4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

Rapport de Recherche

Functional Term Rewriting Systems

Yohan Boichut, Jean-Michel Couvreur,
Duy-Tung Nguyen
LIFO, Université d'Orléans

Rapport n^o **RR-2010-02**

Functional Term Rewriting Systems

Yohan Boichut, Jean-Michel Couvreur, Duy-Tung Nguyen

Université d'Orléans,
Laboratoire d'Informatique Fondamentale d'Orléans,
F-45067 ORLEANS Cedex 2, France
{yohan.boichut,jean-michel.couvreur,duy.nguyen}@univ-orleans.fr
http://www.univ-orleans.fr/lifo/

Abstract. This reseach report proposes the theoretical foundations of a new formal tool for symbolic verification of finite systems. Some approaches reduce the problem of system verification to the reachability problem in term rewriting systems (TRSs). In our approach, states are encoded by terms in a BDD-like manner and the transition relation is represented by a new rewriting relation so called *Functional Term Rewriting Systems* (FTRSs). First, we show that FTRSs are as expressive as TRSs. Then, we focus on a subclass of FTRSs, so called *Elementary Functional Term Rewriting Systems* (EFTRSs), and we show that EFTRSs preserve the FTRSs expressiveness. The main advantage of EFTRSs is that they are well adapted for acceleration techniques usually used in saturation algorithms on BDD-like data structures. Our experiments show that for well-known protocols (e.g. Tree Arbiter, Percolate, Round Robin Mutex protocols,...) our tool is not only better than other rewriting tools such as Timbuk or Maude, but also competitive with other model-checkers such as SPIN, NuSMV or SMART. Moreover, it can also be applied to model-checking invariant properties which are a particular subclass of linear temporal logic formula (LTL).

1 Introduction

In the earliest 90s, in order to show their product (complex electronic components) secure, electronics industry integrates some techniques coming from the world of formal methods: the Binary Decision Diagrams (BDDs) [6, 7]. BDDs are structures coding boolean functions.

They can be considered as trees where nodes specify the choice of a value for the variable corresponding to this node. A total ordered relation on variables ensures the existence of a canonical BDD representation of a given boolean function. Some reduction techniques and data structures well adapted for data sharing leads to very efficient implementation in practice [42, 36]. Considering a state or a configuration of a system as a BDD and the transition relation as operations on BDDs, the techniques mentioned above have made the exhaustive exploration, verification of huge systems (sometimes composed of more than 1 Billion states [42, 36]) possible.

Moreover, as shown in [8], the BDDs are expressive enough for handling a large class of finite systems. In [38], these structures have even been shown to be adapted for some classes of dynamic systems.

The key point of these data structures being the number of variables, some people work on BDD-like data structures [43, 12]. Some others have worked in order to make the application domain of BDDs larger : [3, 34, 33, 39, 44, 40].

In this paper, we aim at proposing a technique mixing acceleration techniques commonly used in the BDD world for saturation [14], and rewriting techniques. Indeed, given a system, its configurations are encoded by terms and the transition relation is described by a *functional term rewriting system* (FTRS). FTRSs are shown to be as expressive as term rewriting systems (TRSs). We also focus on a subclass of FTRSs so called *elementary functional term rewriting systems* (EFTRSs). EFTRSs preserve the expressiveness of FTRS and are well adapted to acceleration techniques.

The remainder of the paper is structured as follows. Section 2 recalls the background on terms, TRSs and rewriting. Section 3 illustrates how systems are specified by terms and how the verification of systems can be performed using rewriting techniques. Section 4 and 5 introduce FTRS and EFTRS, and present our main results concerning their expressiveness, i.e., FTRS and EFTRS are as expressive as TRS. Finally, Section 6 shows how the system described in Section 3 is specified in terms of EFTRS. This section also describes how model-checking can be performed using FTRS/EFTRS and promising results in the field of state space exploration and model-checking are exhibited. For well-known protocols (e.g. Tree Arbiter, Percolate, Round Robin Mutex protocols,...) our tool is not only better than other rewriting tools such as Timbuk [31, 27] or Maude [19], but also competitive with other model-checkers such as SPIN [35], NuSMV [17] or SMART [10].

2 Preliminaries on Terms and TRSs

Comprehensive surveys can be found in [24, 2] for term rewriting systems.

For now, we focus on terms composed of binary functional symbols and of the constant $\perp$. We denote by $\mathcal{F}_{bin}$ this set of symbols. Let $\mathcal{X}$ be a countable set of variables. $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ denotes the set of terms built on those symbols and containing variables, and $\mathcal{T}(\mathcal{F}_{bin})$ denotes the set of ground terms (terms without variables). A substitution is a function σ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, which can be extended uniquely to an endomorphism of $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$. A position p for a term t is a word over $\mathbb{N}$. The empty sequence ϵ denotes the top-most position. The set $\mathcal{P}os(t)$ of positions of a term t is inductively defined by $\mathcal{P}os(f(t_1, t_2)) = \{\epsilon\} \cup \{1.p \mid \text{ and } p \in \mathcal{P}os(t_1)\} \cup \{2.p \mid \text{ and } p \in \mathcal{P}os(t_2)\}$, and $\mathcal{P}os(t) = \{\epsilon\}$ when $t \in \mathcal{X}$ or $t = \perp$. Let $\preceq$ be a partial order on positions. Let p_1 and p_2 be two words of $\mathbb{N}^*$, we say that $p_1 \preceq p_2$ if there exists $\omega \in \mathbb{N}^*$ such that $p_1 = p_2.\omega$.

If $p \in \mathcal{P}os(t)$, then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s. We also denote by $t(p)$ the symbol occurring in t at position p. Given a term

$t \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ and A a set of symbols, let $\mathcal{P}os_A(t) = \{p \in \mathcal{P}os(t) \mid t(p) \in A\}$. Thus $\mathcal{P}os_{\mathcal{F}}(t)$ is the set of positions of t, at each of which a function symbol appears. The set of *frontier* positions of a term t, denoted by $\mathcal{FP}\text{os}(t)$, is defined as follows: $\mathcal{FP}\text{os}(t) = \{p \in \mathcal{P}os(t) \mid p.1 \notin \mathcal{P}os(t)\}$. The set of variables occurring a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is denoted by $\mathcal{V}ar(t)$. More formally, $\mathcal{V}ar(t) = \{t|_p \mid p \in \mathcal{P}os_{\mathcal{X}}(t)\}$.

A TRS $\mathcal{R}$ is a set of *rewrite rules* $(l, r) \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, also denoted by $l \to r$ where $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ and $l \notin \mathcal{X}$. The TRS $\mathcal{R}$ induces a rewriting relation $\to_{\mathcal{R}}$ on terms whose reflexive transitive closure is written $\to^*_{\mathcal{R}}$. More precisely, we say that $t \to_{\mathcal{R}} t'$ if there exist a position p of $\mathcal{P}os(t)$, a rule $l \to r \in \mathcal{R}$ and a substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ such that $t|_p = l\sigma$ and $t' = t[r\sigma]_p$.

The set of $\mathcal{R}$-descendants of a set of terms $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ is $\mathcal{R}^*(E) = \{s \in \mathcal{T}(\mathcal{F}_{bin}) \mid t \to^*_{\mathcal{R}} s \wedge t \in E\}$.

3 Verification of Systems using Rewriting

This section shows how the tree arbiter protocol (TAP), a protocol solving mutual exclusion problem for accessing to a shared resource, can be formalised in terms and TRSs. We also describe how a safety property concerning this protocol can be formulated in a reachability problem.

3.1 Presentation of the Tree Arbiter Protocol (TAP)

As mentioned above, the TAP is an asynchronous circuit that solves the mutual exclusion problem by building a tree of arbiter cells. This circuit has been introduced by Dill [25] and has been handled as a problem of asynchronous model-checking in [18]. Recently, a version of the TAP has been tackled using a technique of tree regular model-checking in [1].

The circuit works by performing elimination rounds: an arbiter cell arbitrates between its two children. The leaves of the tree are processors, which may want to access asynchronously a shared resource. The N processors at the lowest level are arbitrated by N/2 cells. The winners of that level are arbitrated by the next level, and so forth.

3.2 Specification of the TAP in Terms and TRS

We use the binary symbols i, r, t and b respectively for the specification of states *idle, requesting, token* and *token below* where:

- state *idle* means that all children of a cell (or a processor) do(es) not do anything.
- state *requesting* means that this processor (or one of children of this cell) wants to access the shared resource.
- state *token* means that this processor (or this cell) has been granted the shared resource.

– state *token below* means that the shared resource (or the token) is somewhere in one of subtree below this node.

Given an initial *total idle* state for a model of 4 processors $u = t(i(i(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot)))$ where all processors are *idle* and the root cell holds the token (See Figure 1a).

Requests are propagated upwards until the root, holding the token, is reached:

$$i(x, r(y, z)) \rightarrow r(x, r(y, z)) \quad (1)$$
$$i(r(x, y), z) \rightarrow r(r(x, y), z) \quad (2)$$
$$i(\bot, \bot) \rightarrow r(\bot, \bot) \quad (3)$$

By using only these requests rules, the system may reach a *total requesting* state $v = t(r(r(\bot, \bot), r(\bot, \bot)), r(r(\bot, \bot), r(\bot, \bot)))$ (See Figure 1b).

The root cell grants the resource to at most one child, and the grant propagates downward to one of the processors. If both children of a cell are requesting the resource, then the cell chooses one of them non-deterministically:

$$t(x, r(y, z)) \rightarrow b(x, t(y, z)) \quad (4)$$
$$t(r(x, y), z) \rightarrow b(t(x, y), z) \quad (5)$$

Consequently, in such a way, only a processor having requested the resource will get it. When the processor is done with the resource, it sends a release request that is propagated upwards.

$$b(x, t(\bot, \bot)) \rightarrow t(x, i(\bot, \bot)) \quad (6)$$
$$b(t(\bot, \bot), z) \rightarrow t(i(\bot, \bot), z) \quad (7)$$

Along the *come-back to the root* process of the token, three behaviours are possible for a cell:

– If all of the children of such a cell are in the *idle* state then this cell gives the token to the next higher neighboured cell and goes also to the *idle* state.

$$b(x, t(i(y, z), i(u, v))) \rightarrow t(x, i(i(y, z), i(u, v))) \quad (8)$$
$$b(t(i(x, y), i(u, v)), z) \rightarrow t(i(i(x, y), i(u, v)), z) \quad (9)$$

– If one of the children of such a cell has requested the resource then:
  • either the next state of the cell will be the *request* one and the token goes to the next higher neighboured cell using following rules

$$b(x, t(r(y, z), i(u, v))) \rightarrow t(x, r(r(y, z), i(u, v))) \quad (10)$$
$$b(x, t(i(y, z), r(u, v))) \rightarrow t(x, r(i(y, z), r(u, v))) \quad (11)$$
$$b(t(r(x, y), i(u, v)), z) \rightarrow t(r(r(x, y), i(u, v)), z) \quad (12)$$
$$b(t(i(x, y), r(u, v)), z) \rightarrow t(r(i(x, y), r(u, v)), z) \quad (13)$$

  • or the next state of the cell will be the *token below* one because the cell has chosen to give the token to one of its requesting children using rules (4) or (5).

3.3 Verification using Reachability Analysis

An interesting safety property ϕ to check on the TAP is that there is always only one token circulating in the hierarchy tree. So, one can formulated this property as a predicate to check on each reachable configuration.

Considering the symbols $\top$ and $\bot$ as respectively the *true* and *false* values, let $OneToken$ be the predicate defined as follows for a term $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$:

$$OneToken(\alpha) = \begin{cases} \top \;\; if\ \forall p, p' \mathcal{P}os(\alpha), \\ \quad \alpha(p) = \alpha(p') = t \Rightarrow p = p' \\ \bot \;\; otherwise. \end{cases}$$

Consequently,

$$TAP\ satisfies\ \phi \Leftrightarrow \forall \alpha \in \mathcal{R}^*(\{u\}), OneToken = \top,$$

with u is the term $t(i(i(\bot,\bot),i(\bot,\bot)),i(i(\bot,\bot),i(\bot,\bot)))$.

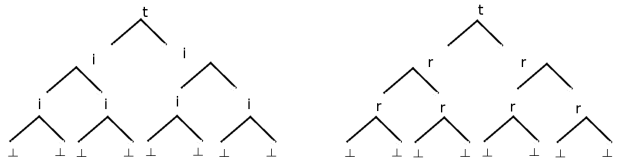

Fig. 1: a) *total idle* global state u and b) *total requesting* global state v

4 Functional Term Rewriting Systems

In this section, we propose a new formalism, so called *Functional Term Rewriting Systems* (FTRSs) being able to simulate classical TRS. The motivations of this new formalism are on the one hand that to make the control of the rewriting process easier and completly independent of the implementation of the rewriting engine, and on the other hand to offer a formalism well adapted to acceleration techniques commonly used in saturation algorithm on BDDs.

We first introduce new functional symbols so called non-terminals. We denote by $\mathcal{F}_{NT}$ the set of non-terminal symbols such that $\mathcal{F}_{NT} \cap \mathcal{F}_{bin} = \emptyset$ and for each $F \in \mathcal{F}_{NT}$, the arity of F is one. To make the distinction between functional symbols easier, from now, only functional symbols of $\mathcal{F}_{NT}$ will be written in capital letters. A functional TRS $\mathcal{R}_\lambda$ is a set of rules of the form $F(t) \rightarrow \alpha$ where $F \in \mathcal{F}_{NT}$, $t \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ and $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$. A FTRS $\mathcal{R}_\lambda$ induces a functional rewriting relation $\rightarrow_{\mathcal{R}_\lambda}$ on terms in $\mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$: $u \rightarrow_{\mathcal{R}_\lambda} v$ iff there exist a position p of u and a rule $F(t) \rightarrow \alpha$ in $\mathcal{R}_\lambda$ such that there exists a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}_{bin})$ with $u|_p = F(t)\sigma$ and $v = u[\alpha\sigma]_p$.

We also denote by $\rightarrow^*_{\mathcal{R}_\lambda}$ the transitive closure of the functional rewriting relation $\rightarrow_{\mathcal{R}_\lambda}$. The following definition describes the set of terms of $\mathcal{T}(\mathcal{F}_{bin})$ reachable by function rewriting from a term of $\mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$.

Definition 1 ($\mathcal{R}^*_\lambda$). *Let $\mathcal{R}_\lambda$ be a FTRS and $E \subseteq \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$ such that $E \cap \mathcal{T}(\mathcal{F}_{bin}) = \emptyset$. The set of reachable terms of $\mathcal{T}(\mathcal{F}_{bin})$ from E is denoted by $\mathcal{R}^*_\lambda(E)$ and is defined as follows:*

$$\mathcal{R}^*_\lambda(E) = \{\beta \in \mathcal{T}(\mathcal{F}_{bin}) | \alpha \rightarrow^*_{\mathcal{R}_\lambda} \beta \wedge \alpha \in E\}.$$

Now, let us compare the expressiveness of FTRSs and TRSs. Proposition 1 shows that a rewrite step using $\mathcal{R}$ can be simulated with a FTRS $\mathcal{R}_\lambda$.

Proposition 1. *Let $\mathcal{R}$ be a TRS over $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$, $\beta \in \mathcal{T}(\mathcal{F}_{bin})$ and $l \rightarrow r \in \mathcal{R}$. Let $\mathcal{R}_\lambda$ be the FTRS such that $\mathcal{R}_\lambda = \{F_{l\rightarrow r}(a(x,y)) \rightarrow a(F_{l\rightarrow r}(x), y), F_{l\rightarrow r}(a(x,y)) \rightarrow a(x, F_{l\rightarrow r}(y)) \mid a \in \mathcal{F}_{bin}\} \cup \{F_{l\rightarrow r}(l) \rightarrow r\}$. Thus,*

$$\alpha \rightarrow_{\{l\rightarrow r\}} \beta \Leftrightarrow F_{l\rightarrow r}(\alpha) \rightarrow^*_{\mathcal{R}_\lambda} \beta.$$

Proof.
- $\alpha \rightarrow_{\{l\rightarrow r\}} \beta \Rightarrow F_{l\rightarrow r}(\alpha) \rightarrow^*_{\mathcal{R}_\lambda} \beta$: By Definition, there exist a position p of α and a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}_{bin})$ such that $\alpha|_p = l\sigma$ and $\beta = \alpha[r\sigma]_p$. Let α' be the term constructed as follows: $\alpha' = \alpha[F_{l\rightarrow r}(\alpha|_p)]_p$. Clearly, using only rules of the form $F_{l\rightarrow r}(a(x,y)) \rightarrow a(F_{l\rightarrow r}(x), y)$ and $F_{l\rightarrow r}(a(x,y)) \rightarrow a(x, F_{l\rightarrow r}(y))$, one has

  $$F_{l\rightarrow r}(\alpha) \rightarrow^*_{\mathcal{R}_\lambda} \alpha'. \tag{14}$$

  By hypothesis, there exist a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}_{bin})$ such that $\alpha|_p = l\sigma$. Consequently, one has $F_{l\rightarrow r}(\alpha|_p) = F_{l\rightarrow r}(l\sigma)$. Thus, $F_{l\rightarrow r}(\alpha|_p) = F_{l\rightarrow r}(l)\sigma$. By construction of α', there exist a rule of $\mathcal{R}_\lambda$, i.e. $F_{l\rightarrow r}(l) \rightarrow r$, a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}_{bin})$ and a position p such that $\alpha'|_p = F_{l\rightarrow r}(l)\sigma$. Consequently, one can construct the term $\alpha'[r\sigma]$ which is equal to β by construction of α'. So, one has

  $$\alpha' \rightarrow_{\mathcal{R}_\lambda} \beta. \tag{15}$$

  So, using (14) and (15), one can deduce that $F_{l\rightarrow r}(\alpha) \rightarrow^*_{\mathcal{R}_\lambda} \beta$. Consequently, $\alpha \rightarrow_{\{l\rightarrow r\}} \beta \Rightarrow F_{l\rightarrow r}(\alpha) \rightarrow^*_{\mathcal{R}_\lambda} \beta$.
- $\alpha \rightarrow_{\{l\rightarrow r\}} \beta \Leftarrow F_{l\rightarrow r}(\alpha) \rightarrow^*_{\mathcal{R}_\lambda} \beta$: Let $\mathcal{R}'_\lambda$ be the FTRS such that $\mathcal{R}'_\lambda = \mathcal{R}_\lambda \setminus \{F_{l\rightarrow r}(l) \rightarrow r \mid l \rightarrow r \in \mathcal{R}\}$.. Trivially, one can show that $\mathcal{R}'^*_\lambda(\{F_{l\rightarrow r}(\alpha)\}) = \emptyset$. Consequently, it implies that, for obtaining the term $\beta \in \mathcal{R}^*_\lambda(\{F_{l\rightarrow r}(\alpha)\})$, the rule $F_{l\rightarrow r}(l) \rightarrow r$ has to be applied at least one time. Actually, we claim that this rule is applied exactly one times. Indeed, let us proceed by induction on the rewriting path leading us from $\mathcal{F}_{l\rightarrow r}(\alpha)$ to β i.e. there exist $t_0, \ldots, t_n \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$ such that $t_0 = F_{l\rightarrow r}(\alpha) \rightarrow_{\mathcal{R}_\lambda} t_1 \rightarrow_{\mathcal{R}_\lambda} \ldots t_{n-1} \rightarrow_{\mathcal{R}_\lambda} t_n = \beta$. We first show that each t_i has one position p at most such that $t_i(p) = F_{l\rightarrow r}$. Let P_n be the following proposition: for all $p, p' \in \mathcal{P}os(t_n)$, if $t_n(p) = t_n(p') = F_{l\rightarrow r}$ then $p = p'$.

- P_0: Since $t_0 = F_{l\to r}(\alpha)$ and $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$, ε is the unique position of t_0 where $F_{l\to r}$ occurs. So, P_0 is true.
- $P_n \Rightarrow P_{n+1}$: Suppose P_n to be true. Consequently, t_n satisfies the property: $p, p' \in \mathcal{P}os(t_n)$, if $t_n(p) = t_n(p') = F_{l\to r}$. Moreover, since $t_n \to_{\mathcal{R}_\lambda} t_{n+1}$, there exists necessarily a unique position p of t_n such that $t_n(p) = F_{l\to r}$. Let us proceed by rewriting case analysis:
  - $F_{l\to r}(a(x,y)) \to a(F_{l\to r}(x), y)$ is applied at position p: So there exists a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $t_n|_p = F_{l\to r}(a(x,y))\sigma = F_{l\to r}(a(\sigma(x), \sigma(y)))$. Consequently, $t_{n+1} = t_n[a(F_{l\to r}(x), y)\sigma] = t_n[a(F_{l\to r}(\sigma(x)), \sigma(y))]$. Since $\sigma(x), \sigma(y) \in \mathcal{T}(\mathcal{F}_{bin})$ and $\forall p' \in \mathcal{P}os(t_n)$ such that $p \neq p'$, $t_n(p') \notin \mathcal{F}_{NT}$, t_{n+1} satisfies also $\forall p, p' \in \mathcal{P}os(t_{n+1})$, if $t_{n+1}(p) = t_{n+1}(p') = F_{l\to r}$. More precisely, the unique position p' of t_{n+1} such that $t_{n+1}(p) = F_{l\to r}$ is $p' = p.1$.
  - $F_{l\to r}(a(x,y)) \to a(x), F_{l\to r}(y)$ is applied at position p: the proof case is similar to the one above.
  - $F_{l\to r}(l) \to r$ is applied at position p: So there exists a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $t_n|_p = F_{l\to r}(l)\sigma = F_{l\to r}(l\sigma)$. Consequently, $t_{n+1} = t_n[r\sigma]$. Since $r \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ and $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$, $r\sigma$ is thus a term of $\mathcal{T}(\mathcal{F}_{bin})$ and $t_{n+1} \in \mathcal{T}(\mathcal{F}_{bin})$. So, t_{n+1} satisfies the property: $\forall p, p' \in \mathcal{P}os(t_{n+1})$, if $t_{n+1}(p) = t_{n+1}(p') = F_{l\to r}$.

We have just shown that each term within the rewriting path contains exactly one symbol $F_{l\to r}$ and as soon as the rule $F_{l\to r}(l) \to r$ is applied, a term of $\mathcal{T}(\mathcal{F}_{bin})$ is obtained. So the FTRS $\mathcal{R}_\lambda$ cannot be applied anymore. Proving the claim.
So, the rewriting path leading us from $F_{l\to r}(\alpha)$ to β is of the following form: $t_0 = F_{l\to r}(\alpha) \to_{\mathcal{R}_\lambda \setminus \{F_{l\to r}(l)\to r\}} t_1 \to_{\mathcal{R}_\lambda \setminus \{F_{l\to r}(l)\to r\}} \dots t_{n-1} \to_{\{F(l)\to r\}} t_n = \beta$. With an induction very close to the previous one, one can show that there exist a position p of α and a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $t_{n-1} = \alpha[F(l)\sigma]_p$. One can show also that $t_{n-1}|_{p.1} = \alpha|_p$. Consequently,

$$l\sigma = \alpha|_p. \tag{16}$$

Moreover, $t_n = \beta = t_{n-1}[r\sigma]_p$. Thus,

$$\beta = \alpha[r\sigma]_p. \tag{17}$$

By construction of $\mathcal{R}_\lambda$, $l \to r \in \mathcal{R}$. So, according to (16) and (17), there exist a rule $l \to r \in \mathcal{R}$, a position $p \in \mathcal{P}os(\alpha)$ and a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $l\sigma = \alpha|_p$ and $\beta = \alpha[r\sigma]_p$. Consequently, $\alpha \to_{\{l\to r\}} \beta$. To summarize, we have shown that $\alpha \to_{\{l\to r\}} \beta \Leftarrow F_{l\to r}(\alpha) \to^*_{\mathcal{R}_\lambda} \beta$, concluding the proof.

Thus, the main result of this section is given in Theorem 1 i.e. for any TRS $\mathcal{R}$, one can build a FTRS $\mathcal{R}_\lambda$ simulating $\mathcal{R}$.

Theorem 1. *Let $\mathcal{R}$ be a TRS over $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$. Let $l \to r$ be a rule of $\mathcal{R}$ and $\mathcal{R}_{l\to r}$ be the FTRS such that $\mathcal{R}_{l\to r} = \{F_{l\to r}(a(x,y)) \to$*

$a(F_{l\to r}(x), y), F_{l\to r}(a(x, y)) \to a(x, F_{l\to r}(y)) \mid a \in \mathcal{F}_{bin}\} \cup \{F_{l\to r}(l) \to r\}$. *Let* $\mathcal{R}_{FP}$ *be the FTRS such that* $\mathcal{R}_{FP} = \{G(x) \to G(F_{l\to r}(x)) \mid l \to r \in \mathcal{R}\} \cup \{G(x) \to x\}$. *Let* $\mathcal{R}_\lambda$ *be the FTRS such that* $\mathcal{R}_\lambda = \bigcup_{l\to r\in\mathcal{R}} \mathcal{R}_{l\to r} \cup \mathcal{R}_{FP}$. *Thus,*

$$\mathcal{R}^*(E) = \mathcal{R}^*_\lambda(E')$$

with $E' = \{G(t)|t \in E\}$.

Proof. – $\mathcal{R}^*(E) \subseteq \mathcal{R}^*_\lambda(E')$: Let $\alpha \in E$ and $\beta \in \mathcal{R}^*(\{\alpha\})$. By Definition of $\mathcal{R}^*$, there exist $l_1 \to r_1, \dots l_n \to r_n \in \mathcal{R}$ and $t_0, \dots, t_n$such that $\alpha \to_{\{l_0\to r_0\}} t_0 \to_{\{l_1\to r_1\}} \dots \to_{\{l_{n-1}\to r_{n-1}\}} t_{n-1} \to_{\{l_n\to r_n\}} t_n = \beta$. The FTRS $\mathcal{R}_{l_0\to r_0}$ fulfills the expected conditions specified in Proposition 1. Consequently, according to Proposition 1, $t_0 \to_{l_0\to r_0} t_1$ implies that $F_{l_0\to r_0}(t_0) \to^*_{\mathcal{R}_{l_0\to r_0}} t_1$. Unfortunately, $F_{l_0\to r_0}(t_0) \notin E'$. However, $t_0 = \alpha$ and $\alpha \in E$. So, $G(t_0) \in E'$. Trivially, $G(t_0) \to_{G(x)\to G(F_{l_0\to r_0}(x))} G(F_{l_0\to r_0}(\alpha))$. Consequently, $G(t_0) \to^*_{\mathcal{R}_\lambda} G(t_1) \to_{G(x)\to x} t_1$ implies that $t_1 \in \mathcal{R}^*_\lambda(E')$. By generalizing the process, one obtains the following rewriting path: $G(t_0) \to^*_{\mathcal{R}_\lambda} G(t_1) \dots \to^*_{\mathcal{R}_\lambda} G(t_n) \to_{G(x)\to x} t_n = \beta$. Consequently, $\beta \in \mathcal{R}^*_\lambda(E')$.
So, one can deduce that

$$\mathcal{R}^*(\{\alpha\}) \subseteq \mathcal{R}^*_\lambda(\{G(\alpha)\}). \tag{18}$$

– $\mathcal{R}^*_\lambda(E') \subseteq \mathcal{R}^*(E)$: Let us study $\mathcal{R}^*_\lambda(\{G(\alpha)\})$.
- Suppose that one rule of the form $G(x) \to G(F_{l\to r}(x))$ is applied. So, $G(\alpha) \to_{G(x)\to G(F_{l\to r}(x))} G(F_{l\to r}(\alpha)))$. Note that $G(x) \to G(F_{l\to r}(x))$ cannot be applied as long as the symbol $F_{l\to r}$ occurs. If $\mathcal{R}^*_{l\to r}(F_{l\to r}(\alpha)) = \emptyset$ then no term of $\mathcal{T}(\mathcal{F}_{bin})$ can be reached from $F_{l\to r}(\alpha)$. Thus, since $\mathcal{R}_{l\to r}$ follows the definition of $\mathcal{R}_\lambda$ in Proposition 1, one can deduce that the rule $l \to r \in \mathcal{R}$ cannot be applied on α.
  If $\mathcal{R}^*_{l\to r}(F_{l\to r}(\alpha)) \neq \emptyset$ then there exists one term of $t_1 \in \mathcal{T}(\mathcal{F}_{bin})$ that can be reached from $F_{l\to r}(\alpha)$. Since $F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{l\to r}} t_1$, one has $G(\alpha) \to^+_{\mathcal{R}_\lambda} G(t_1)$. Moreover, according to Proposition 1, $\alpha \to_{\{l\to r\}} t_1$. Note that $t_1 \in \mathcal{R}^*_\lambda(E')$ since the rule $G(x) \to x$ can be applied on $G(t_1)$. Thus, this process can be iterated and on can build a sequence such that $G(\alpha) \to^+_{\mathcal{R}_\lambda} G(t_1) \to^+_{\mathcal{R}_\lambda} G(t_2) \dots \to^+_{\mathcal{R}_\lambda} G(t_n)$ with $t_i \in \mathcal{R}^*_\lambda(E')$ if the rule $G(x) \to x$ is applied on each term $G(t_i)$ (this is possible since $t_i \in \mathcal{T}(\mathcal{F}_{bin})$). Moreover, according to Proposition 1, one has $t_i \to_{\{l\to r\}} t_{i+1}$ for $i = 1, \dots, n-1$. So one can deduce that $t_n \in \mathcal{R}^*(\{\alpha\})$. Finally, one can deduce that

$$\mathcal{R}^*(\{\alpha\}) \supseteq \mathcal{R}^*_\lambda(\{G(\alpha)\}). \tag{19}$$

- Suppose that no rule of the form $G(x) \to G(F_{l\to r}(x))$ is applied. Consequently, no rule of $\bigcup_{l\to r\in\mathcal{R}} \mathcal{R}_{l\to r}$ can be applied. The unique reachable term of $\mathcal{T}(\mathcal{F}_{bin})$ is then obtained by applying the rules $G(x) \to x$. So, $\alpha \in \mathcal{R}^*_\lambda(\{G(\alpha)\})$. By definition, $\alpha \in \mathcal{R}^*(\{\alpha\})$.

To conclude, for any $\alpha \in E$, from (18) and (19), one obtains that $\mathcal{R}^*(E) = \mathcal{R}^*_\lambda(E')$.

5 Elementary Functional Term Rewriting Systems

In this section, given a TRS $\mathcal{R}$ over $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ and a term $t \in \mathcal{T}(\mathcal{F}_{bin})$, we show that it is possible to compute exactly the same set of reachable terms using a FTRS of a particular form i.e. elementary functional TRS.

Definition 2. *Let $\mathcal{R}_{\lambda^e}$ be a FTRS. $\mathcal{R}_{\lambda^e}$ is said to be elementary (also called EFTRS) iff each rule of $\mathcal{R}_{\lambda^e}$ is of one of the following forms:*

1. $F(a(x, y)) \to \alpha$ *with* $a \in \mathcal{F}_{bin}$, $F \in \mathcal{F}_{NT}$, $x, y \in \mathcal{X}$, $x \neq y$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, X)$ *and* $\mathcal{V}ar(\alpha) = \{x, y\}$
2. $F(\bot) \to \alpha$ *with* $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$
3. $F(a(x, \bot)) \to x$ *with* $x \in \mathcal{X}$, $F \in \mathcal{F}_{NT}$ *and* $a \in \mathcal{F}_{bin}$.

Actually, EFTRSs, presented in Definition 2, are expressive enough for the specification of the whole rewriting mechanism for a rule $l \to r \in \mathcal{R}$: 1. to find a position for rewriting, 2. to check if l matches with the current subterm, then 3. to compute the resulting substitution σ, and 4. to replace the subterm by $r\sigma$.

Consequently, one has the following result.

Proposition 2. *Let $\mathcal{R}$ be a TRS over $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$, $\beta \in \mathcal{T}(\mathcal{F}_{bin})$ and $l \to r \in \mathcal{R}$. There exists an EFTRS $\mathcal{R}_{\lambda^e}$ such that*

$$\alpha \to_{\{l \to r\}} \beta \Leftrightarrow F_{l \to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta.$$

Proof (Sketched proof).

The key point is to construct an EFTRS $\mathcal{R}_{\lambda^e}$ implementing the whole rewriting process. The EFTRS $\mathcal{R}_{\lambda^e}$ is build as follows: $\mathcal{R}_{\lambda^e} = \mathcal{R}^{l\to r}_{visit} \cup \mathcal{R}^{l\to r}_{TV} \cup \mathcal{R}^{l\to r}_{check} \cup \mathcal{R}^{l\to r}_{\sigma} \cup \mathcal{R}^{l\to r}_{GS} \cup \mathcal{R}^{l\to r}_{\sigma-apply}$

where

- $\mathcal{R}^{l\to r}_{visit} = \{F_{l\to r}(a(x, y)) \to a(F_{l\to r}(x), y),\ F_{l\to r}(a(x, y)) \to a(x, F_{l\to r}(y))\}$
- $\mathcal{R}^{l\to r}_{TV} = \{F_{l\to r}(x) \to F^{l\to r}_{rewrite}(Check_{l\to r}(F^{l\to r}_{copy}(\ F^{\varepsilon}_{l\to r}(x))))\}$.
- $\mathcal{R}^{l\to r}_{check}$ is an EFTRS allowing us to check if a term of $\mathcal{T}(\mathcal{F}_{bin})$ *matches* with l. The application of this EFTRS is fired by the presence of the non-terminal symbol $F^{\varepsilon}_{l\to r}$. Given a term $t \in \mathcal{T}(\mathcal{F}_{bin})$ matching with l, $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\})$ is $\{t'\}$ where
  - for all $p \in \mathcal{P}os_{\mathcal{F}}(l)$, $t(p) = t'(p)$;
  - for all $p \in \mathcal{P}os_{\mathcal{X}}(l)$, $t'(p) = \oplus_{x,p} \in \mathcal{F}_{bin}$;
  - for all $p \in \mathcal{FP}\text{os}(l) \backslash \mathcal{P}os_{\mathcal{X}}(l)$, and for all $p' \in \mathcal{P}os(t|_p)$, $t(p.p') = t'(p.1.p')$.

  The symbol $\oplus_{x,p}$ is a marker meaning that the term under this marker is the value that the variable x at position p in l takes when matching occurs. When the matching between t and l is not possible, $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\}) = \emptyset$. Note that, at this point, we do not check the case that two identical variables have to share the same value. So $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\}) = \emptyset$ iff t and l are different structurally speaking.

- $\mathcal{R}_{\sigma}^{l\to r}$ is an EFTRS allowing us to construct an ordered list of terms indexed by variables of l. This list is represented by a term and semantically speaking, one can see this list as the substitution resulting from the *matching* step. Let t' be the term resulting from $(\mathcal{R}_{check}^{l\to r})^*(\{F_{l\to r}^{\varepsilon}(t)\})$. $(\mathcal{R}_{\sigma}^{l\to r})^*(F_{copy}^{l\to r}(t'))$ leads to a unique term t'' representing the substitution resulting from the *matching* between t and l.
- $\mathcal{R}_{GS}^{l\to r}$ is an EFTRS specifying the verification of a well-formed substitution. Indeed, if l is not linear then a variable x appears two times at least. If all occurrences of x share the same term as value then the substitution is well-formed. $(\mathcal{R}_{check}^{l\to r})^*(Check_{l\to r}(t'')) = \{t''\}$ is the substitution is well-formed, $\emptyset$ otherwise. $\mathcal{R}_{GS}^{l\to r}$ handles each variable occurring more than once as mentionned above.
- $\mathcal{R}_{\sigma-apply}^{l\to r}$ is an EFTRS specifying the application of the well-formed substitution resulting from t and l on the term r. Thus, $(\mathcal{R}_{\sigma-apply}^{l\to r})^*(F_{rewrite}^{l\to r}(t'')) = \{r\sigma\}$.

The EFTRSs listed above are fully defined in Appendix.

- $\alpha \to_{\{l\to r\}} \beta \Rightarrow F_{l\to r}(\alpha) \to_{\mathcal{R}_{\lambda^e}}^* \beta$: Trivially, if $\alpha \to_{\{l\to r\}} \beta$ then there exist a position p of α and a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $\alpha|_p = l\sigma$ and $\beta = \alpha[r\sigma]_p$. From the definition of $\mathcal{R}_{\lambda^e}$, one can deduce that $\mathcal{F}_{l\to r}(\alpha) \to_{\mathcal{R}_{\lambda^e}}^* \alpha[F_{l\to r}(\alpha|_p)]_p$. Thus, applying the rule $F_{l\to r}(x) \to F_{rewrite}^{l\to r}(Check_{l\to r}(F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(x))))$ on $\alpha[F_{l\to r}(\alpha|_p)]_p$, one obtains $\alpha[F_{rewrite}^{l\to r}(Check_{l\to r}($ $F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(\alpha|_p))))]_p$. Since σ is a substitution from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}_{bin})$, and according to the set of rules $\mathcal{R}_{TV}^{l\to r} \cup \mathcal{R}_{check}^{l\to r} \cup \mathcal{R}_{\sigma}^{l\to r} \cup \mathcal{R}_{GS}^{l\to r}$, $\alpha[F_{rewrite}^{l\to r}(Check_{l\to r}(F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(\alpha|_p))))]_p$ can be rewritten in $\alpha[F_{rewrite}^{l\to r}(t)]_p$ where t is defined as follows:
  - $t(p) = \sigma(x_i)$, $t(p') = \top_{\mathcal{X}} \in \mathcal{F}_{bin}$ with $p' = (2.)^i\ p = p'.1$ and x_i is the i^{th} variable of l read from the left of l
  - $t(p_{rmp}) = \bot_{\mathcal{X}} \in \mathcal{F}_{bin}$ with $p_{rmp} = (2.)^m 2$ and m is the number of variables occurring in l

  Thus, the term t represents as a term the substitution σ. Finally, the EFTRS $\mathcal{R}_{\sigma-apply}^{l\to r}$ allows us to reconstruct r and the value of variables are copied symbol per symbol. The unique reachable terms is then $r\sigma$. So, to summarize, $F_{rewrite}^{l\to r}(t) \to_{\mathcal{R}_{\sigma-apply}^{l\to r}}^* r\sigma \in \mathcal{T}(\mathcal{F}_{bin})$. Consequently, $\alpha[F_{rewrite}^{l\to r}(t)]_p \to_{\mathcal{R}_{\sigma-apply}^{l\to r}}^* \alpha[r\sigma]_p \in \mathcal{T}(\mathcal{F}_{bin})$ and since $\alpha[r\sigma]_p = \beta$, $\beta \in \mathcal{R}_{\lambda^e}^*(\{F_{l\to r}(\alpha)\})$.
- $\alpha \to_{l\to r} \beta \Leftarrow F_{l\to r}(\alpha) \to_{\mathcal{R}_{\lambda^e}}^* \beta$: The proof for this case is close to the similar case handled in the proof of Proposition 1 in the sense that rewrite steps are ordered and depending on the initial term i.e. $F_{l\to r}(\alpha)$. Eventually, since by hypothesis $F_{l\to r}(\alpha) \to_{\mathcal{R}_{\lambda^e}}^* \beta$, due to the definitions of $\mathcal{R}_{visit}^{l\to r}$ and $\mathcal{R}_{TV}^{l\to r}$, there exists a position $p \in \mathcal{P}os(\alpha)$ such that $F_{l\to r}(\alpha) \to_{\mathcal{R}_{visit}^{l\to r} \cup \mathcal{R}_{TV}^{l\to r}}^* \alpha[F_{rewrite}^{l\to r}(Check_{l\to r}(F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(\alpha|_p))))]_p$ and $F_{rewrite}^{l\to r}(Check_{l\to r}(F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(\alpha|_p)))) \to_{\mathcal{R}_{\lambda^e}}^* \beta$. Due to the stack of symbols of $\mathcal{F}_{NT}$ over $\alpha|_p$, one can show that $F_{rewrite}^{l\to r}(Check_{l\to r}(F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(\alpha|_p))))$ $\beta \Leftrightarrow (\mathcal{R}_{\sigma-apply}^{l\to r})^*((\mathcal{R}_{GS}^{l\to r})^*((\mathcal{R}_{\sigma}^{l\to r})^*((\mathcal{R}_{check}^{l\to r})^*(F_{rewrite}^{l\to r}(\ Check_{l\to r}(F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(\alpha|_p))))))))) =$ $\{\beta\}$. Consequently, according to definitions of $\mathcal{R}_{\sigma}^{l\to r}$, $\mathcal{R}_{GS}^{l\to r}$ and $\mathcal{R}_{check}^{l\to r}$, for a term $t \in \mathcal{T}(\mathcal{F}_{bin})$, one can show that if $(\mathcal{R}_{check}^{l\to r})^*(\{Check_{l\to r}(F_{copy}^{l\to r}(F_{l\to r}^{\varepsilon}(t)))\}) \neq$

$\emptyset$ then there exists a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $l\sigma = t$. Applying such a result to our study case, there exists $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $l\sigma = \alpha|_p$. Not only $(\mathcal{R}^{l\to r}_{check})^*(\{Check_{l\to r}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(t)))\}) \neq \emptyset$, but there exists also a term t' of $\mathcal{T}(\mathcal{F}_{bin})$ such that $(\mathcal{R}^{l\to r}_{check})^*(\{Check_{l\to r}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(t)))\}) = \{t'\}$. The term t' is a representation of σ, and applying the EFTRS $\mathcal{R}^{l\to r}_{\sigma-apply}$ on t' allows us to reconstruct r by substituting each variable x_i of r by $\sigma(x_i)$. The unique reachable term from t' is then $r\sigma$. So, to summarize, there exist a position of α and a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ such that $\alpha|_p = l\sigma$, $\beta = \alpha[r\sigma]_p$ and $F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta$. One can deduce that $\alpha \to_{l\to r} \beta \Rightarrow F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta$ which concludes the proof.

Trivially, one obtains the more general result presented below.

Theorem 2. *Let $\mathcal{R}$ be a TRS over $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ and $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$. Thus, there exists a EFTRS $\mathcal{R}_{\lambda^e}$ and a symbol $G \in \mathcal{F}_{NT}$ such that:*

$$\mathcal{R}^*(E) = \mathcal{R}^{\star}_{\lambda^e}(E'),$$

where $E' = \{G(\alpha) | \alpha \in E\}$.

Proof. Considering the EFTRS $\mathcal{R}_{\lambda^e} = \bigcup_{l\to r\in\mathcal{R}}(\mathcal{R}^{l\to r}_{visit} \cup \mathcal{R}^{l\to r}_{TV} \cup \mathcal{R}^{l\to r}_{check} \cup \mathcal{R}^{l\to r}_{\sigma} \cup \mathcal{R}^{l\to r}_{GS} \cup \mathcal{R}^{l\to r}_{\sigma-apply}) \cup \mathcal{R}_{FP}$ with $\mathcal{R}_{FP} = \{G(a(x,y)) \to G(F_{l\to r}(a(x,y))) \mid l \to r \in \mathcal{R} \wedge a \in \mathcal{F}_{bin}\} \cup \{G(a(x,y)) \to a(x,y) \mid a \in \mathcal{F}_{bin}\}$, the proof is very close to the one of Theorem 1.

6 Specification with EFTRS and Benchmarks

In Section 6.1, we present how the Tree Arbiter Protocol is encoded using EFTRS and Section 6.2 describes how the verification of the property ϕ defined in Section 3 can be entirely encoded in EFTRS. Finally, benchmarks are presented in Section 6.3.

6.1 Specification TAP using EFTRSs

Let $\mathcal{F}_{NT}$ be the set of non-terminal symbols such that $\mathcal{F}_{NT} = \{H, R, RT, TI, I, Arbiter\}$. The set $\mathcal{F}_{bin}$ is the one defined in Section 3, i.e., $\{i, r, t, b, \bot\}$.

The rewriting rules (1), (2) and (3) given in Section 3 are translated as follows in EFTRS rules.

$$H(i(x,y)) \to r(x, R(y)) \tag{20}$$
$$H(i(x,y)) \to r(R(x), y) \tag{21}$$
$$R(r(x,y)) \to r(x,y) \tag{22}$$
$$R(\bot) \to \bot \tag{23}$$

Let us illustrate the rewriting of the following configuration $t_0 = H(t(i(\bot, \bot), i(\bot, \bot)))$. None of rules mentioned above can be applied. So, no term of $\mathcal{T}(\mathcal{F}_{bin})$ is reachable from $H(t(i(\bot, \bot), i(\bot, \bot)))$.

So we need rules for specifying the circulation of the symbol H before applying the rules (20), ...,(23). These rules are given below and are defined for each symbol of $\mathcal{F}_{bin}$ except $\bot$.

$$H(i(x,y)) \to i(H(x),y) \tag{24}$$
$$H(i(x,y)) \to i(x,H(y)) \tag{25}$$
$$H(r(x,y)) \to r(H(x),y) \tag{26}$$
$$H(r(x,y)) \to r(x,H(y)) \tag{27}$$
$$H(b(x,y)) \to b(H(x),y) \tag{28}$$
$$H(b(x,y)) \to b(x,H(y)) \tag{29}$$
$$H(t(x,y)) \to t(H(x),y) \tag{30}$$
$$H(t(x,y)) \to t(x,H(y)) \tag{31}$$

Now, let us illustrate the use of the rules (24),..., (31). Let us apply the rule (24) (resp. (25)) on t_0. So, the term $t'_1 = t(H(i(\bot,\bot)), i(\bot,\bot))$ (resp. $t''_1 = t(i(\bot,\bot), H(i(\bot,\bot)))$) is obtained. By applying rule (20) and rule (23) on t'_1 (resp. t''_1), one obtains the term $t'_2 = t(r(\bot,\bot), i(\bot,\bot))$ (resp. $t''_2 = t(i(\bot,\bot), r(\bot,\bot))$).

Note that t'_2 and t''_2 do not contain symbol of $\mathcal{F}_{NT}$. So the EFTRS can not be applied anymore. Thus, we introduce the symbol *Arbiter* whose semantics is described by rules (42) to (45). In other words, this symbol allows us to rewrite as much as needed. The rules below fill out the specification of TAP by an EFTRS.

$$H(t(x,y)) \to b(x,RT(y)) \tag{32}$$
$$H(t(x,y)) \to b(RT(x),y) \tag{33}$$
$$RT(r(x,y)) \to t(x,y) \tag{34}$$
$$H(b(x,y)) \to t(x,TI(y)) \tag{35}$$
$$H(b(x,y)) \to t(TI(x),y) \tag{36}$$
$$TI(t(x,y)) \to i(I(x),I(y)) \tag{37}$$
$$TI(t(x,y)) \to r(R(x),I(y)) \tag{38}$$
$$TI(t(x,y)) \to r(I(x),R(y)) \tag{39}$$
$$I(i(x,y)) \to i(x,y) \tag{40}$$
$$I(\bot) \to \bot \tag{41}$$
$$Arbiter(t(x,y)) \to t(x,y) \tag{42}$$
$$Arbiter(t(x,y)) \to Arbiter(H(t(x,y))) \tag{43}$$
$$Arbiter(b(x,y)) \to b(x,y) \tag{44}$$
$$Arbiter(b(x,y)) \to Arbiter(H(b(x,y))) \tag{45}$$

Now, consider the starting term $t_0 = Arbiter(t(i(\bot,\bot), i(\bot,\bot)))$. From t_0 and applying the rules (43), (24), (20) and (23), the term $t_1 = Arbiter(t(i(\bot,\bot), r(\bot,\bot)))$ is computed. Applying the rules (32) and (34) from t_1, the term t_2 is computed with $t_2 = Arbiter(b(i(\bot,\bot), t(\bot,\bot)))$. Note that from t_1 (resp. t_2), we can also apply the rule (42) (resp. 44) and then obtain the term $t(i(\bot,\bot), r(\bot,\bot))$ (resp. $b(i(\bot,\bot), t(\bot,\bot))$). The two terms mentioned above represent two configurations actually reachable.

We note $\mathcal{R}_{\lambda^e}^{TAP}$ the whole EFTRS defined in this section.

6.2 Invariant Verification of TAP in Functional Reachability

In Section 3, we have defined the following property ϕ: there is always only one token circulating in the hierarchy tree. Let us consider $\neg\phi$. In words, $\neg\phi$ specifies that there may exists one configuration containing zero token or two tokens at least.

The following EFTRS $\mathcal{R}_{\lambda^e}^{\neg\phi}$ specifies the property $\neg\phi$.

$$No\_t(r(x,y)) \rightarrow r(No\_t(x), No\_t(y)) \quad (46)$$
$$No\_t(b(x,y)) \rightarrow b(No\_t(x), No\_t(y)) \quad (47)$$
$$No\_t(i(x,y)) \rightarrow i(No\_t(x), No\_t(y)) \quad (48)$$
$$No\_t(\bot) \rightarrow \bot \quad (49)$$
$$Two\_t(r(x,y)) \rightarrow r(Two\_t(x), y) \quad (50)$$
$$Two\_t(r(x,y)) \rightarrow r(x, Two\_t(y)) \quad (51)$$
$$Two\_t(b(x,y)) \rightarrow b(Two\_t(x), y) \quad (52)$$
$$Two\_t(b(x,y)) \rightarrow b(x, Two\_t(y)) \quad (53)$$
$$Two\_t(i(x,y)) \rightarrow i(Two\_t(x), y) \quad (54)$$
$$Two\_t(i(x,y)) \rightarrow i(x, Two\_t(y)) \quad (55)$$
$$Two\_t(t(x,y)) \rightarrow i(x, Two\_t(y)) \quad (56)$$

Semantically speaking, the symbol $No\_t \in \mathcal{F}_{NT}$ is introduced for checking that a term does not contain the symbol t. Indeed, there is no rule of the form $No\_t(t(x,y)) \rightarrow \beta$. Consequently, for a term $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$ for which there exists $p \in \mathcal{P}os(\alpha)$ such that $\alpha(p) = t$, $(\mathcal{R}_{\lambda^e}^{\neg\phi})^*(\{No\_t(\alpha)\}) = \emptyset$.

The symbol $Two\_t \in \mathcal{F}_{NT}$ is introduced for checking that the symbol t occurs in a term one time at least. So, for a term $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$,

- $(\mathcal{R}_{\lambda^e}^{\neg\phi})^*(\{Two\_t(Two\_t(\alpha))\}) = \emptyset$ means that the symbol t occurs in α only once or never;
- $(\mathcal{R}_{\lambda^e}^{\neg\phi})^*(\{Two\_t(Two\_t(\alpha))\}) \neq \emptyset$ means that the symbol t occurs in α at least two time.

Consequently,

the TAP satisfies $\phi \Leftrightarrow (\mathcal{R}_{\lambda^e}^{\neg\phi})^*((\mathcal{R}_{\lambda^e}^{TAP})^*(\{Two\_t(Two\_t(t'_0)), No\_t(t'_0)\})) = \emptyset$

where $t'_0 = Arbiter(t_0)$ and t_0 is the initial configuration of the TAP.

Using our tool, we have been able to check that ϕ is true for the TAP handling 2^{400} processors in 2.933s.

6.3 Some Experiments

Hereafter reported results have been obtained on a PC 1.7GHz 1GB RAM. We have compared our tool with Timbuk [31, 27] and Maude [19]. Timbuk is a collection of tools for achieving proofs of reachability over TRSs. This tool manipulates tree automata for representing possibly infinite set of terms. Maude is rewrite engine which is often used in a verification context. The table below summarizes

some results obtained on well-known protocols i.e. Tree Arbiter Protocol (TAP), Percolate Protocol (PP) and Leader Election Protocol (LEP). For this experiments, we have adapted local fixpoint techniques in our EFTRS framework. Of course, acceleration techniques are well adapted for this kind of protocols where the number of configurations may be huge but finite. The acceleration techniques can be implemented automatically for all of these study cases. Thus, it explains the difference in terms of computation times between tools. However, Maude and Timbuk can be used for the verification of infinite systems [23, 28, 29, 41, 5]. For now, our technique is exclusively defined for finite systems (and thus terminating TRS).

In the following table, Σ denotes the study case, N the number of processors at the bottom of the trees and Confs the size of the state space to explore. The number of configurations is evaluated with a Java program, thus, "?" denotes that the library *java.math.BigDecimal* we used cannot return the result within three hours. For further details of model descriptions and the benchmarks, see [4].

Σ	N	Confs	TRS		EFTRS
			Timbuk	Maude	Our Tool
TAP	2^5	8.539 E+8	$> 3h$	$> 3h$	0.109
	2^{10}	1.989 E+273	-	-	0.196
	2^{20}	5.350 E+278807	-	-	0.256
	2^{400}	?	-	-	2.866
PP	2^5	11047	0.138	1.479	0.018
	2^{10}	8.445 E+135	24.179	$> 3h$	0.053
	2^{20}	4.508 E+139402	$> 3h$	-	0.102
	2^{500}	?	-	-	0.396
LEP	2^5	16	$> 3h$	0.457	0.093
	2^{10}	512	-	$> 3h$	0.159
	2^{20}	5.243 E+5	-	-	0.256
	2^{400}	1.291 E+120	-	-	138.699

7 Applications of EFTRS

7.1 Application on Petri net and Experimental results

First, we show how to simulate some Petri net models (presented in [15, 14, 11]) such as Round-Robin Mutex Protocol, Slotted Ring Protocol, and the Dining Philosophers problem on EFTRSs. Then we compare our EFTRS Model-checker (written in JAVA) with some available tools like SPIN (downloadable from http://spinroot.com/spin/whatispin.html), NuSMV [17] (downloadable from http://nusmv.irst.itc.it/), SMART (http://www.cs.ucr.edu/ ciardo/SMART/), DDD, SDD and HSDD (downloadable from http://move.lip6.fr/ software/DDD/, http://sourceforge.net/projects/buddy/, http://vlsi.colorado.edu/ fabio/CUDD/).

Dining philosophers problem is an illustrative example of a common computing problem in concurrency. This classic model is obtained by connecting N

identical submodels (see Figure 2), one per philosopher, in a circular fashion. Each philosopher starts in the idle state and occasionally decides to eat: to do so he must acquire both his left and right forks, which he then releases when he has finished eating.

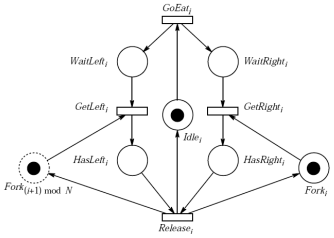

Fig. 2: Petri subnet P_i

On the first phase, construct a term for sub-net level: $local(u, v)$ where u and v is the encoding of the identical model, e.g. $local(\bot, hasL(false(\bot, \bot), hasR(true(\bot, \bot), ...)))$. On the second phase, construct a hierarchical model by combining two sub-nets like $sys(local(u, v), local(u, v))$. From this *1-level* model, we can construct *2-level* by the same way, and so forth.

There is relations between two neighbours. So we have a synchronization operator by going down from root to sub-net levels:
$H(sys(x, y)) \rightarrow SyncIIPP(sys(x, y))$
$H(sys(x, y)) \rightarrow Sync1N(sys(x, y))$
$Sync1(sys(x, y)) \rightarrow sys(Sync1(x), y)$
$SyncN(sys(x, y)) \rightarrow sys(x, SyncN(y))$

Go down from *k-level* to *k-1-level* of model:
$SyncIIPP(sys(x, y)) \rightarrow sys(SyncIIPP(x), SyncIIPP(y))$

Then at each level of the model:
$SyncI(sys(x, y)) \rightarrow sys(x, SyncI(y))$
$SyncIPP(sys(x, y)) \rightarrow sys(SyncIPP(x), y)$

Finally, at $sub - net$ levels of the model:

$$Sync1(local(x, y)) \rightarrow \alpha_1(x, y)\ \ SyncN(local(x, y)) \rightarrow \alpha_N(x, y)$$
$$SyncI(local(x, y)) \rightarrow \beta_I(x, y)\ \ SyncIPP(local(x, y)) \rightarrow \beta_{IPP}(x, y)$$

In the other hand, at $sub - net$ levels of the model:
$Local(local(x, y)) \rightarrow \gamma(x, y)$

where α, β, γ are compositions of some primitive rules $Post_p$ and Pre_p for each place p:

$$Post_p(p(x, y)) \rightarrow p(Post(x), y)\ \ Pre(true(x, y)) \rightarrow false(x, y)$$
$$Pre_p(p(x, y)) \rightarrow p(Pre(x), y) \quad Post(false(x, y)) \rightarrow true(x, y)$$

Our implementation shows that our automatical Petri-EFTRS transformation gives the same configurations number of some Petri net model with other approachs.

The following experiments deal with Petri Net problems. We also compare our tool with some tools like SPIN [35], NuSMV [17], SMART [10, 15, 11] and HSDD [46, 47]. These values concerning the other tools are directly taken from [16, 9, 22, 46, 47]. Some missing results are indicated by "N/A".

N	Confs	SPIN	SMV	SMART	HSDD	EFTRS
Dining Philosophers.						
2^5	1.15E+20	N/A	0.4	0.01	0.00	0.65
2^{10}	1.02E+642	N/A	-	1.8	0.00	1.25
2^{15}	2.09E+20544	N/A	-	65.5	0.00	1.96
2^{20}	1.8E+657418	N/A	-	-	0.01	2.77
2^{35}	?	N/A	-	-	0.02	135.4
Slotted Ring Protocol.						
2^2	5136	0.0	0.0	0.0	0.0	1.13
2^3	6.80E+7	8.2	0.13	0.06	0.0	18.53
2^4	1.65E+16	-	2853	0.18	0.03	2398
Round Robin Mutex Protocol [32].						
2^4	2.359E+6	43.0	0.34	0.01	0.0	3.316
2^6	2.656E+21	-	11.7	0.09	0.2	30.91
2^8	6.696E+79	-	-	7.04	1.0	1115

1. **Dining philosophers problem.** The hierarchical model can resolve problem having size until $N = 2^{35}$.

N	#Confs	EFTRS
2^5	1.155 E+20	0.654
2^{10}	1.023 E+642	1.251
2^{15}	2.094 E+20544	1.966
2^{20}	1.868 E+657418	2.779
2^{25}	?	3.661
2^{30}	?	4.782
2^{35}	?	135.407

We can also compare model size N with some related results [1]:

EFTRS	SPIN	NuSMV	SDD	SMART	HSDD
2^{35}	-	200	5000	5000	2^{20000}

2. **Slotted Ring Protocol.** The hierarchical model can resolve problem having size until $N = 2^4$.

N	#Confs	EFTRS
2^2	5136	1.131
2^3	6.802 E+7	18.537
2^4	1.651 E+16	2398.754

[1] Some are missing as indicated by "-".

We can also compare model size N with some related results:

EFTRS	SPIN	NuSMV	SDD	SMART	HSDD
2^4	6	15	50	300	200

3. **Round Robin Mutex Protocol** (presented in [32]).
   This protocol solves a specific type of mutual exclusion problem among N processes organized in a circular fashion, requiring access to a shared resource.

N	#Confs	EFTRS
2^1	18	1.719
2^2	144	1.602
2^3	4608	2.108
2^4	2.359 E+ 6	3.316
2^5	3.092 E+ 11	8.003
2^6	2.656 E+ 21	30.914
2^7	9.800 E+ 40	171.321
2^8	6.696 E+ 79	1115.704

The hierarchical model can resolve problem having size until $N = 2^8$. We can also compare model size N with some related results [2]:

EFTRS	SPIN	NuSMV	SDD	SMART	HSDD
2^8	16	60	-	1100	1000

Saturation approach (presented by Ciardo et al. in [13] for MDD, a BDD-like structure). This technique is also called local fixed point (LFP) technique which fights the intermediate peak size effect in the structure liked-BDD as DDD [21], SDD [22], HSDD [46, 45].

7.2 Application of EFTRS on LTL model-checking

[48, 20] show us how transform a LTL formula to Büchi Automata. The problem now is to encode a Büchi automaton into FTRS (or EFTRS) like $buchi(u, v)$ then construct the synchronized product of Büchi automaton and the system is $ltl(buchi(u, v), sys(w, t))$ where $sys(w, t))$ is an encoding of system and $u, v, w, t \in \mathcal{F}_{bin}$.

Operations on the synchronized product is defined for each property φ as the following: $Sync_\varphi(ltl(buchi(u, v), sys(w, t))) \rightarrow ltl(Buchi_\varphi(buchi(u, v)), Sys_\varphi(sys(w, t)))$ where

- $Buchi_\varphi$ is $Change_{s \xrightarrow{\varphi} s'}(s(x, y)) \rightarrow s'(x, y)$
- Sys_φ is a composition of some operations on the system: $Succ$ compute the next sytem states and $Test_\varphi$ works as a filter keeping only states satifying φ.

[2] These values were not measured by us, but are directly taken from [16, 22, 9].

In Dining philosophers problem, since forks can be acquired in any order, this model is known to have two deadlock states, the one where each philosopher has acquired his left fork and waits for his right fork (which, being also the left fork of his right neighbor, will never be released), and the symmetric state where each philosopher has acquired his right fork.

Thus, we are interested in the behavior such as: *In a moment in the future, the first philosopher has the left but he never has the right* by using the LTL formula $F_1 = F(p_1.hasL \wedge G(\neg p_1.hasR))$ (See Büchi automaton in Figure 3a).

We are also interested in the behavior such as: *First the first philosopher has the right then put this fork and finally his neighbor has the left* by using the LTL formula $F_2 = (p_1.hasR \cup (p_1.Fork \cup p_N.hasL))$. See this Büchi automaton in Figure 3b.

An first implementation of LTL model-checker based on EFTRS of these LTL formulas is in the below table.

N	$\#Confs_{F_1}$	F_1	$\#Confs_{F_2}$	F_2
2^3	28657	3.139	1.036 E+5	1.777
2^4	2.971 E+9	46.584	1.074 E+10	8.936
2^5	3.194 E+19	5486.060	1.155 E+20	251.871

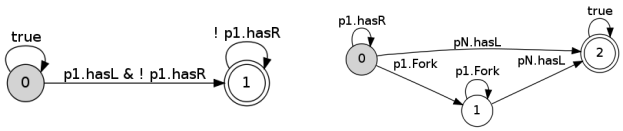

Fig. 3: Büchi automatons a) F_1 and b) F_2

8 Conclusion

In this paper, we have introduced a new formalism, so called FTRS, for the specification of transition relation for finite system verification. We have shown that the expressiveness of FTRSs and EFTRSs is the same as classical FTRSs. By adapting acceleration techniques for our new formalism and for the protocol to study, the computation times are really convincing. Moreover, the exploration of huge state spaces has been possible with our tool when some others fail. Some experiments have been carried out in the field of Petri nets. Actually, our results are not so good as expected, but we plan to refine the transformation Petri nets towards EFTRS [4] in a couple of months.

A first step towards the verification of invariants on finite systems has been initiated also in this paper. Some investigations are carried out on the field of the LTL model-checking. And it would look like possible to express the whole LTL model-checking with FTRS [4].

References

1. P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Tree regular model checking: A simulation-based approach. *The Journal of Logic and Algebraic Programming*, 2006.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
3. R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E.Macii, A.Pardo, and F.Somenzi. Algebraic decision diagrams and their applications. In *ICCAD'93*, volume 2860, pages 188–191, 1993.
4. Y. Boichut, J-M Couvreur, and D-T Nguyen. Functional term rewriting systems. Research report, LIFO, 2010.
   `http://eftrs.svn.sourceforge.net/viewvc/eftrs/`.
5. Y. Boichut, T. Genet, T. Jensen, and L. Le Roux. Rewriting Approximations for Fast Prototyping of Static Analyzers. In *Proceedings of the 18th Conference on Rewriting Techniques and Applications*, volume 4533 of *LNCS*, pages 48–62, 2007.
6. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on computers*, C-35(8):677–691, August 1986.
7. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.
8. J.R. Burch, E.M. Clarke, and K.L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation (Special issue for best papers from LICS90)*, 98(2):153–181, 1992.
9. G. Ciardo. Reachability set generation for petri nets: Can brute force be smart? In *ICATPN*, pages 17–34, 2004.
10. G. Ciardo, R. L. Jones III, Marmorstein R. M., A. S. Miner, and R. Siminiceanu. Smart: Stochastic model-checking analyzer for reliability and timing. In *DSN*, page 545, 2002.
    `http://www.cs.ucr.edu/ ciardo/SMART/`.
11. G. Ciardo, G. Lüttgen, and A. S. Miner. Exploiting interleaving semantics in symbolic state-space generation. *Formal Methods in System Design*, 31(1):63–100, 2007.
12. G. Ciardo, G. Lüttgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In *ICATPN'2000*, volume 1825 of *LNCS*, pages 103–122. Springer Verlag, 2000.
13. G. Ciardo, R. Marmorstein, and R. Siminiceanu. The saturation algorithm for symbolic state-space exploration. *Int. J. Softw. Tools Technol. Transf.*, 8(1):4–25, 2006.
14. G. Ciardo, R. M. Marmorstein, and R. Siminiceanu. Saturation unbound. In *TACAS*, pages 379–393, 2003.
15. G. Ciardo and R. Siminiceanu. Using edge-valued decision diagrams for symbolic generation of shortest paths. In *FMCAD '02: Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, pages 256–273, London, UK, 2002. Springer-Verlag.
16. G. Ciardo and A. J. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *CHARME*, pages 146–161, 2005.
17. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic

Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.
`http://nusmv.irst.itc.it/`.
18. E. Clarke, D. Long, and K. McMillan. Compositional model checking. In *Proceedings of the Fourth Annual Symposium on Logic in computer science*, pages 353–362, Piscataway, NJ, USA, 1989. IEEE Press.
19. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001.
`http://maude.cs.uiuc.edu`.
20. J.-M. Couvreur. On-the-fly verification of linear temporal logic. In Springer Verlag, editor, *Proc. of FM'99*, pages 253–271, 1999. Lecture Notes in Computer Science.
21. J-M. Couvreur, E. Encrenaz, E. PaviotAdet, D. Poitrenaud, and P. Wacrenier. Data decision diagram for petri net analysis. In *ICATPN*, volume 2360, pages 1–101. Springer Verlag, 2002.
22. J-M. Couvreur and Y. Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In *FORTE*, pages 443–457, 2005.
23. G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *Proc. 2nd WRLA Workshop, Pont à Mousson (France)*, 1998.
24. N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.
25. D. L. Dill. *Trace theory for automatic hierarchical verification of speed-independent circuits*. MIT Press, Cambridge, MA, USA, 1989.
26. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In Fabio Gadducci and Ugo Montanari, editors, *Fourth Workshop on Rewriting Logic and its Applications, WRLA '02*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.
27. G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *JAR*, 33 (3-4):341–383, 2004.
28. T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. 17th CADE Conf., Pittsburgh (Pen., USA)*, volume 1831 of *LNAI*. Springer-Verlag, 2000.
29. T. Genet, Y.-M. Tang-Talpin, and V. Viet Triem Tong. Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems. In *In Proceedings of Workshop on Issues in the Theory of Security*, 2003.
30. T. Genet and V. Viet Triem Tong. Reachability analysis of term rewriting systems with timbuk. In *In: LPAR Proceedings. (2001*, pages 691–702. Springer-Verlag, 2001.
`http://www.irisa.fr/celtique/genet/timbuk/`.
31. T. Genet and V. Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with *timbuk*. In *Proc. 8th LPAR Conf., Havana (Cuba)*, volume 2250 of *LNAI*, pages 691–702. Springer-Verlag, 2001.
`http://www.irisa.fr/celtique/genet/timbuk/`.
32. S. Graf, B. Steffen, and G. Lüttgen. Compositional minimisation of finite state systems using interface specifications. *Formal Asp. Comput.*, 8(5):607–616, 1996.
33. A. Gupta. *Inductive Boolean Function Manipulation*. PhD thesis, Carnegie Mellon University, 1994.
34. A. Gupta and A. L. Fisher. Representation and symbolic manipulation of linearly inductive boolean functions. In *ICCAD'93*, pages 111–116, 1993.

35. G.J. Holzmann. The model checker SPIN. *IEEE Trancastions on Software Engineering*, 23(5), may 1986.
`http://spinroot.com/spin/whatispin.html`.
36. H. Hulgaard, P. F. Williams, and H. R. Andersen. Equivalence checking of combinational circuits using boolean expression diagrams. *IEEE Transactions of Computer-Aided Design*, 18(7), 1999.
37. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *THEORETICAL COMPUTER SCIENCE*, pages 424–435. Springer, 1997.
38. T. Kolks, B. Lin, and H. De Man. Sizing and verification of communication buffers for communicating processes. In *ICCAD'93*, volume 1825, pages 660–664, 1993.
39. L. Mauborgne. Binary decision graphs. In *SAS'99*, volume 1694 of *LNCS*, pages 101–116. Springer-Verlag, 1999.
40. L. Mauborgne. An incremental unique representation for regular trees. *Nordic Journal of Computing*, 7(4):290–311, 2000.
41. J. Meseguer, M. Palomino, and N. Mart-Oliet. Equational Abstractions. In *Proc. 19th CADE Conf., Miami Beach (Fl., USA)*, volume 2741 of *LNCS*, pages 2–16. Springer, 2003.
42. S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagrams with attributed edges for efficient boolean function manipulation. In *DAC'90*, pages 52–57. ACM/IEEE, IEEE Computer Society Press, 1990.
43. A.S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *ICATPN'99*, volume 1639 of *LNCS*, pages 6–25. Springer Verlag, 1999.
44. F. Reffel. BDD-Nodes Can Be More Expressive. In *ASIAN'99*, volume 1742 of *LNCS*, pages 294–307. Springer Verlag, 1999.
45. Y. Thierry-Mieg. Techniques pour le model-checking de spécifications de haut niveau. *Thèse de doctorat en Informatique, LIP6, Univ. Paris VI*, 2004.
46. Y. Thierry-Mieg, A. Hamez, and F. Kordon. Building efficient model checkers using hierarchical set decidion diagrams and automatic saturation. *Fundamenta Inforaticae Petri Nets*, 1-25, 2008.
47. Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon. Hierarchical set decision diagrams and regular models. In *TACAS*, pages 1–15, 2009.
`http://move.lip6.fr/ software/DDD/`
`http://sourceforge.net/projects/buddy/`
`http://vlsi.colorado.edu/ fabio/CUDD/` .
48. P. Wolper. Constructing automata from temporal logic formulas: A tutorial. In *Lectures on Formal Methods in Performance Analysis (First EEF/Euro Summer School on Trends in Computer Science)*, volume 2090 of *Lecture Notes in Computer Science*, pages 261–277. Springer-Verlag, July 2001.

A EFTRS

Definition 3 ($\mathcal{R}^{l\to r}_{check}$). *Let $l \to r$ be a rule of $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$. We denote by $\mathcal{R}^{l\to r}_{check}$ the elementary FTRS built from l as follows:*

1. $F^p_{l\to r}(x) \to \oplus_{l|_p}(x) \in \mathcal{R}^{l\to r}_{check}$ *if* $p \in \mathcal{P}os_{\mathcal{X}}(l)$ *and where* $\oplus_{l|_p}$ *is a special symbol that we consider to be in* $\mathcal{F}_{bin}$
2. $F^p_{l\to r}(a(x,y)) \to a(F^{p.1}_{l\to r}(x), F^{p.2}_{l\to r}(y)) \in \mathcal{R}^{l\to r}_{check}$ *if* $p \in \mathcal{P}os(l) \setminus \mathcal{FP}\text{os}(l)$ *and* $a = l(p)$
3. $F^p_{l\to r}(\bot) \to \bot \in \mathcal{R}^{l\to r}_{check}$ *if* $p \in \mathcal{FP}\text{os} \setminus \mathcal{P}os_{\mathcal{X}}(l)$

Example 1. Consider the term $a(b(x,\bot), c(d(\bot,y),z)) \to r$ be a rewrite rule, $x,y,z \in \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \{x,y,z\}$. For a matter of readability, we denote this term by l. Thus,

$$\mathcal{R}^{l\to r}_{check} = \left\{ \begin{array}{l} F^{\varepsilon}_{l\to r}(a(x,y)) \to a(F^{1}_{l\to r}(x), F^{2}_{l\to r}(y)) \\ F^{1}_{l\to r}(b(x,y)) \to a(F^{11}_{l\to r}(x), F^{12}_{l\to r}(y)) \\ F^{11}_{l\to r}(x) \to \oplus_x(x) \\ F^{12}_{l\to r}(\bot) \to \bot \\ F^{2}_{l\to r}(c(x,y)) \to c(F^{21}_{l\to r}(x), F^{22}_{l\to r}(y)) \\ F^{21}_{l\to r}(d(x,y)) \to c(F^{211}_{l\to r}(x), F^{212}_{l\to r}(y)) \\ F^{211}_{l\to r}(\bot) \to \bot \\ F^{212}_{l\to r}(x) \to \oplus_y(x) \\ F^{22}_{l\to r}(x) \to \oplus_z(x) \end{array} \right\}.$$

We now introduce the notion of variable tree build from a term.

Definition 4. *Let t be a term of $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$. Let $x_0,\ldots,x_n$ be the variables of t occurring in t from the the leftmost variable of t to the rightmost variable of t. The empty variable tree (variable tree for short) of t, denoted by $\top_{\mathcal{X}}(t)$, is defined by the following term: $\top(\oplus_{x_0}(\bot), \top(\oplus_{x_1}(\bot),\ldots,\top(\oplus_{x_n}(\bot),\bot_\top)\ldots))$.*

Definition 5 ($\mathcal{R}^{l\to r}_{\sigma}$). *Let $l \to r$ be a rule of $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$. Let $x_0,\ldots,x_n$ be the set of variables occurring in l and that can be read from the left to the right in l. Thus, the EFTRS $\mathcal{R}^{l\to r}_{\sigma-compute}$ is defined as follows:*

1. $F^{l\to r}_{copy}(x) \to D_{copy}(RW_{x_0}(RW_{x_1}(\ldots RW_{x_n}(\top(F^{\varepsilon}_t(x), \top_{\mathcal{X}}(t)))\ldots)))$
2. $RW_{x_i}(x) \to RW_{x_i}(RW_{x_i}(x))$
3. $RW_{x_i}(x) \to NMA_{x_i}(x)$
4. $\{NMA_{x_i}(a(x,y)) \to a(NMA_{x_i}(x), NMA_{x_i}(y)) \mid a \in \mathcal{F}_{bin}\}$
5. $\{NMA_{x_i}(\oplus_{x_j}(x)) \to \oplus_{x_j,}(x) \mid x_i \neq x_j\}$
6. $NMA_{x_i}(\bot) \to \bot$
7. $NMA_{x_i}(\top(x,y)) \to \top(NMA_{x_i}(x), NMA_{x_i}(y))$
8. $NMA_{x_i}(\bot_\top) \to \bot_\top$
9. $RW_{x_i}(x) \to RW_{x_i,a}(x)$ *with* $a \in \mathcal{F}_{bin}$
10. $RW_{x_i,a}(\top(x,y)) \to \top(R_{x_i,a}(x), SW^{x_i}_{x_i,a}(y))$ *with* $a \in \mathcal{F}_{bin}$
11. $\{SW^{x}_{x_0,a}(\top(x,y)) \to \top(W_{x,a}(x),y)\} \cup \{SW^{x}_{x_i,a}(\top(x,y)) \to \top(x, SW^{x}_{x_{i-1},a}(y)) \mid i > 0 \wedge i \leq n \wedge x \in \mathcal{V}ar(t)\}$
12. $R_{x,a}(g(x,y)) \to g(R_{x,a}(x), y)$ *with* $g \in \mathcal{F}_{bin}$
13. $R_{x,a}(g(x,y)) \to g(?_x(x), R_{x,a}(y))$ *with* $g \in \mathcal{F}_{bin}$
14. $R_{x,a}(\oplus_x(x)) \to R_{\oplus_x,a}(x)$ *with* $a \in \mathcal{F}_{bin}$
15. $R_{\oplus_x,a}(a(x,y)) \to a(\oplus_x(x), \oplus_x(y))$
16. $R_{\oplus_x,\bot}(\bot) \to \bot$

17. $W_{x,a}(g(x,y)) \to g(W_{x,a}(x),y)$ *with* $g \in \mathcal{F}_{bin}$
18. $W_{x,a}(g(x,y)) \to g(?_{\oplus_x}(x), W_{x,a}(y))$ *with* $g \in \mathcal{F}_{bin}$
19. $W_{x,a}(\oplus_x(\bot)) \to a(\oplus_x(\bot),\oplus_x(\bot))$
20. $W_{x,\bot}(\oplus_x(\bot)) \to \bot$
21. $D_{copy}(\top(x,y)) \to D'_{copy}(\top(y,D(x))$
22. $D'_{copy}(\top(x,\bot)) \to x$
23. $D(x) \to D(D(x))$
24. $D(\bot) \to \bot$
25. $D(x) \to D'(x)$
26. $D'(g(x,y)) \to g(D'(x),D'(y))$
27. $D'(\bot) \to \bot$
28. $D'(g(x,\bot)) \to x$
29. $?_{\oplus_x}(g(x,y)) \to g(?_{\oplus_x}(x),?_{\oplus_x}(y))$ *with* $g \in \mathcal{F}_{bin}$ *and* $x \in \mathcal{V}ar(t)$
30. $?_{\oplus_x}(\bot) \to \bot$

Definition 6 ($\mathcal{R}_{GS}^{l\to r}$)**.** *Let $l \to r$ be a rule of $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$. Let $x_0,\ldots,x_n$ be the variables from the leftmost one of l to the rightmost one. Let Indexes be the set of pairs of variable indexes corresponding to the same initial variable. Let $\mathcal{R}_{GS}^{l\to r}$ be the EFTRS dedicated to check that two variables have the same term on the instantiated variable tree. Thus, $\mathcal{R}_{GS}^{l\to r}$ is composed of the following rules:*

1. $Check_{l\to r}(x) \to CFor_{i_n,j_n}(CFor_{i_{n-1},j_{n-1}}(\ldots CFor_{i_1,j_1}(x)\ldots))$ *with* $Indexes = \{(i_1,j_1),\ldots,(i_n,j_n)\}$
2. $PutMarks_i(\top(x,y)) \to \top(x, PutMarks_{i-1}(y))$ *for* $i > 1$
3. $PutMarks_0(\top(x,y)) \to \top(\oplus_=(x),y)$
4. $CFor_{i,j}(x) \to Test_{=_{i,j}}(PutMarks_i(PutMarks_j(x)))$
5. $Test_{=_{i,j}}(x) \to Test_{=_{i,j}}(RR_{i,j}(x))$
6. $Test_{=_{i,j}}(x) \to ?_{x,p}(x)$
7. $\{RR_{i,j}(x) \to RR_{i,j,g}(x)\}$ *with* $g \in \mathcal{F}_{bin}$
8. $\{RR_{i,j,g}(\top(x,y)) \to \top(x, RR_{i-1,j-1,g}(y))\}$ *with* $g \in \mathcal{F}_{bin}$ *and* $i > 0$
9. $\{RR_{0,j,g}(\top(x,y)) \to \top(R_g(x), R_{j-1,g}(y))\}$ *with* $g \in \mathcal{F}_{bin}$ *and* $i > 0$
10. $\{RR_{i,g}(\top(x,y)) \to \top(x, R_{i-1,g}(y))\}$ *with* $g \in \mathcal{F}_{bin}$ *and* $i > 0$
11. $\{RR_{0,g}(\top(x,y)) \to \top(R_g(x),y)\}$ *with* $g \in \mathcal{F}_{bin}$ *and* $i > 0$
12. $\{R_g(g(x,y)) \to g(R_g(x),y), R_g(g(x,y)) \to g(?_{\oplus_=}(x), R_g(y))\}$ *with* $g \in \mathcal{F}_{bin}$
13. $R_g(\oplus_=(x)) \to R_{g,\oplus_=}(x)$
14. $R_{g,\oplus_=}(g(x,y)) \to g(\oplus_=(x),\oplus_=(y))$
15. $R_{\bot,\oplus_=}(\bot) \to \bot$

Definition 7 (Substitution of variables by markers). *Let $l \to r$ be a rewrite rule of $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$. We denote by $\sigma_{l\to r}$ a marker substitution built such that:*

$$\sigma_{l\to r} = \{x \mapsto \oplus_x(\bot) \mid x \in \mathcal{V}ar(r)\}.$$

Definition 8 ($\mathcal{R}_{\sigma-apply}^{l\to r}$)**.** *Let $l \to r$ be a rule of $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$. Thus, $\mathcal{R}_{\sigma-apply}^{l\to r}$ contains the following rules:*

1. $F_{rewrite}^{l\to r}(x) \to D_\top(RW_{x_0}^{rew}(RW_{x_1}^{rew}(\ldots RW_{x_n}^{rew}(\top(r\sigma_{l\to r},x))\ldots)))$
2. $RW_{x_i}^{rew}(x) \to RW_{x_i}^{rew}(RW_{x_i}^{rew}(x))$
3. $RW_{x_i}^{rew}(x) \to NMA_{x_i}(x)$
4. $RW_{x_i}^{rew}(x) \to RW_{x_i,a}^{rew}(x)$ *with* $a \in \mathcal{F}_{bin}$
5. $RW_{x_i,a}^{rew}(\top(x,y)) \to \top(W_{x_i,a}(x), SR_{x_i,a}(MS_{x_i}^{x_i}(?_{\oplus_{x_i}}y)))$ *with* $a \in \mathcal{F}_{bin}$
6. $\{MS_{x_0}^z(\top(x,y)) \to \top(\oplus_z(x),y)\} \cup \{MS_{x_i}^z(\top(x,y)) \to \top(x, MS_{x_{i-1}}^z(y)) \mid i > 0 \wedge i \leq n \wedge z \in \mathcal{V}ar(r)\}$
7. $\{SR_{x_0,a}(\top(x,y)) \to \top(R_{x_0,a}(x),y)\} \cup \{SR_{x_i,a}(\top(x,y)) \to \top(x, SR_{x_{i-1},a}(y)) \mid i > 0 \wedge i \leq n\}$
8. $D_\top(\top(x,y)) \to D_\top(\top(x,D(y))$
9. $D_\top(\top(x,\bot)) \to x$
10. $D'(\top(x,y)) \to \top(D'(x),D'(y))$
11. $D'(\top(x,\bot_\top)) \to x$
12. $D'(\top(x,\bot)) \to x$

Fig. 4 illustrates the copy of the value stored in the variable x_2. The process is the same for the variables x_1 and x_0.

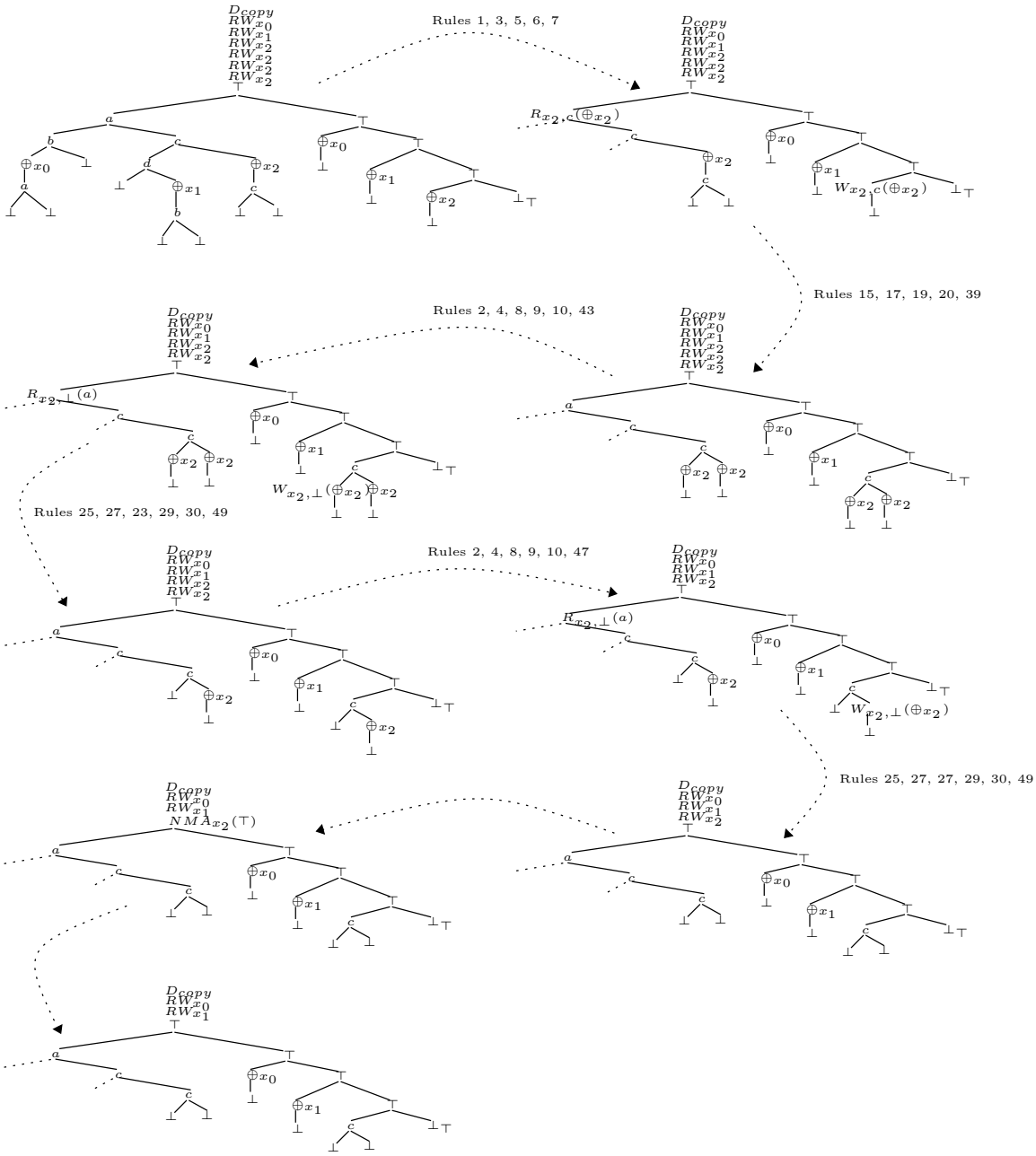

Fig. 4: Copy of the value stored in x_2

B Model-checking of TRS, FTRS and EFTRS

B.1 Simple models

Tree Arbiter Protocol In paper submitted to FMCAD2010.

Percolate Protocol Program percolate consists of a tree of processes, each of them has its local label, which ranges over the set of values "p","n","u". The values "p" (or "n") should be interpreted as "positif" (or "negatif"), the value "u" should be interpreted as "undefined yet", which implies that it will eventually change to either "p" or "n".

Initially, all the leaf processes in the tree are presented as "p"$(\bot, \bot)$ or "n"$(\bot, \bot)$, and all other processes have value "u". The purpose of this model is to percolate to the root of the tree a value "p" if at least one of the leaves has value "p", and a value of "n", if all leaves have value "n". If a process does not yet have a defined value but all its childrens' values are defined then the process sets its value to the disjunction of the values of its children.

Consequently, we can represent a configuration of program percolate as a tree over the alphabet $\{$"p","n","u",$\bot\}$ where "p","n","u" are binary symbols and $\bot$ is a special nullary symbol (see the figure 5).

TRSs $\mathcal{R}_1$:

$u(n(x,y),n(z,t)) \rightarrow n(n(x,y),n(z,t))$

$u(p(x,y),z) \rightarrow p(p(x,y),z)$

$u(x,p(y,z)) \rightarrow p(x,p(y,z))$

Corresponding FTRSs:

- *Generated rules (Rules with the left side $H(u(...))$)*
  $H(u(n(x,y),n(z,t))) \rightarrow n(n(x,y),n(z,t))$
  $H(u(p(x,y),z)) \rightarrow p(p(x,y),z)$
  $H(u(x,p(y,z))) \rightarrow p(x,p(y,z))$

- *Circulation (Rules with the left side $H(p(x,y)), H(n(x,y))$ are not necessary generated here)*
  $H(u(x,y)) \rightarrow u(H(x),y)$
  $H(u(x,y)) \rightarrow u(x,H(y))$

- *Fixed point*
  $Percolate(x) \rightarrow x$
  $Percolate(x) \rightarrow Percolate(H(x))$

Corresponding EFTRS:

- *Generated rules*
  $H(u(x,y)) \rightarrow n(Neg(x),Neg(y))$
  $H(u(x,y)) \rightarrow p(Pos(x),y)$
  $H(u(x,y)) \rightarrow p(x,Pos(y))$
  $Neg(n(x,y)) \rightarrow n(x,y)$
  $Pos(p(x,y)) \rightarrow p(x,y)$

- *Circulation*
  $H(u(x,y)) \rightarrow u(H(x),y)$
  $H(u(x,y)) \rightarrow u(x,H(y))$

- *Fixed point*
  $Percolate(u(x,y)) \rightarrow u(x,y)$
  $Percolate(u(x,y)) \rightarrow Percolate(H(u(x,y)))$

TRSs $\mathcal{R}_1$ and corresponding functional TRSs can give the same result set E_1 from an initial set E_0: $E_1 = \mathcal{R}_1^\star(E_0) = Percolate(E_0)$.

We are interested only in a configuration of E_1 or $\mathcal{R}_1^\star(E_0)$ that contains only labels "n", "p" and $\perp$ (see the figure 6). We can find the final result E_1' by using $Inv(Percolate(E_0))$. Futher detail of FTRS about Inv in Section B.2.

Leader Election Protocol A set of processes, denoted by the leaves, want to elect a leader. Each of them decides first whether to be a candidate or not. The election process proceeds in two phases.

The first phase consists of the internal nodes polling their children nodes to see if at least one of them is candidate. In such a case, the internal node becomes a candidate as well and will be labelled "p", otherwise it will be labelled "n".

Similarly as the program Percolate, all the leaf processes in the tree have val between n and p, and all other processes have val "u". The purpose of the first phase is to percolate to the root of the tree a value p if at least one of the leaves has value "p", and a value of "n", if all leaves have value n. If a process does not yet have a defined value but all its childrens' values are defined then the process sets its value to the disjunction of the values of its children.

The second phase is the actual election procedure. In case the root assigned "n", we have no candidate and can not continue the election. From the result of the first phase, we prepare for the second phase by marking root by a label "c" (see the figure 7). The root chooses (elects) one candidate non-deterministically among its children labelled "p". An internal node that has been elected (noted by the label "c"), elects in turn one of its children that declared itself candidate.

TRSs $\mathcal{R}_2$:

$c(p(x,y),p(z,t)) \to c(c(x,y),p(z,t))$

$c(p(x,y),p(z,t)) \to c(p(x,y),c(z,t))$

$c(p(x,y),n(z,t)) \to c(c(x,y),n(z,t))$

$c(n(x,y),p(z,t)) \to c(n(x,y),c(z,t))$

Corresponding FTRS:

- *Generated rules*
  $H(c(p(x,y),p(z,t))) \to c(c(x,y),p(z,t))$
  $H(c(p(x,y),p(z,t))) \to c(p(x,y),c(z,t))$
  $H(c(p(x,y),n(z,t))) \to c(c(x,y),n(z,t))$
  $H(c(n(x,y),p(z,t))) \to c(n(x,y),c(z,t))$

- *Circulation (Rules with the left side $H(p(x,y)), H(n(x,y))$ are not necessary generated here)*
  $H(c(x,y)) \to c(H(x),y)$
  $H(c(x,y)) \to c(x,H(y))$

- *Fixed point*
  $Election(x) \to x$
  $Election(x) \to Election(H(x))$

Corresponding EFTRS:

- *Generated rules*
  $H(c(x,y)) \to c(Elec(x),Neg(y))$
  $H(c(x,y)) \to c(Elec(x),Pos(y))$

$H(c(x,y)) \to c(Pos(x), Elec(y))$
$H(c(x,y)) \to c(Neg(x), Elec(y))$
$Elec(p(x,y)) \to c(x,y)$
$Neg(n(x,y)) \to n(x,y)$
$Pos(p(x,y)) \to p(x,y)$
$Elec(\bot) \to \bot$
$Pos(\bot) \to \bot$
$Neg(\bot) \to \bot$

– *Circulation*
$H(c(x,y)) \to c(H(x), Neg(y))$
$H(c(x,y)) \to c(H(x), Pos(y))$
$H(c(x,y)) \to c(Pos(x), H(y))$
$H(c(x,y)) \to c(Neg(x), H(y))$

– *Fixed point*
$Election(c(x,y)) \to c(x,y)$
$Election(c(x,y)) \to Election(H(c(x,y)))$

TRSs $\mathcal{R}_2$ and corresponding functional TRSs can give the same result set E_2 from an initial set $E"_1$: $E_2 = \mathcal{R}_2^\star(E"_1) = Election(E"_1)$.

From $\mathcal{R}_2^\star(E"_1)$, we are interested in tree having a path only "c" from root to leaf. We can find the final result by using $Inv(Election(E"_1))$ (see the figure 8). Futher detail of FTRS about Inv in Section B.2.

B.2 Invariants

Invariants are the most common and useful of safety properties, that is, properties stating that some bad things should never happen. Given a system and an initial state *init*, an invariant Inv is a predicate defining a subset of states having two properties:

– it contains *init*,
– it contains all of reachable states from *init*.

We check if the set of bad states $Bad = \neg Inv = \mathcal{R}^\star(init) \backslash Inv(\mathcal{R}^\star(init))$ is empty.

For example, given a term $clock = time(i(\bot,\bot),\bot)$, and a FTRS: $\mathcal{R} = \{H(time(x,y)) \to time(H(x),y),\ H(0(x,y)) \to 1(x,y),\ H(1(x,y)) \to 2(x,y)$,..., $H(23(x,y)) \to 0(x,y)\}$ with $i = 1..23 \in \mathcal{F}_{bin}$.

Here we want to verify if $time(i(\bot,\bot),\bot)$ has $i = 1..23 \in \mathcal{F}_{bin}$ by launching a test $\mathcal{R}_{test} = \{Inv(time(x,y)) \to time(Inv(x),y), Inv(0(x,y)) \to 0(x,y),$ $Inv(1(x,y)) \to 1(x,y)$,..., $Inv(23(x,y)) \to 23(x,y)\}$.

1. **Tree Arbiter Protocol.** In paper submitted to FMCAD2010.
We have another solution: Compute $Inv = Inv_{OnlyT}(\mathcal{R}^\star(init))$:
$Inv_{OnlyT}(a(x,y)) \to a(Inv_{OnlyT}(x), Inv_{NoT}(y))$
$Inv_{OnlyT}(a(x,y)) \to a(Inv_{NoT}(x), Inv_{OnlyT}(y))$
$Inv_{OnlyT}(t(x,y)) \to t(Inv_{NoT}(x), Inv_{NoT}(y))$
$Inv_{NoT}(a(x,y)) \to a(Inv_{NoT}(x), Inv_{NoT}(y))$
$Inv_{NoT}(\bot) \to \bot$

with $a \in \{i, r, b\}$

We also show how we can find this results in TRS: We must use markers that always provide an explosion of system states.
Let begin from the bottom and go upward step by step, we will replace each node by "conflict" if all of his children are "onlyt" or at least one is "conflict", by "not" if all of his children are "not", and by "onlyt" in otherwise.
TRS: $a(\bot, \bot) \to not(\bot, \bot)$ with $a \in \{i, r, b\}$
$t(\bot, \bot) \to onlyt(\bot, \bot)$

$a(conflict(x, y), z) \to conflict(conflict(x, y), z)$
$a(x, conflict(y, z)) \to conflict(x, conflict(y, z))$

$a(onlyt(x, y), onlyt(z, w))$
$\to conflict(onlyt(x, y), onlyt(z, w))$
$t(onlyt(x, y), onlyt(z, w))$
$\to conflict(onlyt(x, y), onlyt(z, w))$

$a(not(x, y), not(z, w)) \to not(not(x, y), not(z, w))$
$t(not(x, y), not(z, w)) \to onlyt(not(x, y), not(z, w))$

$a(not(x, y), onlyt(z, w)) \to onlyt(not(x, y), onlyt(z, w))$
$a(onlyt(x, y), not(z, w)) \to onlyt(onlyt(x, y), not(z, w))$

$t(not(x, y), onlyt(z, w)) \to conflict(not(x, y), onlyt(z, w))$
$t(onlyt(x, y), not(z, w)) \to conflict(onlyt(x, y), not(z, w))$

2. **Percolate Protocol.** An initial set E_0 can be depicted in the figure 5.

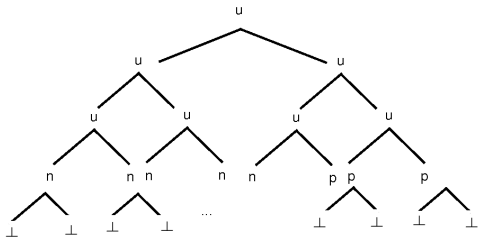

Fig. 5: Initial set E_0

We are interested in tree containing only labels "n", "p" and $\bot$ (see the figure 6):

$$E_1 = (\mathcal{R}_1^\star(E_0) \cap T(\{n, p, \bot\})) = Inv(Percolate(E_0)).$$

Corresponding FTRSs is decribed below: $Inv(\bot) \to \bot$
$Inv(p(x, y)) \to p(Inv(x), Inv(y))$

$Inv(n(x,y)) \rightarrow n(Inv(x), Inv(y))$

Finally, we consider the sub tree whose root is labeled by "p":

$$E'_1 = (E_1 \cap \{p(t_1,t_2)|t_1,t_2 \in T\}) = P(E_1).$$

where $P(p(x,y)) \rightarrow p(x,y)$.
For conclusion, $E'_1 \neq \emptyset$ means that at least one of the leaves has value "p", $E'_1 = \emptyset$ means that all leaves have value "n".

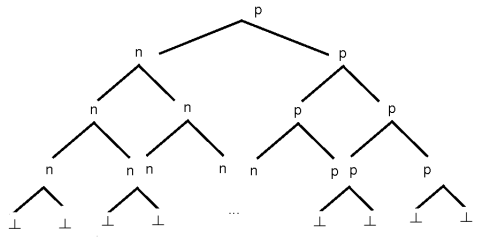

Fig. 6: $E'_1 = Inv(Percolate(E_0))$

3. **Leader Election Protocol.** From the result of the first phase (see the figure 6), we prepare for the second phase by by marking root by a label "c" (see $E"_1$ the figure 7).

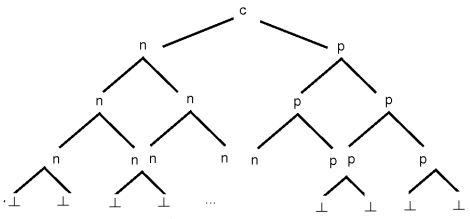

Fig. 7: Initial set $E"_1$ of the second phase

In the second phase we tried to go down and selected the candidate (labeled by "p") if possible: $E_2 = \mathcal{R}_2^\star(E"_1) = Election(E"_1)$.
From the result of the second phase $R^\star(E"_1)$, we are interested in tree having a path only "c" from root to leaf. The filtration is computed by $E'_2 = Inv(E_2)$.
EFTRSs is decribed below:
$O(a(x,y)) \rightarrow a(x,y)$ with $a \in \{n,p\}$
$Inv(\perp) \rightarrow \perp$

$O(\bot) \to \bot$
$Inv(c(x,y)) \to c(O(x), Inv(y))$
$Inv(c(x,y)) \to c(Inv(x), O(y))$
This trees set E'_2 is depicted in the figure 8.
We also show how we can find this results in TRS: We can count the number of "c" in the leaves, etc.
We have another solution: We begin from the bottom, replace each label "c" by "o" and go back step by step. The sub tree that has a root labeled "o" is the result we need but now it is tree having a path where there are only the labels "o" from root to leaf.
TRS: $c(\bot,\bot) \to o(\bot,\bot)$
$c(o(x,y),z) \to o(o(x,y),z)$
$o(x,o(y,z)) \to o(x,o(y,z))$

FTRS:
- *Generated rules* $O(o(x,y)) \to o(x,y)$
  $O(\bot) \to \bot$
  $H(c(x,y)) \to o(O(x),y)$
  $H(c(x,y)) \to o(x,O(y))$
- *Circulation* $H(c(x,y)) \to c(H(x),y)$
  $H(c(x,y)) \to c(x,H(y))$
- *Fixed point* $Inv(x) \to x$
  $Inv(x) \to Inv(H(x))$

Finally, we consider the sub tree whose root is labeled by "o".

$$E"_2 = (E'_2 \cap \{o(t_1,t_2) | t_1,t_2 \in \mathcal{T}(\mathcal{F}_{bin})\}) = O(E'_2).$$

where $O(o(x,y)) \to o(x,y)$

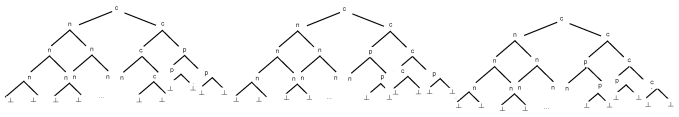

Fig. 8: $E_2 = Inv(Election(E"_1))$

This tree is exactly liked the set of trees in the figure 8 except that the path of labels "c" is replaced by the path of labels "o".

B.3 Experimental results of TRSs, FTRSs and EFTRSs

In this section, we aim at comparing our EFTRS Model-checker (written in JAVA) with two available tools like timbuk ([30], downloadable from http://www.irisa.fr/celtique/genet/timbuk/), maude([26], downloadable from http://maude.cs.uiuc.edu), etc by testing on some communication protocols [1, 37] like Percolate Protocol, Leader Election Protocol and Tree Arbiter Protocol.

Hereafter reported results were obtained on a PC 1.7GHz 1GB RAM. More details of the representations of tests in Timbuk, Maude can be found in Section C.

1. **Tree Arbiter Protocol.** These models are decribed in the paper. We begin by compute $\mathcal{R}^{\star}(v)$ without the requests rules.
   In the below table the computation time is computed by second. It should be noted that on the table the exponential increase of the model size N leads to the drammatical increase of reachability configuration number $\#Confs$.

N	#Confs	Timbuk	Maude	EFTRS
2^5	43264	56.430	3.672	N/A
2^{10}	-	$> 3h$	$> 3h$	N/A

TRSs

   N/A means that our EFTRS model-checker does not support for TRSs and FTRSs.

N	#Confs	Timbuk	Maude	EFTRS
2^5	43264	$> 3h$	82.373	N/A
2^{10}	-	-	$> 3h$	N/A

FTRSs

   Maude is efficient in either TRS or FTRSs while Timbuk can not realize FTRSs and EFTRSs.

N	#Confs	Timbuk	Maude	EFTRS
2^5	43264	$> 3h$	379.816	0.139
2^{10}	4.758 E+16	-	$> 3h$	0.184
2^{20}	2.173 E+63	-	-	0.218
2^{50}	8.591 E+383	-	-	0.308
2^{150}	1.929 E+3409	-	-	0.634
2^{250}	8.648 E+9444	-	-	1.075
2^{350}	7.732 E+18490	-	-	1.418
2^{450}	1.379 E+30547	-	-	1.809

EFTRSs

   Maude for EFTRSs are broken very soon while EFTRSs-checker latency remains quasi constant for this model. It can be explained by the speeding up of LFP approach intergrated in our EFTRSs-checker.

   In the other hand, we also have Benchmarks of the model with the requests rules.

N	#Confs	Timbuk	Maude	EFTRS
2^5	8.539 E+8	$> 3h$	$> 3h$	N/A
2^{10}	-	-	-	N/A

TRSs

This problem now becomes so difficult that neither Maude nor Timbuk can reach to the model size $N = 2^5$.

N	#Confs	Timbuk	Maude	EFTRS
2^5	8.539 E+8	$> 3h$	$> 3h$	N/A
2^{10}	-	-	-	N/A

FTRSs

But LFP technique still brings to our checker a surprise performance [3] to reach to the model size $N = 2^{400}$.

N	#Confs	Timbuk	Maude	EFTRS
2^5	8.539 E+8	$> 3h$	$> 3h$	0.109
2^{10}	1.989 E+273	-	-	0.196
2^{20}	5.350 E+278807	-	-	0.256
2^{50}	?	-	-	0.430
2^{100}	?	-	-	0.789
2^{200}	?	-	-	1.453
2^{300}	?	-	-	2.137
2^{400}	?	-	-	2.866

EFTRSs

2. **Percolate Protocol.** Their descriptions and models occur in Section B.1.

N	#Confs	Timbuk	Maude	EFTRS
2^5	11047	0.138	1.479	N/A
2^{10}	8.445 E+135	24.179	$> 3h$	N/A
2^{20}	-	$> 3h$	-	N/A

TRSs

For this example, Timbuk has alway the best performance in either TRSs or FTRSs.

N	#Confs	Timbuk	Maude	EFTRS
2^5	11047	0.380	6.804	N/A
2^{10}	8.445 E+135	2.956	$> 3h$	N/A
2^{20}	-	$> 3h$	-	N/A

FTRSs

N	#Confs	Timbuk	Maude	EFTRS
2^5	11047	0.165	19.041	0.018
2^{10}	8.445 E+135	0.555	$> 3h$	0.053
2^{20}	4.508 E+139402	248.830	-	0.102
2^{50}	?	$> 1G$	-	0.139
2^{100}	?	-	-	0.183
2^{200}	?	-	-	0.293
2^{500}	?	-	-	0.396

[3] The configurations count is noted "?" when the library *java.math.BigDecimal* we used can not return the result within three hours.

EFTRSs

Like the Tree Arbiter Protocol, EFTRSs-checker latency also remains quasi constant while the others are broken very soon. It can be explained by the speed up of LFP approach for EFTRSs.

3. **Leader Election Protocol.** Their descriptions and models occur in Section B.1. The performance of the LFP approach is shown on the table below with the Election Leader Protocol. Notice that in the worst case, the problem size (2^n) will produce $2^{(n-1)}$ election possibilities. N/A in the below table means that our EFTRS model-checker does not support for TRSs.

N	#Confs	Timbuk	Maude	EFTRS
2^5	16	0.159	0.014	N/A
2^{10}	512	7.595	71.981	N/A
2^{20}	-	$> 3h$	$> 3h$	N/A

TRSs

Like the Tree Arbiter Protocol, the tables also show that Timbuk tool is efficient in TRS but it can not realize FTRSs and EFTRSs as Maude.

N	#Confs	Timbuk	Maude	EFTRS
2^5	16	$> 3h$	0.457	N/A
2^{10}	512	-	$> 3h$	N/A
2^{20}	5.243 E+5	-	-	N/A

FTRSs

N	#Confs	Timbuk	Maude	EFTRS
2^5	16	$> 3h$	0.920	0.093
2^{10}	512	-	$> 3h$	0.159
2^{20}	5.243 E+5	-	-	0.256
2^{50}	5.629 E+14	-	-	0.956
2^{100}	6.338 E+29	-	-	3.194
2^{200}	8.035 E+59	-	-	18.355
2^{300}	1.019 E+90	-	-	57.430
2^{400}	1.291 E+120	-	-	138.699

EFTRSs

The tables also shows that our EFTRS model-checker is alway efficient than Maude.

C Timbuk, Maude tests

This section we show how we can build the tests in Timbuk, Maude. Here is the representations of tests in Timbuk, Maude:

- Timbuk There is no problem in classic TRSs but we must add the equations *Rules H(H(X))=H(X)* to ensure the termination of the fixpoint in FTRSs.
- Maude There is no problem in classic TRSs but if we want to use the fixpoint in FTRSs, we should define $\mathcal{F}_{bin}$ and $\mathcal{F}_{NT}$ to ensure the termination.

C.1 Timbuk

1. **Classical TRS.**
2. **FTRS with Timbuk.** Add the equations *Rules H(H(X))=H(X)* to ensure the termination of the fixpoint.

```
F   (  X )                ->X
F   (  X )                ->F   (  H   (  X ) )
```

3. **EFTRS with Timbuk.** Add the equations *Rules H(H(X))=H(X)* to ensure the termination of the fixpoint.

```
F   ( b (X,Y) )           ->b (X,Y)
F   ( t (X,Y) )           ->t (X,Y)
F   ( b (X,Y) )           ->F   (  H   (  b (X,Y) ) )
F   ( t (X,Y) )           ->F   (  H   (  t (X,Y) ) )
```

Further details see at http://eftrs.svn.sourceforge.net/viewvc/eftrs/timbuk/

C.2 Maude

```
load arbiter.frs.maude$
search F(t(r(T,T),r(T,T))) =>* t(X:Terminal, Y:Terminal) .
```

1. **Classical TRS.**
2. **FTRS with Maude.** We should distinguish reducible and irreducible symbols to ensure the termination.

```
rl F   (  X )                =>X .
rl F   (  X )                =>F   (  H   (  X ) )  .
```

3. **EFTRS with Maude.** We should distinguish reducible and irreducible symbols to ensure the termination.

```
rl F   (  b (X,Y) )          =>b (X,Y) .
rl F   (  t (X,Y) )          =>t (X,Y) .
rl F   (  b (X,Y) )          =>F   (  H   (  b (X,Y) ) )  .
rl F   (  t (X,Y) )          =>F   (  H   (  t (X,Y) ) )  .
```

Further details see at http://eftrs.svn.sourceforge.net/viewvc/eftrs/maude/