Electronic Communications of the EASST
Volume 72 (2015)

Proceedings of the
15th International Workshop on
Automated Verification of Critical Systems (AVoCS 2015)

First-order logic for safety verification of hedge rewriting systems

Alexei Lisitsa

14 pages

Guest Editors: Gudmund Grov, Andrew Ireland
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

First-order logic for safety verification of hedge rewriting systems

Alexei Lisitsa[1]

Department of Computer Science, The University of Liverpool

Abstract: In this paper we deal with verification of safety properties of hedge rewriting systems and their generalizations. The verification problem is translated to a purely logical problem of finding a finite countermodel for a first-order formula, which is further tackled by a generic finite model finding procedure. We show that the proposed approach is at least as powerful as the methods using regular invariants. At the same time the finite countermodel method is shown to be efficient and applicable to the wide range of systems, including the protocols operating on unranked trees.

Keywords: hedge rewriting, safety verification, first-order logic

1 Introduction

Hedges, or arbitrary width trees over unranked alphabets, also known as *unranked trees* or *forests* provide with the abstractions useful in several verification contexts. Hedges and hedge transformations (rewritings) have been used for specification and verification of at least the following:

- Protocols working in tree-shaped networks of unbounded degree [27];
- XML transformations [30, 23, 27];
- Multithreaded recursive programs [8, 29].

The most challenging task of infinite state or parameterized verification appears when one would like to establish the correctness of a system in a widest possible context - either for all possible sizes of the system or for unbounded computations. In general such a problem is undecidable and the only way to address it is to focus on restricted classes of systems and properties.

In this paper we consider the problem of *safety* verification for hedge rewriting systems and explore the applicability of the very general method based on disproving the first-order formulae by finding a countermodel. The basic intuition behind the method is that if the evolution of the system of interest can be faithfully modelled by the derivations within first-order logic then the safety (non-reachability of bad states) can be naturally reduced to disproving the formulae (non-provability).

Such an approach to the safety verification has been proposed in the early work on the verification of cryptographic protocols [28, 25, 12] and later has been extended to various classes of parameterized and infinite state verification tasks [16, 18, 19, 20, 21].

Two attractive properties of the approach have emerged. On the one side it has turned out to be practically efficient in many applications. On the other side it has proved to be relatively complete with respect to the methods using *regular* invariants, in particular regular model checking, regular tree model checking and tree completion.

Safety verification for hedge rewriting systems has been already addressed e.g. in [14, 11, 27] with the *regular* invariants playing the major role.

We show in this paper the Finite CounterModel method (FCM) provides a viable alternative to the methods using regular invariants for the verification of hedge rewriting systems. We show that theoretically the proposed approach is at least as powerful as the methods using regular invariants. At the same time the finite countermodel method is shown to be very flexible and applicable to the wide range of systems. The practical efficiency of the method is illustrated on a set of examples on verification of (1) parameterized protocols operating on arbitrary width tree topology – for most of them an automated verification is reported for the first time; and (2) dynamic multithreaded recursive programs.

The paper is organized as follows. The next section provides with necessary preliminaries. In Section 3 we formulate basic verifcation problem and its translation into a logical problem of first-order formulae disproving. In Section 4 the verification method is discussed and its relative completeness is demonstrated. Section 5 presents the experimental results and Section 6 concludes the paper.

2 Preliminaries

2.1 First-order Logic

A *first-order vocabulary* is defined as a finite set $\Sigma = \mathscr{F} \cup \mathscr{P}$ where $\mathscr{F}$ and $\mathscr{P}$ are the sets of functional and predicate symbols, respectively. Each symbol in Σ has an associated arity, and we have $\mathscr{F} = \cup_{i\geq 0}\mathscr{F}_i$ and $\mathscr{P} = \cup_{i\geq 1}\mathscr{P}_i$, where $\mathscr{F}_i$ and $\mathscr{P}_i$ consist of symbols of arity i. The elements of $\mathscr{F}_0$ are also called *constants*.

A *first-order model* over vocabulary Σ, or just a *model* is a pair $\mathscr{M} = \langle D, [\![\Sigma]\!]_D\rangle$ where D is a non-empty set called *domain* of $\mathscr{M}$ and $[\![\Sigma]\!]_D$ denotes the *interpretations* of all symbols from Σ in D. For a domain D and a function symbol f of arity $n \geq 1$ an interpretation of f in D is a function $[\![f]\!]_D : D^n \to D$. For a constant c its interpretation $[\![c]\!]_D$ is an element of D. For a domain D and a predicate symbol P of arity n an interpretation of P in D is a relation of arity n on D, that is $[\![P]\!]_D \subseteq D^n$. The model $\mathscr{M} = \langle D, [\![\Sigma]\!]_D\rangle$ is called *finite* if D is a finite set.

We assume that the reader is familiar with the standard definitions of first-order formula, first-order sentence, satisfaction $\mathscr{M} \models \varphi$ of a formula φ in a model $\mathscr{M}$, deducibility (derivability) $\Phi \vdash \varphi$ of a formula φ from a set of formulae Φ, first-order theory and equational theory.

2.2 Hedge Rewriting

We borrow the definitions from [14] with minor modifications. Let Σ be a finite alphabet and $\mathscr{X}$ be a countable set of variables. Then the set of *terms* $\mathscr{T}(\Sigma, \mathscr{X})$ and the set of hedges $\mathscr{H}(\Sigma, \mathscr{X})$, both over Σ and $\mathscr{X}$, are defined inductively as the least sets satisfying

- $\mathscr{T}(\Sigma, \mathscr{X}) = \mathscr{X} \cup \{f(h) \mid f \in \Sigma, h \in \mathscr{H}(\Sigma, \mathscr{X})\}$
- $\mathscr{H}(\Sigma, \mathscr{X}) = \bigcup_{n>0}\{t_1, \dots, t_n \mid t_j \in \mathscr{T}(\Sigma, \mathscr{X}), j = 1, \dots, n\} \cup \{\Lambda\}$

Thus, a hedge is a finite (possibly empty) sequence of terms, whereas a term is obtained by applying an unranked functional symbol to a hedge. We denote by Λ the empty sequence of terms. We do not make a difference between a term and a hedge of length one, i.e. consider that $\mathscr{T}(\Sigma,\mathscr{X}) \subset \mathscr{H}(\Sigma,\mathscr{X})$. We will also do not distinguish the hedges of the form $f_{i_1}(\Lambda),\dots,f_{i_k}(\Lambda)$ and words $f_{i_1},\dots,f_{i_k} \in \Sigma^*$.

The sets of ground terms and ground hedges (i.e., terms and hedges without variables) are denoted $\mathscr{T}(\Sigma)$ and $\mathscr{H}(\Sigma)$. A variable $x \in \mathscr{X}$ is called *linear* in a hedge $h \in \mathscr{H}(\Sigma,\mathscr{X})$ if it has exactly one occurrence in h.

The set of variables occurring in a term $t \in \mathscr{T}(\Sigma,\mathscr{X})$ is denoted $var(t)$. A *substitution* σ is a mapping from $\mathscr{X}$ to $\mathscr{H}(\Sigma,\mathscr{X})$. The application of a substitution σ to a hedge h, denoted $h\sigma$, is defined inductively as follows. For $t_1,\dots,t_n \in \mathscr{T}(\Sigma,\mathscr{X})$ we have $(t_1,\dots,t_n)\sigma = t_1\sigma\dots t_n\sigma$ and $f(h)\sigma = f(h\sigma)$.

A *context* is a hedge $h \in \mathscr{H}(\Sigma,\mathscr{X})$ with a distinguished variable x linear in h. We write $C[x]$ to denote a context with a distinguished variable x. The application of a context $C[x]$ to a hedge h is defined by $C[h] = C\{x \mapsto h\}$.

A hedge rewriting system (HRS) is a set of rewriting rules of the form $l \to r$, where $l, r \in \mathscr{H}(\Sigma,\mathscr{X})$. The rewrite relation $\to_{\mathscr{R}}$ (resp. *outer* rewrite relation $\rightsquigarrow_{\mathscr{R}}$) of an HRS $\mathscr{R}$ is defined as follows: $h \to_{\mathscr{R}} h'$ iff there is a context $C[x]$ (resp. a trivial context $C[x] = x$), a rule $l \to r \in \mathscr{R}$ and a substitution σ such that $h = C[l\sigma]$ and $h' = C[r\sigma]$. The reflexive and transitive closure of $\to_{\mathscr{R}}$ (resp. $\rightsquigarrow_{\mathscr{R}}$) is denoted by $\to^*_{\mathscr{R}}$ (resp. $\rightsquigarrow^*_{\mathscr{R}}$).

Given a set of ground hedges $L \subseteq \mathscr{H}(\Sigma)$ and an HRS $\mathscr{R}$ we denote by $post^*_{\mathscr{R}}(L)$ the set of all hedges reachable from L, that is $\{h \in \mathscr{H}(\Sigma) \mid \exists g \in L, g \to^*_{\mathscr{R}} h\}$. Similarly $post^*_{\mathscr{R},\rightsquigarrow}(L)$ denotes $\{h \in \mathscr{H}(\Sigma) \mid \exists g \in L, g \rightsquigarrow^*_{\mathscr{R}} h\}$.

2.3 Forest Automata and Regular Hedge Languages

The following definition is taken from [5].

Definition 1 A forest automaton over an unranked alphabet Σ is a tuple $\mathscr{A} = ((Q,e,*),\Sigma,\delta : (\Sigma \times Q \to Q), F \subseteq Q)$ where $(Q,e,*)$ is a finite monoid, δ is a transition function and F is a set of accepting states.

For every *ground* hedge h the automaton assigns a value $h^{\mathscr{A}} \in Q$ which is defined by induction:

- $\Lambda^{\mathscr{A}} = e$;
- $f(h)^{\mathscr{A}} = \delta(f, h^{\mathscr{A}})$
- $(t_1,\dots t_n)^{\mathscr{A}} = t_1^{\mathscr{A}} * \dots * t_n^{\mathscr{A}}$

A hedge h is accepted by the forest automaton $\mathscr{A}$ iff $h^{\mathscr{A}} \in F$. The hedge language $L_{\mathscr{A}}$ of the forest automaton $\mathscr{A} = ((Q,e,*),\Sigma,\delta : (\Sigma \times Q \to Q), F \subseteq Q)$ is defined as $L_{\mathscr{A}} = \{h \mid h^{\mathscr{A}} \in F\}$.

A hedge language L is called *regular* iff it is a hedge language $L_{\mathscr{A}}$ of some forest automaton $\mathscr{A}$. Alternative but equivalent definitions of regular hedge languages using *hedge* automata and *unranked tree automata* have been considered e.g. in [14, 27, 26, 15].

2.4 Finitely based sets of hedges

For a finite set of hedges $B \subseteq \mathscr{H}(\Sigma,\mathscr{X})$ the set of ground instances of all hedges from B is denoted by $GI(B)$, that is $GI(B) = \{h \mid \exists h' \in B \wedge h = h'\theta;\ h'\theta \text{ is ground}\}$.

Example 1 *For* $\Sigma = \{a,b,f\}$ *and* $B = \{f(x,b,b,y)\}$ *the language* $GI(B)$ *is the set of all ground hedges with the outer symbol* f *and containing two consecutive symbols* b *at leaves.*

Notice that finitely based sets of hedges are not necessarily regular; on the other hand, it is easy to see that if every hedge in a set B is *linear* then $GI(B)$ is regular.

3 Safety Verification: from hedge rewriting to FO logic

In this section we define the basic verification problem and its translation into a purely logical problem of disproving the first-order formula.

3.1 Basic verification problem

The general form of safety verification problems we address in this paper is as follows.

Given: An unranked vocabulary Σ, a hedge rewriting system $\mathscr{R}$ over Σ, a language $I \subseteq \mathscr{H}(\Sigma)$ of *initial* ground hedges, a language $U \subseteq \mathscr{H}(\Sigma)$ of *unsafe* ground hedges.

Question: Is it true that $\forall h \in I \forall h' \in U\ \ h \not\to^{*}_{\mathscr{R}} h'$?

We will also consider a variant of the basic verification problem for outer rewriting, where $\to^{*}$ above is replaced by $\rightsquigarrow^{*}$.

Notice that in the definition of the basic verification problem the sets I and U, in general, may be *infinite*. In that case we assume that the sets are defined by some finitary and constructive means, for example as regular languages given by forest automata, or as finitely based sets of hedges.

3.2 From hedge rewriting to first-order logic

For an unranked alphabet Σ we denote by Σ_{fo} a ranked vocabulary $\Sigma^{r} \cup \{e,*\} \cup \{R^{(2)}\}$ where $\Sigma^{r} = \{\tilde{f} \mid f \in \Sigma\}$, with all $\tilde{f}$ being unary functional symbols, e is a constant (0-ary functional) symbol, $*$ is a binary functional symbol, which we will use in infix notation and R is a binary predicate symbol. The constant e will denote the empty hedge, i.e. the hedge of length 0, $*$ will denote the *concatenation* of hedges and the semantics of R is going to capture *reachability* for hedge rewriting.

Then we define the translation $\tau : \mathscr{H}(\Sigma,\mathscr{X}) \to \mathscr{T}(\Sigma_{fo},\mathscr{X})$ from hedges to terms over the extended alphabet inductively as follows:

- $\tau(x) = x$ for $x \in \mathscr{X}$

- $\tau(\Lambda) = e$
- $\tau(t_1,\dots,t_n) = \Pi_{i=1}^{n}\tau(t_i)$
- $\tau(f(h)) = \tilde{f}(\tau(h))$

Here $\Pi_{i=1}^{n}\tau(t_i)$ denotes the $*$-product of $\tau(t_i)$ defined as

- $\Pi_{i=1}^{1}\tau(t_i) = \tau(t_1)$
- $\Pi_{i=1}^{k+1}\tau(t_i) = (\Pi_{i=1}^{k}\tau(t_i)) * \tau(t_{k+1})$

Notice that we will specify *associativity* of $*$, so exact association of brackets in the product will be immaterial.

For a hedge rewriting system $\mathscr{R}$ over alphabet Σ and a set of variables $\mathscr{X}$ we define its translation $\Phi_{\mathscr{R}}$ as the set of the following universally closed first-order formulae:

1. $(x*y)*z = x*(y*z)$
2. $e*x = x$
3. $x*e = x$
4. $R(\tau(h_1),\tau(h_2))$ for all $(h_1 \to h_2) \in \mathscr{R}$
5. $R(x,y) \to R((z*x)*v,(z*y)*v)$
6. $R(x,y) \to R(\tilde{f}(x),\tilde{f}(y))$ for all unary $\tilde{f} \in \Sigma_{fo}$
7. $R(x,x)$
8. $R(x,y) \wedge R(y,z) \to R(x,z)$.

Proposition 1 *(Adequacy of the first-order translation) $h_1 \to_{\mathscr{R}}^{*} h_2 \Leftrightarrow \Phi_{\mathscr{R}} \vdash R(\tau(h_1),\tau(h_2))$ for all ground h_1 and h_2.*

Proof.

$\Rightarrow$ Due to transitivity of R specified in $\Phi_{\mathscr{R}}$ it is sufficient to show that if $h_1 \to_{\mathscr{R}} h_2$ then $\Phi_{\mathscr{R}} \vdash R(\tau(h_1),\tau(h_2))$. Assume $h_1 \to_{\mathscr{R}} h_2$ then $h_1 = C[l\sigma]$ and $h_2 = C[r\sigma]$ for some context $C[x]$, some substitution σ and a rewriting rule $(l \to r) \in \mathscr{R}$. Now the argument proceeds by a simple induction on the context construction. Indeed, for the base case of the simplest context $C[x] \equiv x$ we have $h_1 \equiv l\sigma$ and $h_2 \equiv r\sigma$ and $R(\tau(h_1),\tau(h_2))$ is a ground instance of the formula $R(\tau(l),\tau(r)) \in \Phi_{\mathscr{R}}$ (item 4 in the translation). It follows $\Phi_{\mathscr{R}} \vdash R(\tau(h_1),\tau(h_2))$. There are two step cases. Assume $C[x] \equiv g_1C'[x]g_2$ where g_1 and g_2 are ground hedges and $C'[x]$ is a context. Then we have $h_1 \equiv g_1C'[l\sigma]g_2$, $h_2 \equiv g_1C'[r\sigma]g_2$ and $C'[l\sigma] \to_{\mathscr{R}} C'[r\sigma]$. By induction hypothesis $\Phi_R \vdash R(\tau(C'[l\sigma]),\tau(C'[r\sigma]))$. That together with the congruence axiom (5) being in $\Phi_{\mathscr{R}}$ implies $\Phi_{\mathscr{R}} \vdash R(\tau(g_1)*\tau(C'[l\sigma])*\tau(g_2),\tau(g_1)*\tau(C'[r\sigma])*\tau(g_2))$, and therefore, by definition of τ, $\Phi_{\mathscr{R}} \vdash R(\tau(h_1),\tau(h_2))$. The second step case with $C[x] \equiv \tilde{f}(C'[x])$ is dealt with in a similar way using the congruence axiom (6).

$\Leftarrow$ Consider the following first-order model $\mathfrak{M}$ in the vocabulary Σ_{fo}. The domain of the model is a set $\mathscr{H}(\Sigma)$ of all ground hedges over Σ. The interpretations of functional symbols and constants are defined inductively as

- $[\![e]\!] = \Lambda$;
- $[\![x * y]\!] = [\![x]\!][\![y]\!]$;
- $[\![f(x)]\!] = f([\![x]\!])$ for all $f \in \Sigma$.

The interpretation of R is given by $[\![R]\!] = \{(h,h') \mid h,h' \in \mathscr{H}(\Sigma) \wedge h \to^*_{\mathscr{R}} h'\}$. By simple structural induction we have $[\![\tau(h)]\!] = h$. That concludes the construction of $\mathfrak{M}$.

A straightforward check shows now that for such defined $\mathfrak{M}$ we have $\mathfrak{M} \models \Phi_{\mathscr{R}}$. Assume now $\Phi_{\mathscr{R}} \vdash R(\tau(h_1), \tau(h_2))$. It follows that $\mathfrak{M} \models R(\tau(h_1), \tau(h_2))$, that is $([\![\tau(h_1)]\!], [\![\tau(h_2)]\!]) \in [\![R]\!]$, and therefore $(h_1, h_2) \in [\![R]\!]$, which in turn means $h \to^*_{\mathscr{R}} h'$ by definition of $[\![R]\!]$. □

Taking a contraposition of Proposition 1 we get

$$\Phi_{\mathscr{R}} \not\vdash R(\tau(h_1), \tau(h_2)) \Rightarrow h_1 \not\to^*_{\mathscr{R}} h_2$$

which expresses the essence of the proposed verification method: in order to prove non-reachability($\approx$ safety) it is sufficient to disprove a first-order formula. In order to apply this observation to the solution of basic verification problems we need to provide also the first-order representations for the sets of initial and unsafe terms. We start with very general

Definition 2 Given an unranked alphabet Σ and a set of ground hedges $H \subseteq \mathscr{H}(\Sigma)$ a first-order formula $\varphi_H(x)$ with one free variable and in vocabulary Σ_{fo}, possibly extended by relational and fucntional symbols, is called first-order representation of H if $h \in H$ implies $\vdash \varphi_H(\tau(h))$.

3.2.1 First-order representations of regular hedge languages

Let $L_{\mathscr{A}}$ be a regular hedge language given by a forest automaton $\mathscr{A} = ((Q, e, *), A, \delta : (A \times Q \to Q), F \subseteq Q)$.

The vocabulary of the first-order representation of $\mathscr{A}$ consists of unary predicates P_q for each $q \in Q$ and unary functional symbols $\tilde{a}$ for each $a \in A$.

Define $\Phi_{\mathscr{A}}$ as the set of the following universally closed formulae:

1. $(x * y) * z = x * (y * z)$;
2. $(x * e) = x$;
3. $(e * x) = x$;
4. $P_{\delta(a,e)}(\tilde{a}(e))$ for every $a \in A$;
5. $P_q(x) \to P_{\delta(a,q)}(\tilde{a}(x))$ for every $a \in A$ and $q \in Q$.
6. $P_{q_1}(x) \wedge P_{q_2}(y) \to P_{q_3}(x * y)$ for all $q_1, q_2, q_3 \in Q$ such that $q_1 * q_2 = q_3$ in Q.

Now for the forest automaton $\mathscr{A}$ we denote by $\Phi_{L_{\mathscr{A}}}$ the formula $\Phi_{\mathscr{A}} \to \bigvee_{q \in F} P_q(x)$

Proposition 2 $\Phi_{L_{\mathscr{A}}}$ *is a first-order representation of* $L_{\mathscr{A}}$

Proof *(Sketch)* Straightforward induction on the hedge construction shows that $\Phi_{\mathscr{A}} \vdash P_{h^{\mathscr{A}}}(\tau(h))$. The proposition statement follows immediately. □

3.2.2 Finitely based sets of hedges

For a finite set of hedges $B \subseteq \mathscr{H}(\Sigma, \mathscr{X})$ the formula $\bigvee_{h \in B}(\exists \bar{y}(x = \tau(h)))$ is a first-order representation of $GI(B)$. Here $\bar{y}$ denotes all variables in $\tau(h)$ and x is different from all variables in $\bar{y}$.

3.3 Outer rewriting and unary reachability encoding

In many specification and verification scenarii *outer* rewriting is sufficient to model all essential aspects of the evolution of the system of interest. In that case the first-order translations can be simplified as there is no need in the congruence axioms and the *unary reachability* predicate does suffice.

Let $\mathscr{R}$ be a hedge rewriting system over alphabet Σ. We define its *unary* translation $\Phi^U_{\mathscr{R}}$ as the set of the following universally closed first-order formulae:

1. $(x * y) * z = x * (y * z)$;
2. $(x * e) = x$;
3. $(e * x) = x$;
4. $R(\tau(h_1)) \rightarrow R(\tau(h_2))$ for all $(h_1 \rightarrow h_2) \in \mathscr{R}$.

Such a simplified translation captures reachability by the outer rewrite relation $\rightsquigarrow_{\mathscr{R}}$ as the following proposition states.

Proposition 3 *(Adequacy of the unary first-order translation)* $h_1 \rightsquigarrow^*_{\mathscr{R}} h_2 \Leftrightarrow \Phi^U_{\mathscr{R}} \wedge R(\tau(h_1)) \vdash R(\tau(h_2))$ *for all ground* h_1 *and* h_2.

Proof (Hint) The proof follows the arguments in the proof of Proposition 1 appropriately modified. To prove $\Rightarrow$ entailment an easy induction on the length of rewriting is used. To prove $\Leftarrow$ a semantical argument taking as the interpretation of unary predicate R the set of the term encodings of all reachable from h_1 hedges, is applied. □

4 Back to safety verification

Now we have defined all necessary concepts and the basic verification problem defined in Sect. 3.1 gets its full specification.

Given: An unranked vocabulary Σ, a hedge rewriting system $\mathscr{R}$ over Σ, a first-order representation φ_I of the language $I \subseteq \mathscr{H}(\Sigma)$ of *initial* ground hedges, a first-order representation φ_U of the language $U \subseteq \mathscr{H}(\Sigma)$ of *unsafe* ground hedges.

Question: Is it true that $\forall h \in I \forall h' \in U \;\; h \not\rightarrow^*_{\mathscr{R}} h'$?

Proposition 4 *If the verification problem above has a negative answer, that is* $\exists t_1 \in I \exists t_2 \in U t_1 \rightarrow^*_{\mathscr{R}} t_2$ *then* $\Phi_{\mathscr{R}} \vdash \exists x\, \exists y\, (\varphi_I(x) \wedge \varphi_U(y) \wedge R(x,y))$.

Proof. The proposition statement follows immediately from Proposition 1 and Definition 2. □

4.1 Verification method

Taking contraposition of Proposition 4 we have $\Phi_{\mathscr{R}} \not\vdash \exists x\, \exists y\, (\varphi_I(x) \wedge \varphi_U(y) \wedge R(x,y))$ implies $\forall h \in I \forall h' \in U \;\; h \not\rightarrow^*_{\mathscr{R}} h'$, that is a positive answer to an instance of the basic verification problem.

Thus, the essence of the finite countermodels (FCM) verification method we advocate in this paper is:

> To demonstrate a positive answer to an instance of a basic verification problem apply an automated generic finite model procedure to find a countermodel to $\Phi_{\mathscr{R}} \rightarrow \exists x \exists y \varphi_I(x) \wedge \varphi_U(y) \wedge R(x,y)$, or equivalently a model for $\Phi_{\mathscr{R}} \wedge \neg \exists x \exists y \varphi_I(x) \wedge \varphi_U(y) \wedge R(x,y)$.

The next theorem, which is a generalization of similar theorems in [17, 21] (the case of word languages) and [19] (the case of tree languages), shows the applicability of FCM method and its relative completeness with respect to the methods based on *regular* invariants [14, 27].

Theorem 1 *Let* $\mathscr{R}$ *be a hedge rewriting system over unranked alphabet* Σ, φ_I *and* φ_U *be first-order formulae representing the regular sets* I *and* U *of initial and unsafe hedges, respectively. Let* Inv *be a regular set of hedges such that* $post^*_{\mathscr{R}}(I) \subseteq Inv$ *and* $Inv \cap U = \emptyset$, *that is a regular invariant separating* I *and* U. *Then there is a finite model* $\mathscr{M}$ *such that* $\mathscr{M} \models \Phi_{\mathscr{R}} \wedge \neg \exists x,y (R(x,y) \wedge \varphi_I(x) \wedge \varphi_U(y))$.

Proof. (*Sketch*) Assume the condition of the theorem holds. Let $\mathscr{A}_I = ((Q_I, e_I, *_I), A, \delta_I, F_I \subseteq Q_I)$, $\mathscr{A}_{Inv} = ((Q_{Inv}, e_I, *_{Inv}), A, \delta_{Inv}, F_{Inv} \subseteq Q_{Inv})$ and $\mathscr{A}_U = ((Q_U, e_I, *_U), A, \delta_U, F_U \subseteq Q_U)$ be forest automata recognizing regular hedge languages I,Inv and U, respectively. Then the required finite model $\mathscr{M}$ is constructed as follows. The domain D of the model is $Q_I \times Q_{Inv} \times Q_U$. The interprettaion of e is $[\![e]\!] = (e_I, e_{Inv}, e_U)$. The interpretation of $*$ is given by $(q_1,q_2,q_3)[\![*]\!](q'_1,q'_2,q'_3) = (q_1 *_I q'_1, q_2 *_{Inv} q'_2, q_3 *_U q'_3)$. For all $a \in A$ the interpretations are given by $[\![a]\!](q_1,q_2,q_3) = (\delta_I(a,q_1), \delta_{Inv}(a,q_2), \delta_U(a,q_3))$. Once we defined the interpretations of all functional symbols (including constants) any ground term t gets its interpretation $[\![t]\!] \in D$ in a standard way. Define the interpretation of R as $[\![R]\!] = \{([\![\tau(h)]\!], [\![\tau(h')]\!] | h \rightarrow^*_{\mathscr{R}} h'\}$. For all predicates P_q with $q \in Q_I$ used in φ_I the interpretations are defined as $[\![P_q]\!] = \{(q,x,y) | x \in Q_{Inv}, y \in Q_U\}$. Similarly, $[\![P_q]\!] = \{(x,q,y) | x \in Q_I, y \in Q_U\}$ for $q \in Q_{Inv}$ and $[\![P_q]\!] = \{(x,y,q) | x \in Q_I, y \in Q_{Inv}\}$. Now it is straightforward exercise to show that the such defined model is indeed as required. □.

5 Experiments

In this section we present the experimental results of applications of the FCM method for safety verification of hedge rewriting systems. In the experiments we used the finite model finder Mace4 [24] within the package Prover9-Mace4, version 05, December 2007[1].

5.1 Tree arbiter protocol and other protocols on unranked trees

We consider here as a case study the verification of parameterized *Tree-arbiter Protocol* working on the trees of unbounded branching degree. We take the description of the protocol from [4] where its automated verification has been demonstrated for the case of *binary* trees.

The protocol supervises the access to a shared resource of a set of processes arranged in a tree topology. The processes competing for the resource are located in the leaves of the trees. The safety property to be verified is that of *mutual exclusion* - at no point, two or more processes at leaves of the tree can get an access to the resource (obtain a token). A process in the protocol can be in state *idle* (i), *requesting* (r), *token* (t) or *below* (b). All the processes are initially in state i. A node is in state b whenever there is a node below (descendant) in state t. When a leaf is in state r, the request is propagated upwards until it encounters a node in state t of b (aware of the presence of the token). A node state t can choose to pass it upwards or pass it downwards to a requesting node in state r.

We model the tree arbiter protocol as a hedge rewriting system $\mathscr{R}_{ta}$ consisting of the rules:

1. $i(x\,r(y)\,z) \to r(xr(y)z)$ (request propagated upwards)
2. $t(x\,r(y)\,z) \to b(xt(y)z)$ (token passed downwards)
3. $b(x\,t(y)\,z) \to t(xi(y)z)$ (token passed upwards)
4. $i(\Lambda) \to r(\Lambda)$ (idle leaf node becomes requesting)

The set of initial configurations is a hedge language consisting of all hedges in which all leaves are either $r(\Lambda)$ or $i(\Lambda)$, all intermediate nodes (neither leaves, nor the top) are $i(\ldots)$ and the top node is $t(\ldots)$.

The hedge language of unsafe configurations consists of all hedges with two or more leaves with $t(\Lambda)$ (tokens). Now we represent the verification problem in the format required by FCM method.

The automata $\mathscr{A}_{\mathscr{I}}$ and $\mathscr{A}_{\mathscr{U}}$ reconginizing the sets of initial, respectively unsafe states and the first-order translation Φ of the hedge rewriting system and the automata can be found in the Appendix A of the extended version [22] of this paper.

After translation of the protocol specification and forest automata recognizing the initial and unsafe configurations into a first order formula Φ and *(un)safety* condition into a formula Ψ the safety is established automatically by Mace4 finding a countermodel for $\Phi \to \Psi$ of size 3 in 0.03s.

[1] the system configuration used in experiments: Intel(R) Core (TM)2 Duo CPU, T7100 1.8Ghz 1.79 Ghz, 1.00 GB of RAM.

We have applied FCM to the verification of other protocols operating on the unranked trees. The specification of the protocols are taken from [4, 6] where their automated verification has been demonstrated for the binary trees case. In [19] we have considered the application of FCM for the verification of binary tree case protocols. The table below lists the parameterized tree protocols and shows the time it took Mace4 to find a countermodel and verify a safety property for the case of unranked trees (first column). For comparison the times it took to verify the *binary tree versions* of the same protocols by FCM [19] and by regular tree model checking [6] are given in the second and third columns, respectively.

Protocol	Time	Time from [19]	Time from[6]*
Token	0.04	0.02s	0.06s
Two-way Token	0.04	0.03s	0.09s
Percolate	0.06	0.09s	2.4s
Tree arbiter	0.03	0.03s	0.31s

* the system configuration used in [6] was *Intel Centrino 1.6GHZ with 768MB of RAM*

Further details of the experiments can be found in [16].

As far as we are aware the automated verification of the above parameterized protocols over trees of unbounded degree is reported here for the first time, with an exception being Two-way Token protocol verified in [27].

5.2 Verification of Synchronised PAD Systems

The verification techniques based on disproving in first-order logic are very flexible and can accommodate various constraints on the properties of the systems of interest. To illustrate this point we consider an automated verification of a program which involves *dynamic creation of processes*. We take as an example a program from [29] specified in terms of Synchronised PAD Systems [29]. Such systems can be seen as an extension of hedge rewriting systems to the case of two *different* associative constructors, one of which being commutative. Furthermore the rewriting is *constrained* to capture the effects of modelled synchronisation.

Let $Sync = \{a, b, c, \ldots\}$ be a set of *actions* such that every action $a \in Sync$ corresponds to a co-action $\bar{a} \in Sync$ s.t. $\bar{\bar{a}} = a$. Let $Act = Sync \cup \{\tau\}$ be the set of all the actions, where τ is a special action.

Let $Var = \{X, Y, \ldots\}$ be a set of process variables and $\mathscr{T}$ be the set of *process terms t* over *Var* defined by:

$$t ::= 0 \mid X \mid t \cdot t \mid t || t$$

Definition 3 A Synchronised PAD (SPAD) is a finite set of rules of the form $X \hookrightarrow^a t$ or $X \cdot Y \hookrightarrow^a t$, where $X, Y \in Var, t \in \mathscr{T}$ and $a \in Act$

The process terms are considered modulo the following equational theory *SE* of *structural equivalence:*

A1: $t \cdot 0 = 0 \cdot t = t || 0 = 0 || t = t$

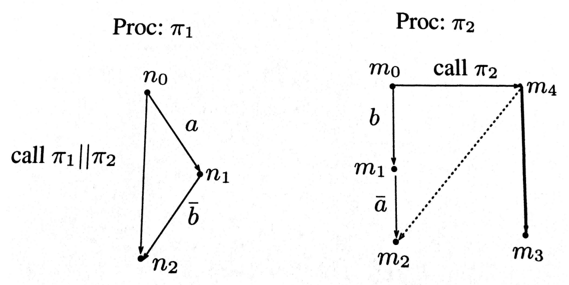

Figure 1: An example from [29]

A2: $(t \cdot t') \cdot t'' = t \cdot (t' \cdot t'')$

A3: $t||t' = t'||t$

A4: $(t||t')||t'' = t||(t'||t'')$

A SPAD $\mathscr{R}$ induces a transition relation $\rightarrowtail^a$ over $\mathscr{T}$ by the following inference rules:

$$\theta_1 : \frac{t_1 \hookrightarrow^a t_2 \in \mathscr{R}}{t_1 \rightarrowtail^a t_2};\ \theta_2 : \frac{t_1 \rightarrowtail^a t_1'}{t_1 \cdot t_2 \rightarrowtail^a t_1' \cdot t_2};\ \theta_3 : \frac{t_1 = 0, t_2 \rightarrowtail^a t_2'}{t_1 \cdot t_2 \rightarrowtail^a t_1 \cdot t_2'}$$

$$\theta_4 : \frac{t_1 \rightarrowtail^a t_1'}{t_1||t_2 \rightarrowtail^a t_1'||t_2; t_2||t_1 \rightarrowtail^a t_2||t_1'}\ \theta_5 : \frac{t_1 \rightarrowtail^a t_2; t_2 \rightarrowtail^{\bar{a}} t_2'; a, \bar{a} \in Sync}{t_1||t_2 \rightarrowtail^{\tau} t_1'||t_2'}$$

The equational theory SE induces a transition relation $\rightarrowtail^a_{SE}$ over $\mathscr{T}$ defined by $\forall t, t' \in \mathscr{T}, t \rightarrowtail^a_{SE} t'$ iff $\exists u, u'$ s.t. $t =_{SE} u, u \rightarrowtail^a u'$ and $u' =_{SE} t'$. For $w \in Act^*$ the transition relations $\rightarrowtail^w$ and $\rightarrow^w_{SE}$ are defined in a standard way.

5.2.1 Example

Consider the program represented in Fig 1 which involves *dynamic creation of processes*. The figure represents the flow graph of a program having two procedures π_1 and π_2 such that:

- π_1 calls itself in parallel with another procedure π_2.
- π_2 calls itself recursively,
- π_1 and π_2 communicate via the synchronizing actions a, b, and their corresponding co-actions $\bar{a}$ and $\bar{b}$, and the program starts at point n_0

The safety property to be verified is "starting from n_0, the program never reaches a configuration where the control point m_1 is active."

The program is modelled by the Syncronized PAD $\mathscr{R}$ [29] which includes the following rules:

$\mathscr{R}_1 : n_0 \hookrightarrow^{a} n_1$
$\mathscr{R}_2 : n_1 \hookrightarrow^{\bar{b}} n_2$
$\mathscr{R}_3 : n_0 \hookrightarrow^{\tau} (n_0 \parallel m_0) \cdot n_2$
$\mathscr{R}_4 : m_0 \hookrightarrow^{b} m_1$
$\mathscr{R}_5 : m_1 \hookrightarrow^{\bar{a}} m_2$
$\mathscr{R}_6 : m_0 \hookrightarrow^{\tau} m_0 \cdot m_4$
$\mathscr{R}_7 : m_3 \cdot m_4 \hookrightarrow^{\tau} m_2$
$\mathscr{R}_8 : m_2 \cdot m_4 \hookrightarrow^{\tau} m_3$
The above verification task is formulated in terms of $\mathscr{R}$ as follows. Show that there are no $w \in \tau^*$ and $t \in \mathscr{T}$ such that m_0 is a subterm of t and $n_0 \rightarrowtail^{w}_{SE} t$.

Notice that the transition relation $\rightarrowtail^{w}_{SE}$ with $w \in \tau^*$ encodes the reachability by a rewriting process, where the rules with $\hookrightarrow^{\tau}$ can be applied without any restrictions, whereas the rules with $\hookrightarrow^{a}$ with $a \in Sync$ can be applied only if a some rule $\hookrightarrow^{\bar{a}}$ with a co-action superscript is applied simultaneously "in parallel" subterm (cf. the rule θ_5).

We claim now that one can specify the first-order formulae:

- $\Phi_{\mathscr{R}}$ describing the reachability by $\rightarrowtail^{w}_{SE}$, the initial configuration and the set of unsafe configurations, and
- Ψ specifying unsafety condition

such that if $\Phi_{\mathscr{R}} \vdash \Psi$ then the program is unsafe. See the details in Appendix B of the extended version [22] of this paper. We apply Mace4 and a countermodel of the size 3 for $\Phi_{\mathscr{R}} \rightarrow \Psi$ is found in 2s.

6 Conclusion

We have shown in this paper that the simple encoding of hedge rewriting systems by first-order logic and using available finite model finders provides with an interesting and viable alternative to the existing methods for the verification of systems working on the trees of unbounded degree. On the one side it is relatively complete w.r.t. the methods using regular invariants. On the other side it is either practically efficient, as our automated verification of parameterized protocols working on the trees of unbounded degree illustrates well, or at least looks promising to explore the boundaries of its practical applicability, as our SPAD example shows. The advantages of the FCM method are (1) it is simple, (2) it is flexible, (3) it is modular and (4) it reuses existing tools. The FCM method has a natural limitation that it can be used only to establish the safety. If safety actually does not hold looking for the countermodels does not help. One can use then automated theorem provers to confirm that indeed the safety is violated by searching for the proofs of formulas of the form $\Phi \rightarrow \Psi$. One direction for the future work here is the development of the systematic procedures which would extract the unsafe traces from first-order proofs.

Bibliography

[1] Parosh Aziz Abdulla, Jonsson B. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91-101, June 15, 1996.

[2] Abdulla, Parosh Aziz and Jonsson, Bengt and Mahata, Pritha and d'Orso, Julien, Regular Tree Model Checking, in Proceedings of the 14th International Conference on Computer Aided Verification, CAV '02, 2002, 555–568

[3] Abdulla, P.A., Jonsson,B., Nilsson, M., & Saksena, M., (2004) A Survey of Regular Model Checking, In Proc. of CONCUR'04, volume 3170 of LNCS, pp 35–58, 2004.

[4] Parosh Aziz Abdulla, Noomene Ben Henda, Giorgio Delzanno, Frdric Haziza and Ahmed Rezine, Parameterized Tree Systems, in Formal Techniques for Networked and Distributed Systems, FORTE 2008, Lecture Notes in Computer Science, 2008, Volume 5048/2008, 69-83.

[5] Mikolaj Bojanczyk, Igor Walukiewicz, Forest Algebras, in Logic and Automata, History and Perspectives (J.Flum, E.Gradel, T. Wilke eds.), Amsterdam University Press, 2007, 107–132.

[6] A. Bouajjani, P. Habermehl,A. Rogalewicz, T. Vojnar, Abstract Regular Tree Model Checking, Electronic Notes in Theoretical Computer Science, 149, (2006), 37–48.

[7] Parosh Aziz Abdulla, Giorgio Delzanno, Noomene Ben Henda, Ahmed Rezine. Monotonic Abstraction: on Efficient Verification of Parameterized Systems. *Int. J. Found. Comput. Sci.* 20(5): 779-801 (2009)

[8] A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems, in Proc. 16th Int. Confenerence on Rewriting Techniques And Applications (RTA'05), LNCS 3467, 2005

[9] R. Caferra, A. Leitsch, N. Peltier, *Automated Model Building*, Applied Logic Series, 31, Kluwer, 2004.

[10] G. Delzanno. Constraint-based Verification of Parametrized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3):257–301, 2003.

[11] Julien d'Orso and Tayssir Touili. Regular Hedge Model Checking, Proc. of the 4th IFIP International Conference on Theoretical Computer Science (TCS'06), 2006.

[12] Goubault-Larrecq, J., (2010), Finite Models for Formal Security Proofs, *Journal of Computer Security*, 6: 1247–1299, 2010.

[13] Guttman, J., (2009) Security Theorems via Model Theory, Proceedings 16th International Workshop on Expressiveness in Concurrency, EXPRESS, EPTCS, vol. 8 (2009)

[14] F. Jacquemard and M. Rusinowitch, Closure of Hedge-Automata Languages by Hedge Rewriting, in A. Voronkov (Ed.): RTA 2008, LNCS 5117, pp 157–171, 2008.

[15] L. Libkin, Logics for unrankeed trees: an overview. *Logical Methods in Computer Science*,2:1-31,2006.

[16] A. Lisitsa Verification via countermodel finding
`http://www.csc.liv.ac.uk/~alexei/countermodel/`

[17] Lisitsa, A., (2010c), Finite model finding for parameterized verification, CoRR abs/1011.0447: (2010)

[18] Lisitsa, A., (2010b), Reachability as deducibility, finite countermodels and verification. In Proceedings of ATVA 2010, LNCS 6252, 233–244

[19] Lisitsa, A., (2011), Finite countermodels for safety verification of parameterized tree systems, CORR abs/1107.5142 (2011)

[20] Lisitsa, A, (2012) Finite Models vs Tree Automata in Safety Verification, In Proc. RTA'12, pp 225–239

[21] Lisitsa, A., (2013) Finite Reasons for Safety. Parameterized Verification by Finite Model Finding, *Journal of Automated Reasoning*, 51(4): 431-451 (2013)

[22] Lisitsa, A., (2015) First-order logic for safety verification of hedge rewriting systems, extended version of this paper, available at `www.csc.liv.ac.uk/~alexei/countermodel/avocs15.pdf`

[23] S. Maneth, A. Berlea, T. Perst, and H. Seidl, Xml type checking with macro tree transducers, PODS, 2005

[24] W. McCune Prover9 and Mace4 `http://www.cs.unm.edu/~mccune/mace4/`

[25] Selinger, P., (2001), Models for an adversary-centric protocol logic. Electr. Notes Theor. Comput. Sci. 55(1) (2001)

[26] J.W. Tatcher, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.,* 1:317-322, 1967.

[27] Tayssir Touili, Computing transitive closures of hedge transformations, *IJCCBS*, volume 3, 1/2, 2012, 132–150

[28] Weidenbach, C., (1999), Towards an Automatic Analysis of Security Protocols in First-Order Logic, in H. Ganzinger (Ed.): CADE-16, LNAI 1632, pp. 314–328, 1999.

[29] Tayssir Touili. Dealing with communication for dynamic multithreaded recursive programs. Invited Paper In Proc. of VISSAS'05.

[30] Silvano Dal Zilio and Denis Lugiez, XML schema, tree logic and sheaves automata, RTA'03, 2003.