E~ *is a protecting extension of* E;
- *the poset of sorts of* S~ *extends* (*S*;£) *by adding a new sort Bool that belongs to a new connected component, with constants* T *and ? such that* TE~;*Bool* = f[>];[?]g, *with* >6E~?; *and*
- *for each connected component in* (S, £) *there is a top sort* k ¿ *S*~ *and a binary commutative operator* _~_ : k k fi *Bool in* S~ *such that the following equivalences hold for any ground terms t, u* ¿ $T_{S,k}$:

$$\mathcal{E} \vdash t = u \quad \Leftrightarrow \quad \tilde{\mathcal{E}} \vdash (t \sim u) = \top, \quad (3)$$

$$\mathcal{E} \not\vdash t = u \quad \Leftrightarrow \quad \tilde{\mathcal{E}} \vdash (t \sim u) = \bot. \quad (4)$$

An equality enrichment ? ~of ? is called Boolean *iff it contains all the function symbols and equations making the elements of* $T_{?~;Bool}$ *a two-element Boolean algebra.*

Using the information about `?', a Boolean transformation can be applied recursively to ' with the additional information of the equality enrichment, if any is defined.

The goal of the Boolean transformation process is to obtain, if possible, an inductively equivalent proof obligation ' ´ for which the automatic search tests, explainedbelow, have better chances of success. The following is a description of the Boolean mtransformations applied recursively by the InvA tool:

- If $t = u$ in j is such that t is of the form $t_1 \sim t_2$ and u of the form $\bot$, then j is transformed into $\top = \bot$ if Ù $t_1 = t_2$.
- If $v_1 = v_2$, with v_1, v_2 ¿ $T_S(X)_{Bool}$, is any of the S-equalities in the condition g of j , then:
  - If v_1 is of the form $v_1^1 \sim v_1^2$ and v_2 of the form ¯, then $v_1 = v_2$, is replaced by $v_1^1 = v_1^2$.
  - If v_1 is of the form v_1^1 u⊓...⊓$v^n{}_1$ and v_2 of the form $\top$, then $v_1 = v_2$ is replaced by $v_1^1 = \top \wedge ... \wedge v^n{}_1 = \top$. Note that the v_1^i have sort Bool .
  - f v_1 is of the form v_1^1 u⊓...⊓$v^n{}_1$ and v_2 of the form $\bot$, then $v_1 = v_2$ is replaced by $v_1^1 = \bot \wedge ... \wedge v^n{}_1 = \bot$. Note that the v_1^i have sort Bool .

Symbols ⊓ and ⊔ are used to represent, respectively, the conjunction and disjunction function symbols used by the Boolean equality enrichment in

Definition 1. Also note that _S-equalities are unoriented, and thus in the Boolean transformation the order of terms in the equalities is immaterial.

4.2. Automatic Proof-Search Tests

After the Boolean transformation process is completed, some automatic search tests are applied to the resulting proof obligation following the strategy described below. In what follows, it is assumed that ' has been already simplified by the transformations described in Section 4.1.. Furthermore, let $\bar{t}, \bar{u}, \bar{g}$ be obtained by replacing each variable x ¸ X by a new constant x ¸ $\bar{X}$, with S ˙ $\bar{X}$ = ˘ .

4.2.1. Equational simplification.

The strategy checks if ' holds *trivially*, i.e., if

$$t\downarrow \Sigma, E/A =_A u\downarrow \Sigma, E/A$$

or there is $t_i = u_i$ in g such that t_i#S;E=A; u_i#S;E/B ¸ T_S but

t_i#S;E=A6=A u_i#S;E=A:

Some simplifications in the form of reduction to canonical forms can be made to ' , even if they do not yield a trivial proof of ' . In some cases, such canonical reductions are incorporated into ' and the Boolean transformation is used again.

4.2.2. Context joinability.

It checks whether ' is *context-joinable* [5]. The proof obligation ' is context-joinable iff $\bar{t}$? and $\bar{u}$ are joinable in the rewrite theory R^j_e = (S($\bar{X}$), A, Ē ·· ḡ), obtain by making variables into constants and by orienting the equations E as rewrite rules Ē and *heuristically* orienting each equality $t_i = u_i$ in g as a sequent $\bar{t}_i$ fi $\bar{u}_i$ in ḡ .

4.2.3. Unfeasability.

It checks if the proof obligation is *unfeasible* [5]. The proof obligationj is unfeasible if there is a conjunct $\bar{t}_i$ fi $\bar{u}_i$ in ḡ and v, w ¸ $T_S(X)$ *such that* R^j_e ⊢ $\bar{t}_i$ fi $\bar{u}$ Ù $\bar{t}_i$ fi $\bar{w}$ $CSU_A(v = w) =$ ˘ , and v and w are *strongly irreducible* with Ē modulo A, i.e., if v and w are such that each one of its ground instances is in E-canonical form modulo A.

4.2.4. SMT Solving.

It checks if the proof obligation can be proved by an SMT decision

procedure. The condition g of the proof obligation ' is analyzed and, if possible, a subformula consisting only of arithmetic subexpressions is extracted. This subformula has the following property: if it is a contradiction, then g is unsatisfiable. Therefore, if the SMT decision procedure answers that the given subformula is unsatisfiable, then, as in the previous test, ' is unfeasible.

Because of the admissibility assumptions on (S;*E* ¨ *A*), the first test of the strategy either succeeds or fails in finitely many equational rewrite steps. For the second and third tests, the strategy is not guaranteed to succeed or fail in finitely many rewrite steps because the oriented sequents g fi can falsify a termination assumption of the underlying equational theory. So, for these last two checks, InvA uses a bound on the depth of the proof-search. For the fourth test, the InvA offers support for integer linear arithmetic constraints, which is known to be decidable and for which there may be decision procedures already implemented in the SMT solver of choice.

The code in InvA for tests (2) and (3) was borrowed and adapted from the Church-Rosser Checker Tool [5]. For the test (4), the InvA tool relies on an extension of Maude with the CVC3 theorem prover available from the Matching Logic Project [15].

4.3. Comparison of the Proof-Search Tests

As noted before, the InvA tool has been successfully used in the formal verification of safety properties for concurrent systems in different domains. Its case studies include, among others, the IBOS secure browsing system [13, Ch. 6], the Alternating Bit Protocol [13, Ch. 5], and the Bakery Protocol for both bounded and unbounded number of processes. See the current version of the InvA tool for the complete set of case studies.

Test	Rate os Success
Equational simplification	54.2%
Context joinability(search deptg £ 10)	34.7%
Unfeasability (search depth £ 10)	9.3%
SMT -solving	1.8%

Table 1. Success rate of the automatic search-proof tests for discharged proof obligations

Table 1 compares the rate of success of each automatic search-proof test on the case studies that are part of the InvA's distribution for all automatically discharged proof obligations. Equational simplification is the most successful test, which is natural given the fact that many proof obligations can be handled by directly rewriting from the

definitions. Context joinability is also highly successful and it has proved to be a good complement to the former test. Unfeasability has a success rate of almost 10%, which is high taking into account its third place in the list and the fact that state predicates are not required to be fully defined for the negative case. Finally, SMTsolving is 1:8% successful mainly because, if possible, it is the last test ever applied to a proof obligation and also because many proof obligations in the case studies considered here do not have many integer arithmetic constraints.

5. A Case Study: The Bakery Protocol

This section presents a case study about the deductive analysis of inductive safety properties using the methodology, the proof system, and the Maude Invariant Analyzer tool (InvA) introduced in Section 3.. The subject of study is a concurrent protocol for achieving mutual exclusion. As a result, this section illustrates how the proof obligations generated during the verification task are automatically discharged by the proof-search heuristics presented in Section 4.. Full versions of the specification of the case study and the proof scripts presented here, can be downloaded with the InvA distribution.

5.1. The Bakery Protocol

The Bakery Protocol was proposed by L. Lamport as the first real solution for the mutual exclusion problem between processes. It derives its name from the situation in a busy bakery or deli shop where costumers pick a number at the ticket machine in order to guarantee that they are served in a proper order. The protocol is based on the "first in, first out" principle: the customer whose ticket matches the current available slot number is served first. The Bakery Protocol is considered as a benchmark case study in formal verification [12] because it is inherently concurrent and its state space is unbounded (i.e., it is potentially infinite). This section presents the Bakery Protocol (2BAK) specification for two processes.

The 2BAK specification in Maude has 2 modules. At the top level, he state space is represented by the top sort Sys, defined in module 2BAK-STATE, which is a 4-tuple:

```
sorts Mode Sys .
ops sleep wait crit : -> Mode [ctor] .
op <_,_,_,_> : Mode Nat Mode Nat -> Sys [ctor] .
```

The arguments of a system state are the state of the first customer (as Mode), the ticket number of the first customer (as Nat), the state of the second customer (as Mode), and the ticket number of the second

customer (as Nat). The sort Mode is an enumeration of constants for identifying an inactive customer (as sleep), a customer waiting to be served (as wait), and a customer being served (as crit). The sort Nat is that of natural numbers in Peano notation, together with an equality enrichment and the usual comparison operators.

As an example, consider the following ground term of sort Sys representing a state in the system:

```
< wait, s s 0, crit, s 0 >
```

In this state, the first customer is waiting with ticket number 2 and the second customer is being served with ticket number 1.

Module 2BAK specifies the operation of the protocol with 6 rewrite rules. These rewrite rules model the transitions of the customers through the three different states in the system.

```
rl [p1-s] :
   < sleep, M:Nat, Y:Mode, N:Nat >
=> < wait, s N:Nat, Y:Mode, N:Nat > .

crl [p1-w] :
   < wait, M:Nat, Y:Mode, N:Nat >
=> < crit, M:Nat, Y:Mode, N:Nat >
if M:Nat <= N:Nat = true .

rl [p1-c] :
   < crit, M:Nat, Y:Mode, N:Nat >
=> < sleep, 0, Y:Mode, N:Nat > .

rl [p2-s] :
   < X:Mode, M:Nat, sleep, N:Nat >
=> < X:Mode, M:Nat, wait, s M:Nat > .

crl [p2-w] :
   < X:Mode, M:Nat, wait, N:Nat >
=> < X:Mode, M:Nat, crit, N:Nat >
if N:Nat < M:Nat = true .

rl [p2-c] :
   < X:Mode, M:Nat, crit, N:Nat >
=> < X:Mode, M:Nat, sleep, 0 > .
```

The effects of these rules in a state can be summarized as follows:

[p1-s] models the transition of the first customer from state inactive to state waiting, with a new ticket number corresponding to the next available one (in this case, the next available ticket is the successor of the ticket number of the second customer).
[p1-w] models the transition of the first customer from state waiting to being served (i.e., to the critical section), whenever its ticket number is at most as the ticket number of the second customer.
[p1-c] models the transition of the first customer from being served to state inactive; the new ticket number is zero (which represents a non-existent ticket).
[p2-s] models the transition of the second customer from state inactive to state waiting, with a new ticket number corresponding to the next available one (in this case, the next available ticket is the successor of the ticket number of the first customer).
[p2-w] models the transition of the second customer from state waiting to being served (i.e., to the critical section), whenever its ticket number is greater than the ticket number of the first customer.
[p2-c] models the transition of the second customer from being served to state inactive; the new ticket number is zero (which represents a non-existent ticket).

5.2. Mutual Exclusion

The main property the 2BAK enjoys is the mutual exclusion property. This means that the protocol makes possible *to safely* serve the two costumers, i.e., to serve at most one customer at a time. The goal of the remaining of this section is to illustrate how the proof-search heuristics presented in Section 4. can automatically discharge *all* proof obligations generated by the InvA tool during the verification task of 2BAK's mutual exclusion property.

Mutual exclusion in 2BAK means that if one customer is being served, then the other customer is not being served. Note that this is a property that must hold for each pair of natural numbers. Thus, this property cannot be effectively checked by means of direct algorithmic techniques, such as model checking the 2BAK specification, even if the set of initial states is finite (which is not even the case for 2BAK, as explained below).

The mutual exclusion property is expressed by state predicate mutex and is defined in module 2BAK-PREDS as follows:

```
op mutex : Sys -> [Bool] .
eq [p1-s&p2-s] :
   mutex(< sleep, M:Nat, sleep, N:Nat >)
```

```
= true .
eq [p1-w&p2-s] :
    mutex(< wait, M:Nat, sleep, N:Nat >)
= true .
eq [p1-c&p2-s] :
    mutex(< crit, M:Nat, sleep, N:Nat >)
= true .
eq [p1-s&p2-w] :
mutex(< sleep, M:Nat, wait, N:Nat >)
= true .
eq [p1-w&p2-w] :
    mutex(< wait, M:Nat, wait, N:Nat >)
= true .
eq [p1-s&p2-c] :
    mutex(< sleep, M:Nat, crit, N:Nat >)
= true .
eq [p1-c&p2-c] :
    mutex(< crit, M:Nat, crit, N:Nat >)
= false .
eq [p1-c&p2-w] :
    mutex(< crit, M:Nat, wait, N:Nat >)
= M:Nat <= N:Nat .                        --- strengthening
eq [p1-w&p2-c] :
    mutex(< wait, M:Nat, crit, N:Nat >)
= N:Nat < M:Nat .                         --- strengthening
```

State predicate mutex is fully defined (i.e., for positive and negative cases) by nine equations. In particular, equations [p1-s&p2-s], [p1-w&p2-s], [p1-c&p2-s], [p1-s&p2-w], [p1-w&p2-w], and [p1-s&p2-c] consider the case when the mutual exclusion property is trivially true: there is at most one process in the critical section. Equation [p1-c&p2-c] considers the case when the property is trivially false: the two processes are in the critical section. Equations [p1-c&p2-w] and [p1-w&p2-c] consider the case when one process is in the critical section while the other one is waiting to enter the critical section. An initial observation would suggest that in such a state the property trivially holds. However, with a closer look it becomes evident that this is not the case because such a condition would be too week: if the waiting process has a ticket number less than the ticket number of the process in the critical section, then the waiting process would transition and enter the critical section, causing a mutual exclusion violation. This is evidence for requiring a stronger condition for equations [p1-c&p2-w] and [p1-w&p2-c] to be true. This observation is realized by the strengthening conditions on right-hand side of each one of these two rules.

Note that inference rule INV, presented in Section 3., does ensure that any reachable state from the set of initial states satisfies predicate mutex if it is an inductive invariant. Thus, if the latter is the case, then the mutual exclusion property also holds for a predicate similar to mutex in which the right-hand side of equations [p1-c&p2-w] and [p1-w&p2-c] is replaced by true (this is thanks to the strengthening rule STR1, presented in Section 3.). The above remarks and the strengthening of the invariant are omitted here for brevity, but are worked out in detail in the sources of the case study (available with the InvA distribution).

The goal is to prove that mutex is an inductive invariant of 2BAK for the set of initial states defined by predicate init, defined as follows:

```
op init : Sys -> [Bool] .
eq [init] :
      init(< sleep, M:Nat, sleep, N:Nat >)
= true .
```

The set of initial states corresponds to states in which both customers are inactive and can have any ticket number. Note that the set of initial states is countably infinite.

Formally, the goal is to check if state predicate mutex is an inductive invariant from init:

$\mathit{2BAK} \Vdash \mathit{init} \Rightarrow \Box \mathit{mutex}$:

The following verification commands can be given to the InvA tool in order to check the inductive invariant for achieving mutual exclusion in 2BAK:

```
(analyze init(S:Sys) implies mutex(S:Sys) in 2BAK-PREDS .)

(analyze-stable mutex(S:Sys) in 2BAK-PREDS 2BAK .)
```

When issuing the above-mentioned commands, the InvA tool generates the following output:

```
rewrites: 6540 in 12ms cpu (11ms real) (545000 rewrites/second)
Checking 2BAK-PREDS ||- init(S:Sys) => mutex(S:Sys) ...
Proof obligations generated: 1
Proof obligations discharged: 1
Success!
```

Test	Proof obligations discharged
Equational simplification	15
Context joinability (search depth 10)	2
Unfeasability (search depth 10)	0

Table 2. Success count of search-proof tests for the invariance of mutex in 2BAK.

```
Maude> (analyze-stable mutex(S:Sys) in 2BAK-PREDS 2BAK .)
rewrites: 14694 in 15ms cpu (15ms real) (979600 rewrites/second)
Checking 2BAK-PREDS ||- mutex(S:Sys) => O mutex(S:Sys) ...
Proof obligations generated: 18
Proof obligations discharged: 18
Success!
```

The tool generates 19 proof obligations and its search-proof tests automatically discharges all of them. Table 2 summarizes the effectiveness of each search-proof test in discharging the 19 proof obligations.

The following is the list of all proof obligations generated for the first above-mentioned command:

```
rewrites: 3247 in 2ms cpu (2ms real) (1623500 rewrites/second)
These are all proof obligations:
0. from init : trivially joinable
        mutex(< sleep,#1:Nat,sleep,#2:Nat >) = true
```

The following is the list of all proof obligations generated for the second abovementioned command:

```
0.  from p1-c&p2-c & p2-c : trivially joinable
    mutex(< crit,#3:Nat,sleep,0 >) = true
if false = true .

1.  from p1-c&p2-c & p1-c : trivially joinable
    mutex(< sleep,0,crit,#4:Nat >) = true
if false = true .

2.  from p1-c&p2-s & p2-s : trivially joinable
    mutex(< crit,#3:Nat,wait,s #3:Nat >) = true
3.  from p1-c&p2-s & p1-c : trivially joinable
    mutex(< sleep,0,sleep,#4:Nat >) = true

4.  from p1-c&p2-w & p2-w : smt
    mutex(< crit,#3:Nat,crit,#4:Nat >) = true
```

```
if #3:Nat <= #4:Nat = true
/\ #4:Nat < #3:Nat = true .

5.  from p1-c&p2-w & p1-c : trivially joinable
    mutex(< sleep,0,wait,#4:Nat >) = true
if #3:Nat <= #4:Nat = true .

6.  from p1-s&p2-c & p2-c : trivially joinable
    mutex(< sleep,#3:Nat,sleep,0 >) = true

7.  from p1-s&p2-c & p1-s : trivially joinable
    mutex(< wait,s #4:Nat,crit,#4:Nat >) = true

8.  from p1-s&p2-s & p2-s : trivially joinable
    mutex(< sleep,#3:Nat,wait,s #3:Nat >) = true

9.  from p1-s&p2-s & p1-s : trivially joinable
    mutex(< wait,s #4:Nat,sleep,#4:Nat >) = true

10. from p1-s&p2-w & p2-w : trivially joinable
    mutex(< sleep,#3:Nat,crit,#4:Nat >) = true
if #4:Nat < #3:Nat = true .

11. from p1-s&p2-w & p1-s : trivially joinable
    mutex(< wait,s #4:Nat,wait,#4:Nat >) = true

12. from p1-w&p2-c & p2-c : trivially joinable
    mutex(< wait,#3:Nat,sleep,0 >) = true
if #4:Nat < #3:Nat = true .

13. from p1-w&p2-c & p1-w : smt
    mutex(< crit,#3:Nat,crit,#4:Nat >) = true
if #4:Nat < #3:Nat = true
/\ #3:Nat <= #4:Nat = true .

14. from p1-w&p2-s & p2-s : trivially joinable
    mutex(< wait,#3:Nat,wait,s #3:Nat >) = true

15. from p1-w&p2-s & p1-w : trivially joinable
    mutex(< crit,#3:Nat,sleep,#4:Nat >) = true
if #3:Nat <= #4:Nat = true .

16. from p1-w&p2-w & p2-w : context joinable
    mutex(< wait,#3:Nat,crit,#4:Nat >) = true
if #4:Nat < #3:Nat = true .
```

17. from p1-w&p2-w & p1-w : context joinable
    mutex(< crit,#3:Nat,wait,#4:Nat >) = true
if #3:Nat <= #4:Nat = true .

Internally, the InvA tool classifies proof obligations discharged by equational simplification as "trivially joinable", the ones discharged by context joinability as "context joinable", the ones discharged by unfeasibility as "unfeasible", and the ones discharged by SMT-solving as "smt".

As a final remark, note that the proof obligations discharged by SMT-solving have unfeasible (arithmetic) conditions that could not be dealt with by the unfesability proof-search test, which is applied before calling the SMT solver. The reason for this 'failure' is that the equational specification for the operators <= and < is not convenient for search by equational narrowing. On the other hand, the arithmetic conditions in proof obligations (4) and (13) are easy tests for arithmetic decision procedures, such as the ones available from SMT solvers.

6. Conclusion and Future Work

This paper has presented the automatic proof-search tests implemented in the Maude Invariant Analyzer Tool (InvA). This tool offers interactive support, with a high degree of automation, for deductively proving inductive stability and inductive invariance of a (possibly infinite-state) concurrent system specified by conditional topmost rewrite theories under reasonable conditions. The proof obligations generated by the mechanized inference rules in the InvA tool are equational Horn clauses. The original safety properties of the concurrent system are automatically reduced to such equational proof obligations by the mechanization of these rules and the 1-step inductively complete narrowing procedures. The proof-search tests automatically discharge many of the resulting equational proof obligations. In some case studies these tests have achieved a success rate for automatically discharging proof obligations of 99:8% on thousands of proof obligations. In particular, the effectiveness of the proofsearch tests has been illustrated by a case study on the mutual exclusion property for Lamport's 2-process Bakery Protocol in which all proof obligations are automatically discharged without any user intervention. In the broader picture of the Maude formal environment and community, the InvA tool adds theorem proving support for verifying safety properties of infinite-state rewrite theories to the Maude environment.

The challenging case studies analyzed in the InvA tool have unveiled limitations of the InvA tool which, from the perspective of stress testing the limits of the tool, is a positive experience. First, there should be better management of proof obligations, specially when analyzing large

specifications: it is very complicated, time consuming, and error-prone to analyze a list of more than 30 proof obligations 'by hand'.

Second, the SMT-solving automatic test could perhaps be combined with equational narrowing, which is already available in Maude. This should increase the number of proof obligations automatically discharged by the tool and thus lessen the proof effort of the user. Third, there is also the need for improving the techniques available to the user in tools such as the ITP. These could help in obtaining easy interactive proofs in many cases where the proof obligations cannot be discharged automatically. Inductive techniques such as cover-set induction modulo AC should be investigated, implemented, and offered to the user. The current ITP version supports cover-set induction [8] but for the moment not modulo AC.

References

[1] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. Theoretical Computer Science, 360(1-3):386–414, 2006.

[2] E. M. Clarke, O. Grumberg, and D. A. Peled. Model Checking. The MIT Press, Cambridge, Massachusetts, 1999.

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, volume 4350 of Lecture Notes in Computer Science. Springer, 2007.

[4] M. Clavel and M. Egea. ITP/OCL: A rewriting-based validation tool for UML+OCL static class diagrams. In M. Johnson and V. Vene, editors, AMAST, volume 4019 of Lecture Notes in Computer Science, pages 368–373. Springer, 2006.

[5] F. Durán and J. Meseguer. A Church-Rosser checker tool for conditional ordersorted equational maude specifications. In P. C. Ölveczky, editor, WRLA, volume 6381 of Lecture Notes in Computer Science, pages 69–85. Springer, 2010.

[6] F. Durán, C. Rocha, and J. M. Álvarez. Towards a Maude Formal Environment. In Formal Modeling: Actors, Open Systems, Biological Systems, volume 7000 of Lecture Notes in Computer Science, pages 329–351, 2011.

[7] R. Gutiérrez, J. Meseguer, and C. Rocha. Order-sorted equality enrichments modulo axioms. In F. Durán, editor, Rewriting Logic

and Its Applications, volume 7571 of Lecture Notes in Computer Science, pages 162–181. Springer Berlin Heidelberg, 2012.

[8] J. Hendrix. Decision Procedures for Equationally Based Reasoning. PhD thesis, University of Illinois at Urbana-Champaign, April 2008.

[9] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science, 96(1):73–155, 1992.

[10] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, WADT, volume 1376 of Lecture Notes in Computer Science, pages 18–61. Springer, 1997.

[11] J. Meseguer and J. A. Goguen. Initially, induction and computability. Algebraic Methods in Semantics, 1986.

[12] K. Ogata and K. Futatsugi. Formal analysis of the bakery protocol with consideration of nonatomic reads and writes. In S. Liu, T. Maibaum, and K. Araki, editors, Formal Methods and Software Engineering, volume 5256 of Lecture Notes in Computer Science, pages 187–206. Springer Berlin Heidelberg, 2008.

[13] C. Rocha. Symbolic Reachability Analysis for Rewrite Theories. PhD thesis, University of Illinois at Urbana-Champaign, 2012.

[14] C. Rocha and J. Meseguer. Proving safety properties of rewrite theories. In A. Corradini, B. Klin, and C. Cîrstea, editors, CALCO, volume 6859 of Lecture Notes in Computer Science, pages 314–328. Springer, 2011.

[15] G. Ro¸su and A. ¸ Stef?anescu. Matching Logic: A New Program Verification Approach (NIER Track). In ICSE'11: Proceedings of the 30th International Conference on Software Engineering, pages 868–871. ACM, 2011.

[16] S. Tang, H. Mai, and S. T. King. Trust and protection in the Illinois Browser Operating System. In R. H. Arpaci-Dusseau and B. Chen, editors, OSDI, pages 17–32. USENIX Association, 2010.

[17] P. Viry. Equational rules for rewriting logic. Theoretical Computer Science, 285:487–517, 2002.