Autowrite: A Tool for Term Rewrite Systems and Tree Automata

Irène Durand [1]

LaBRI
Université Bordeaux 1
Talence, France

Abstract

Autowrite is an experimental software tool written in Common Lisp Oriented System (CLOS) which handles term rewrite systems and bottom-up tree automata. A graphical interface written using McCLIM, (the free implementation of the CLIM specification) frees the user of any Lisp knowledge. Software and documentation can be found at `http://dept-info.labri.u-bordeaux.fr/~idurand/autowrite`. Autowrite was initially designed to check call-by-need properties of term rewrite systems. For this purpose, it implements the tree automata constructions used in [11,4,6,14] and many useful operations on terms, term rewrite systems and tree automata.

Key words: Tree automata, Term rewriting

1 Introduction

Huet and Lévy [10] showed that for the class of orthogonal term rewrite systems (TRSs) every term not in normal form contains a needed redex (i.e., a redex contracted in every normalizing rewrite sequence) and that repeated contraction of needed redexes results in a normal form if it exists. However, neededness is in general undecidable. In order to obtain a decidable approximation to neededness, Huet and Lévy introduced the subclass of *strongly sequential* TRSs.

In a strongly sequential TRS, at least one of the needed redexes in every reducible term can be effectively computed. Moreover, Huet and Lévy showed that strong sequentiality is a decidable property of orthogonal TRSs. Strong sequentiality is based on the idea of approximating the TRS by replacing each right-hand side of every rule by a fresh variable (this known as the *strong approximation*). That always gives a really rough approximation of the TRS.

[1] Email: `idurand@labri.fr`

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

Several authors [2,4,11,13,14,15,16] proposed decidable extensions of the class of strongly sequential TRSs. Since Comon's [2] and Jacquemard's [11] work, all the corresponding decidability proofs have been expressed using tree automata techniques: typically a property is satisfied if and only if some associated automaton recognizes the empty language. A uniform framework for the study of call-by-need (sequentiality) is described in [4] where classes of term rewrite systems are parameterized by approximation mappings. Nowadays the best known approximation (easily definable) for which call-by-need is decidable is the growing approximation studied in [14]. No TRS is strong (equal to its strong approximation) because of the requirement that every variable in the right-hand side of a rule should also appear in the left-hand side. In return, some TRSs are already growing (equal to their growing approximation). For these TRSs call-by-need is decidable which makes the results for the growing case very valuable.

`Autowrite` was initially designed to check call-by-need properties of most of the examples presented in [7]. Most of the time no alternative proofs exists. It implements the tree automata constructions of the call-by-need theory and many operations on terms, term rewrite systems and tree automata.

In the first version of `Autowrite` [5], only the call-by-need properties and a few other simple properties were available from the graphical interface. This new version of `Autowrite` includes many new functionalities. There are new functionalities related to TRSs, but the most interesting new feature is the possibility to directly handle (load, save, combine with boolean operations) bottom-up tree automata. In addition, we have added on-line timing information. Since the first version the run-times have been considerably improved due to better choices of data structures. The new features allowed testing many properties of examples presented in [8] for which no easy proof can be written.

2 Preliminaries

We assume that the reader is familiar with the basics of term rewriting [12] and tree automata [3]. Familiarity with the theory of call-by-need [10,15,11,4] [6,2,7,14] is helpful.

A *term rewrite system* (TRS for short) $\mathcal{R}$ over a finite *signature* $\mathcal{F}$ (set of symbols with fixed arity) consists of *rewrite rules* $l \to r$ between terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ that satisfy $l \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. Here $\mathcal{V}$ is a denumerable set of *variables*. If the second condition is not imposed we find it useful to speak of *extended* TRSs (eTRSs). Such TRSs arise naturally when we approximate TRSs, as explained in Section 2.2.

A *ground* term does not contain variables. A *linear* term does not contain multiple occurrences of the same variable. A *redex* is an instance of the left-hand side of a rewrite rule. A *normal form* is a term without redexes. The set of all ground normal forms of a TRS $\mathcal{R}$ is denoted by $\mathsf{NF}(\mathcal{R})$. An eTRS is *left-linear* (*right-linear*, *linear*) if the left-hand sides (right-hand sides, both left and right-hand sides) of its rewrite rules are linear terms. An eTRS is *right-ground* (*ground*) if the right-hand sides (left and right-hand sides) of its rewrite rules are ground terms. A left-linear TRS without critical pairs is *orthogonal*. Orthogonal TRSs have the property that every term has at most one normal form.

In the remainder of this section we recall some basic definitions concerning tree automata. Much more information can be found in [3]. A (finite bottom-up) *tree automaton* is a quadruple $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ consisting of a finite signature $\mathcal{F}$, a finite set Q of states, disjoint from $\mathcal{F}$, a subset $Q_f \subseteq Q$ of final states, and a set of transition rules Δ. Every transition rule is of the form $f(q_1, \ldots, q_n) \to q$ with $f \in \mathcal{F}$ and $q_1, \ldots, q_n, q \in Q$ or $q \to q'$ with $q, q' \in Q$. The latter rules are called ϵ-transitions. So a tree automaton $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ is simply a finite ground TRS Δ over the signature $\mathcal{F} \cup Q$ whose rewrite rules have a special shape, together with a subset Q_f of Q. Let $\mathcal{T}(\mathcal{F})$ be the set of ground terms. A term $t \in \mathcal{T}(\mathcal{F})$ is accepted by $\mathcal{A}$ if $t \to_{\mathcal{A}}^{+} q$ for some $q \in Q_f$. The set of all such terms is denoted by $L(\mathcal{A})$. A subset $L \subseteq \mathcal{T}(\mathcal{F})$ is called *regular* if there exists a tree automaton $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ such that $L = L(\mathcal{A})$. It is well-known and easy to build an automaton recognizing $\mathcal{T}(\mathcal{F})$.

2.1 Call-by-need

Let $\mathcal{R}_\bullet$ be the eTRS $\mathcal{R} \cup \{\bullet \to \bullet\}$ over the extended signature $\mathcal{F}_\bullet = \mathcal{F} \cup \{\bullet\}$. We say that redex Δ in $C[\Delta] \in \mathcal{T}(\mathcal{F})$ is $\mathcal{R}$-*needed* if there is no term $t \in \mathsf{NF}(\mathcal{R})$ such that $C[\bullet] \to_{\mathcal{R}}^{*} t$. Finally, we say that $\mathcal{R}$ is *in* $\mathcal{CBN}$ (*Call-By-Need*) if every reducible term in $\mathcal{T}(\mathcal{F})$ contains a $\mathcal{R}$-needed redex.

Theorem 2.1 *[10] Let $\mathcal{R}$ be an orthogonal TRS.*

(i) *Every reducible term contains a needed redex.*

(ii) *Repeated contraction of needed redexes results in a normal form, whenever the term under consideration has a normal form.*

□

So, for orthogonal TRSs, the strategy that always selects a needed redex for contraction is normalizing and optimal. Unfortunately, needed redexes are not computable in general. Hence, in order to obtain a *computable* optimal strategy, we need to find (1) decidable approximations of neededness and (2) (decidable) classes of rewrite systems which ensure that every reducible term has a needed redex identified by (1).

2.2 Approximation mappings

Let $\mathcal{R}$ and $\mathcal{S}$ be eTRSs over the same signature. We say that $\mathcal{S}$ approximates $\mathcal{R}$ if $\to_{\mathcal{R}}^{*} \subseteq \to_{\mathcal{S}}^{*}$ and $\mathsf{NF}(\mathcal{R}) = \mathsf{NF}(\mathcal{S})$. An approximation mapping is a mapping

α from TRSs to eTRSs with the property that $\alpha(\mathcal{R})$ approximates $\mathcal{R}$, for every TRS $\mathcal{R}$. Given a TRS $\mathcal{R}$, we say that $\mathcal{R}$ is *in* $\mathcal{CBN}_\alpha$ (*α-sequential*) if $\alpha(\mathcal{R})$ is in $\mathcal{CBN}$.

Next we define the approximation mappings s, nv, and gr. Let $\mathcal{R}$ be a TRS. The *strong* approximation [10] $\mathrm{s}(\mathcal{R})$ is obtained from $\mathcal{R}$ by replacing the right-hand side of every rewrite rule by a variable that does not occur in the corresponding left-hand side. The *non-variable* approximation [15] $\mathrm{nv}(\mathcal{R})$ is obtained from $\mathcal{R}$ by replacing the variables in the right-hand sides of the rewrite rules by pairwise distinct variables that do not occur in the corresponding left-hand sides. The *growing* approximation [11,14] $\mathrm{gr}(\mathcal{R})$ is obtained from $\mathcal{R}$ by renaming the variables in the right-hand sides that occur at a depth greater than 1 in the corresponding left-hand side. Given a TRS $\mathcal{R}$ and $\alpha \in \{\mathrm{s}, \mathrm{nv}, \mathrm{gr}\}$, it is decidable whether $\alpha(\mathcal{R})$ is in $\mathcal{CBN}$. However the decision procedures are very complex (exponential for s, doubly exponential for nv and gr [6]) so in general it is impossible to show by hand that a particular TRS is in $\mathcal{CBN}$, even for very small TRSs. However showing that a TRS is not in $\mathcal{CBN}$ can be done more easily by exhibiting a term with no needed redex.

Example 2.2 The following example is taken from [13].

$$\mathcal{R}_1 = \begin{cases} \mathsf{f}(\mathsf{g}(\mathsf{a}, x), \mathsf{b}) & \to \quad x \\ \mathsf{f}(\mathsf{g}(x, \mathsf{a}), \mathsf{c}) & \to \quad x \\ \mathsf{f}(\mathsf{d}, x) & \to \quad x \\ \mathsf{g}(\mathsf{e}, \mathsf{e}) & \to \quad \mathsf{e} \end{cases}$$

For $\mathcal{R}_1$, we obtain the following approximated TRSs:

$$\mathrm{s}(\mathcal{R}_1) = \begin{cases} \mathsf{f}(\mathsf{g}(\mathsf{a}, x), \mathsf{b}) & \to \quad y \\ \mathsf{f}(\mathsf{g}(x, \mathsf{a}), \mathsf{c}) & \to \quad y \\ \mathsf{f}(\mathsf{d}, x) & \to \quad y \\ \mathsf{g}(\mathsf{e}, \mathsf{e}) & \to \quad y \end{cases} \qquad \mathrm{nv}(\mathcal{R}_1) = \begin{cases} \mathsf{f}(\mathsf{g}(x, \mathsf{a}), \mathsf{b}) & \to \quad y \\ \mathsf{f}(\mathsf{g}(\mathsf{a}, x), \mathsf{c}) & \to \quad y \\ \mathsf{f}(\mathsf{d}, x) & \to \quad y \\ \mathsf{g}(\mathsf{e}, \mathsf{e}) & \to \quad \mathsf{e} \end{cases}$$

$$\mathrm{gr}(\mathcal{R}_1) = \begin{cases} \mathsf{f}(\mathsf{g}(x, \mathsf{a}), \mathsf{b}) & \to \quad y \\ \mathsf{f}(\mathsf{g}(\mathsf{a}, x), \mathsf{c}) & \to \quad y \\ \mathsf{f}(\mathsf{d}, x) & \to \quad x \\ \mathsf{g}(\mathsf{e}, \mathsf{e}) & \to \quad \mathsf{e} \end{cases}$$

Example 2.3 The next example ($\mathcal{R}_2$) comes from [15].

$$\mathcal{R}_2 = \begin{cases} \mathsf{f}(\mathsf{g}(\mathsf{a}, x), \mathsf{a}) & \to \mathsf{c} \\ \mathsf{f}(\mathsf{g}(x, \mathsf{a}), \mathsf{b}) & \to \mathsf{c} \\ \mathsf{f}(\mathsf{k}(\mathsf{a}), x) & \to \mathsf{c} \\ \mathsf{g}(\mathsf{b}, \mathsf{b}) & \to \mathsf{h}(\mathsf{b}) \\ \mathsf{h}(x) & \to \mathsf{k}(x) \end{cases}$$

$$\mathrm{nv}(\mathcal{R}_2) = \begin{cases} \mathsf{f}(\mathsf{g}(\mathsf{a}, x), \mathsf{a}) & \to \mathsf{c} \\ \mathsf{f}(\mathsf{g}(x, \mathsf{a}), \mathsf{b}) & \to \mathsf{c} \\ \mathsf{f}(\mathsf{k}(\mathsf{a}), x) & \to \mathsf{c} \\ \mathsf{g}(\mathsf{b}, \mathsf{b}) & \to \mathsf{h}(\mathsf{b}) \\ \mathsf{h}(x) & \to \mathsf{k}(y) \end{cases} \qquad \mathrm{gr}(\mathcal{R}_2) = \mathcal{R}_2$$

Example 2.4 The last example ($\mathcal{R}_3$) is an extension of Berry's example [1].

$$\mathcal{R}_3 = \begin{cases} \mathsf{f}(x, \mathsf{a}, \mathsf{b}) & \to \mathsf{h}(x) \\ \mathsf{f}(\mathsf{b}, x, \mathsf{a}) & \to \mathsf{h}(x) \\ \mathsf{f}(\mathsf{a}, \mathsf{b}, x) & \to \mathsf{h}(x) \\ \mathsf{h}(\mathsf{k}(x)) & \to \mathsf{g}(x, x) \\ \mathsf{g}(\mathsf{a}, \mathsf{a}) & \to \mathsf{g}(\mathsf{a}, \mathsf{a}) \\ \mathsf{g}(\mathsf{a}, \mathsf{b}) & \to \mathsf{a} \\ \mathsf{g}(\mathsf{b}, \mathsf{a}) & \to \mathsf{b} \end{cases} \qquad \mathrm{gr}(\mathcal{R}_3) = \begin{cases} \mathsf{f}(x, \mathsf{a}, \mathsf{b}) & \to \mathsf{h}(x) \\ \mathsf{f}(\mathsf{b}, x, \mathsf{a}) & \to \mathsf{h}(x) \\ \mathsf{f}(\mathsf{a}, \mathsf{b}, x) & \to \mathsf{h}(x) \\ \mathsf{h}(\mathsf{k}(x)) & \to \mathsf{g}(y, y) \\ \mathsf{g}(\mathsf{a}, \mathsf{a}) & \to \mathsf{g}(\mathsf{a}, \mathsf{a}) \\ \mathsf{g}(\mathsf{a}, \mathsf{b}) & \to \mathsf{a} \\ \mathsf{g}(\mathsf{b}, \mathsf{a}) & \to \mathsf{b} \end{cases}$$

`Autowrite` computes $\alpha(\mathcal{R})$ for any approximation α in $\{\mathrm{s}, \mathrm{nv}, \mathrm{gr}\}$.

Theorem 2.5 *The approximation mappings* s, nv, *and* gr *preserve recognizability.* □

In other words, given a recognizable tree language L, the set $(\to^*_{\alpha(\mathcal{R})})[L]$ of terms that $\alpha(\mathcal{R})$-rewrite to a term in L is recognizable.

Nagaya and Toyama [14] proved the above result for the growing approximation; the tree automaton that recognizes $(\to^*_{\mathrm{gr}})[L]$ is defined as the limit of a finite saturation process. This saturation process is similar to the ones defined in Comon [2] and Jacquemard [11], but by working exclusively with deterministic tree automata, non-right-linear rewrite rules can be handled.

3 Real problems solved by `Autowrite`

3.1 Convince someone that $\mathcal{R} \in \mathcal{CBN}$ for a given $\mathcal{R}$

It is quite easy to convince someone that a TRS is not in $\mathcal{CBN}$ by exhibiting a term with no $\mathcal{R}$-needed redex. However convincing someone that a TRS is in $\mathcal{CBN}$ is not an easy matter; any attempt generally ends up in a long and tedious proof which in addition will work only for the particular TRS considered. Often, in papers about Call-By-Need (or Sequentiality) the authors always prove that some $\mathcal{R} \notin \mathcal{CBN}$ but never that some $\mathcal{R} \in \mathcal{CBN}$. Usually, they just conjecture or say they think that the TRS is in $\mathcal{CBN}$.

For TRSs of reasonable size, we can use `Autowrite` to comfort the reader with his intuition that a TRSs is in $\mathcal{CBN}$. Also, in the search of a TRS in $\mathcal{CBN}$ having particular properties, we were often surprised to learn from `Autowrite` that the candidate TRS was not in $\mathcal{CBN}$ contrary to our intuition. With the term with no $\mathcal{R}$-needed redex exposed by `Autowrite` we would be right away convinced of our mistake.

Take for instance the example of Oyamaguchi who in [15] conjectured that the TRS $\mathcal{R}_2$ is nv-sequential. With `Autowrite` one can easily check that $\mathcal{R}_2 \in \mathcal{CBN}_{\mathrm{nv}}$. This does not imply that $\mathcal{R}_2$ is nv-sequential as $\mathcal{CBN}_{\mathrm{nv}}$ properly includes the class of nv-sequential TRSs but shows that there exists an optimal and computable strategy for $\mathcal{R}_2$.

3.2 Properties related to signature extension

Let $\mathcal{R}$ be a left-linear growing eTRS. In [7,8] we have studied the question whether the property that $\mathcal{R} \in \mathcal{CBN}$ is preserved after adding new function symbols. For that problem, we need to specify the underlying signature in our notation. We write $(\mathcal{R}, \mathcal{G})$ instead of just $\mathcal{R}$ to indicate which signature is used. We write $\mathsf{NF}(\mathcal{R}, \mathcal{F})$ for the set of ground normal forms of an eTRS $\mathcal{R}$ over a signature $\mathcal{F}$. We denote by $\mathsf{WN}(\mathcal{R}, \mathcal{F})$ the set of all ground terms in $\mathcal{T}(\mathcal{F})$ that rewrite in $\mathcal{R}$ to a normal form in $\mathsf{NF}(\mathcal{R}, \mathcal{F})$. Let $\mathcal{F} \subseteq \mathcal{G}$. We denote by $\mathsf{WN}(\mathcal{R}, \mathcal{G}, \mathcal{F})$ the set of terms in $\mathcal{T}(\mathcal{F})$ that have a normal form with respect to $(\mathcal{R}, \mathcal{G})$. We write $\mathsf{WN}_\bullet(\mathcal{R}, \mathcal{G}, \mathcal{F})$ for $\mathsf{WN}(\mathcal{R}_\bullet, \mathcal{G}_\bullet, \mathcal{F}_\bullet)$.

The following proposition (whose proof can be found in the appendix) states that for $(\mathcal{R}, \mathcal{F}) \in \mathcal{CBN}$, if $(\mathcal{R}, \{\mathcal{F} \cup @\}) \in \mathcal{CBN}$ (for some fresh constant $@$) then $(\mathcal{R}, \mathcal{G}) \in \mathcal{CBN}$ for any $\mathcal{G}$ such that $\mathcal{F} \subseteq \mathcal{G}$.

Proposition 3.1 *Let α be an arbitrary approximation mapping. Let $(\mathcal{R}, \mathcal{F})$ a TRS such that $\mathcal{R} \in \mathcal{CBN}_\alpha$. Let $\mathcal{G} \supseteq \mathcal{F}$. Let $@$ be fresh constant symbol (not in $\mathcal{G}$). Let $\mathcal{F}_@ = \mathcal{F} \cup \{@\}$. If $(\mathcal{R}, \mathcal{G}) \notin \mathcal{CBN}_\alpha$ then $(\mathcal{R}, \mathcal{F}_@) \notin \mathcal{CBN}_\alpha$.*

This is why `Autowrite` provides the possibility of testing whether $(\mathcal{R}, \mathcal{G}) \in \mathcal{CBN}$ with $\mathcal{G} = \mathcal{F} \cup \{@\}$.

A normal form is *external* if it is not an instance of a proper non variable subterm of a left-hand side of a rewrite rule in $\mathcal{R}$. The set of all ground external normal forms of a TRS $\mathcal{R}$ is denoted by $\mathsf{ENF}(\mathcal{R})$. $\mathsf{ENF}(\mathcal{R}) \neq \emptyset$ is a sufficient condition for $\mathcal{R} \in \mathcal{CBN}$ being preserved under signature extension. When $\mathsf{ENF}(\mathcal{R}) = \emptyset$, orthogonality is needed in all the sufficient conditions that we have obtained.

A rewrite rule $l \to r$ is *collapsing* if $r \in \mathcal{V}$ and so is an eTRS containing a collapsing rule. For orthogonal nv eTRSs, the condition that $\mathcal{R}$ is collapsing and $\mathsf{WN}(\mathcal{R}, \mathcal{G}, \mathcal{F}) = \mathsf{WN}(\mathcal{R}, \mathcal{F})$ is sufficient for having $\mathcal{R} \in \mathcal{CBN}$ preserved by signature extension. `Autowrite` helped us find examples showing that both restrictions are essential.

The following example shows the necessity of the collapsing condition:

Example 3.2 Let $\mathcal{R}_4$ be the following orthogonal TRS:

$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{f}(x, \mathsf{a}, \mathsf{b}(y, z)) & \to & \mathsf{c}(\mathsf{i}) & \mathsf{f}(\mathsf{c}(x), \mathsf{c}(y), z) & \to & \mathsf{i} \\
\mathsf{f}(x, \mathsf{a}, \mathsf{c}(y)) & \to & \mathsf{i} & \mathsf{g}(x) & \to & \mathsf{b}(x, \mathsf{i}) \\
\mathsf{f}(\mathsf{a}, \mathsf{a}, \mathsf{a}) & \to & \mathsf{i} & \mathsf{h}(\mathsf{a}) & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{a}, \mathsf{b}(x, y), z) & \to & \mathsf{a} & \mathsf{h}(\mathsf{b}(\mathsf{a}, x)) & \to & \mathsf{a} \\
\mathsf{f}(\mathsf{a}, \mathsf{c}(x), y) & \to & \mathsf{i} & \mathsf{h}(\mathsf{b}(\mathsf{b}(x, y), z)) & \to & \mathsf{b}(\mathsf{i}, \mathsf{i}) \\
\mathsf{f}(\mathsf{b}(x, y), z, \mathsf{a}) & \to & \mathsf{a} & \mathsf{h}(\mathsf{b}(\mathsf{c}(x), y)) & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{b}(x, y), \mathsf{b}(z, u), \mathsf{b}(v, w)) & \to & \mathsf{i} & \mathsf{h}(\mathsf{c}(x)) & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{b}(x, y), \mathsf{b}(z, u), \mathsf{c}(v)) & \to & \mathsf{i} & \mathsf{j}(\mathsf{a}, \mathsf{a}) & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{b}(x, y), \mathsf{c}(z), \mathsf{b}(u, v)) & \to & \mathsf{i} & \mathsf{j}(\mathsf{a}, \mathsf{b}(x, y)) & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{b}(x, y), \mathsf{c}(z), \mathsf{c}(u)) & \to & \mathsf{i} & \mathsf{j}(\mathsf{a}, \mathsf{c}(x)) & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{c}(x), \mathsf{a}, \mathsf{a}) & \to & \mathsf{i} & \mathsf{j}(\mathsf{b}(x, y), z) & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{c}(x), \mathsf{b}(y, z), \mathsf{a}) & \to & \mathsf{i} & \mathsf{j}(\mathsf{c}(x), y) & \to & \mathsf{a} \\
\mathsf{f}(\mathsf{c}(x), \mathsf{b}(y, z), \mathsf{c}(u)) & \to & \mathsf{i} & \mathsf{i} & \to & \mathsf{i} \\
\mathsf{f}(\mathsf{c}(x), \mathsf{b}(y, z), \mathsf{b}(u, v)) & \to & \mathsf{i} & & &
\end{array}
$$

over the signature $\mathcal{F}$ consisting of all symbols appearing in the rewrite rules and let $\mathcal{G} = \mathcal{F} \cup \{@\}$.

`Autowrite` is able to check that

- $\mathsf{ENF}(\mathcal{R}_4) = \emptyset$,
- $(\mathcal{R}_4, \mathcal{F}) \in \mathcal{CBN}_{\mathrm{nv}}$,
- $(\mathcal{R}_4, \mathcal{G}) \notin \mathcal{CBN}_{\mathrm{nv}}$ as shown by the term with no $(\mathrm{nv}(\mathcal{R}_4), \mathcal{G})$-needed redex $\mathsf{j}(\mathsf{f}(\Delta, \Delta, \Delta), @)$ with $\Delta = \mathsf{h}(\mathsf{g}(@))$,
- $\mathsf{WN}(\mathrm{nv}(\mathcal{R}_4), \mathcal{G}, \mathcal{F}) = \mathsf{WN}(\mathrm{nv}(\mathcal{R}_4), \mathcal{F})$.

One can verify easily that $\mathsf{j}(\mathsf{f}(\Delta, \Delta, \Delta), @)$ has no $\mathrm{nv}(\mathcal{R}_4), \mathcal{G}$)-needed redex:

$\Delta = \mathsf{h}(\mathsf{g}(@)) \to_{\mathrm{nv}} \mathsf{h}(\mathsf{b}(\mathsf{a}, \mathsf{i})) \to_{\mathrm{nv}} \mathsf{a}$ $\quad$ $\Delta = \mathsf{h}(\mathsf{g}(@)) \to_{\mathrm{nv}} \mathsf{h}(\mathsf{b}(\mathsf{b}(\mathsf{a}, \mathsf{a}), \mathsf{i})) \to_{\mathrm{nv}}$ $\mathsf{b}(\mathsf{i}, \mathsf{i})$

$$\mathsf{j}(\mathsf{f}(\bullet,\Delta,\Delta),@) \rightarrow_{\mathrm{nv}} \mathsf{j}(\mathsf{f}(\bullet,\mathsf{a},\mathsf{b}(\mathsf{i},\mathsf{i})),@) \rightarrow_{\mathrm{nv}} \mathsf{j}(\mathsf{c}(\mathsf{i}),@) \rightarrow_{\mathrm{nv}} \mathsf{a} \in \mathsf{NF}(\mathcal{R},\mathcal{G})$$
$$\mathsf{j}(\mathsf{f}(\Delta,\bullet,\Delta),@) \rightarrow_{\mathrm{nv}} \mathsf{j}(\mathsf{f}(\mathsf{b}(\mathsf{i},\mathsf{i}),\bullet,\mathsf{a}),@) \rightarrow_{\mathrm{nv}} \mathsf{j}(\mathsf{a},@) \qquad \in \mathsf{NF}(\mathcal{R},\mathcal{G})$$
$$\mathsf{j}(\mathsf{f}(\Delta,\Delta,\bullet),@) \rightarrow_{\mathrm{nv}} \mathsf{j}(\mathsf{f}(\mathsf{a},\mathsf{b}(\mathsf{i},\mathsf{i}),\bullet),@) \rightarrow_{\mathrm{nv}} \mathsf{j}(\mathsf{a},@) \qquad \in \mathsf{NF}(\mathcal{R},\mathcal{G})$$

The next example in this section shows the necessity of the restriction to $\alpha \in \{\mathrm{s},\mathrm{nv}\}$.

Example 3.3 Let $\mathcal{R}_5$ be the following orthogonal eTRS:

$$\begin{array}{rlrl}
\mathsf{f}(x,\mathsf{a},\mathsf{b}(y),z) & \rightarrow \mathsf{g}(z) & \mathsf{g}(\mathsf{a}) & \rightarrow \mathsf{i} \\
\mathsf{f}(\mathsf{b}(x),y,\mathsf{a},z) & \rightarrow \mathsf{g}(z) & \mathsf{g}(\mathsf{b}(x)) & \rightarrow \mathsf{i} \\
\mathsf{f}(\mathsf{a},\mathsf{b}(x),y,z) & \rightarrow \mathsf{g}(z) & \mathsf{h}(\mathsf{a}) & \rightarrow \mathsf{i} \\
\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{a},x) & \rightarrow \mathsf{i} & \mathsf{h}(\mathsf{b}(x)) & \rightarrow \mathsf{j}(\mathsf{i},x) \\
\mathsf{f}(\mathsf{b}(x),\mathsf{b}(y),\mathsf{b}(z),u) & \rightarrow \mathsf{i} & \mathsf{j}(x,\mathsf{a}) & \rightarrow \mathsf{a} \\
\mathsf{i} & \rightarrow \mathsf{i} & \mathsf{j}(x,\mathsf{b}(y)) & \rightarrow \mathsf{b}(\mathsf{a})
\end{array}$$

over the signature $\mathcal{F}$ consisting of all symbols appearing in the rewrite rules. Note that the growing approximation only modifies the rule $\mathsf{h}(\mathsf{b}(x)) \rightarrow \mathsf{j}(\mathsf{i},x)$ into $\mathsf{h}(\mathsf{b}(x)) \rightarrow \mathsf{j}(\mathsf{i},y)$. Let $\mathcal{G} = \mathcal{F} \cup \{@\}$.

`Autowrite` is able to check that

- $\mathsf{ENF}(\mathcal{R}_5) = \emptyset$,
- $(\mathcal{R}_5,\mathcal{F}) \in \mathcal{CBN}_{\mathrm{g}}$,
- $(\mathcal{R}_5,\mathcal{G}) \notin \mathcal{CBN}_{\mathrm{g}}$ as shown by the term with no $\mathrm{gr}(\mathcal{R}_5)$-needed redex $\mathsf{f}(\Delta,\Delta,\Delta,@)$, with $\Delta = \mathsf{h}(\mathsf{j}(@))$,
- $\mathsf{WN}(\mathrm{gr}(\mathcal{R}_5),\mathcal{F}) = \mathsf{WN}(\mathrm{gr}(\mathcal{R}_5),\mathcal{G},\mathcal{F})$.

Note that $\mathcal{R}$ is not collapsing. This is not essential, since adding the single collapsing rule $\mathsf{k}(x) \rightarrow x$ to $\mathcal{R}$ does not affect any of the above properties.

For an nv eTRS $\mathcal{R}$, we have the nice property that

$$\mathsf{WN}(\mathcal{R},\mathcal{G},\mathcal{F}) = \mathsf{WN}(\mathcal{R},\mathcal{F}) \Rightarrow \mathsf{WN}_\bullet(\mathcal{R},\mathcal{G},\mathcal{F}) = \mathsf{WN}_\bullet(\mathcal{R},\mathcal{F})$$

`Autowrite` helped us showing that the restriction to nv is essential for this implication.

Example 3.4 Let $\mathcal{R}_6$ be the following orthogonal TRS:

$$
\begin{array}{rlrl}
\mathsf{f}(x,\mathsf{a}) & \to \mathsf{a} & \mathsf{h}(x,\mathsf{a},\mathsf{a}) & \to \mathsf{i} \\
\mathsf{f}(\mathsf{a},\mathsf{b}(x)) & \to \mathsf{i} & \mathsf{h}(x,\mathsf{a},\mathsf{b}(y)) & \to \mathsf{i} \\
\mathsf{f}(\mathsf{b}(x),\mathsf{b}(y)) & \to \mathsf{i} & \mathsf{h}(x,\mathsf{b}(y),\mathsf{a}) & \to \mathsf{i} \\
\mathsf{g}(\mathsf{a},\mathsf{a}) & \to \mathsf{i} & \mathsf{h}(x,\mathsf{b}(y),\mathsf{b}(z)) & \to \mathsf{b}(\mathsf{g}(y,\mathsf{f}(x,z))) \\
\mathsf{g}(\mathsf{b}(x),\mathsf{a}) & \to \mathsf{i} & \mathsf{i} & \to \mathsf{b}(\mathsf{i}) \\
\mathsf{g}(x,\mathsf{b}(y)) & \to \mathsf{a} & &
\end{array}
$$

over the signature $\mathcal{F}$ consisting of all symbols appearing in the rewrite rules and let $\mathcal{G} = \mathcal{F} \cup \{@\}$.

`Autowrite` is able to check

- $\mathsf{ENF}(\mathcal{R}_6) = \emptyset$,
- $\mathsf{WN}(\mathrm{gr}(\mathcal{R}_6), \mathcal{G}, \mathcal{F}) = \mathsf{WN}(\mathrm{gr}(\mathcal{R}_6), \mathcal{F})$,
- $\mathsf{WN}_\bullet(\mathrm{gr}(\mathcal{R}_6), \mathcal{G}, \mathcal{F}) \neq \mathsf{WN}_\bullet(\mathrm{gr}(\mathcal{R}_6), \mathcal{F})$ as shown by the term $t = \mathsf{h}(\bullet, \mathsf{i}, \mathsf{i})$.

4 The inside of `Autowrite`

The most important object in `Autowrite` is the tree automaton. Since the first version of `Autowrite` [5], much care has been devoted to improve the representation of automata. Consequently, the performances have improved significantly.

Each state of an automaton is represented by a unique Common Lisp object. Comparing two states is then very cheap: we just need to compare the references of the states. An automaton is represented by its signature (a list of symbols), a list of references to its states and its rules. A TRS is represented by its signature and its rules. The set of rules (of an automaton or a TRSs) is represented by a hash-table which given a key associated with a left-hand side of a rule gives the corresponding right-hand side (or a list of corresponding right-hand sides if the automaton is not deterministic). Given a left-hand side $f(q_1, \ldots, q_n)$, the corresponding key consists of a list containing the root symbol f followed by the references of the states $q_1, \ldots, q_n$.

During the construction of an automaton (for instance during the construction of the $\mathcal{C}_{\mathcal{R},\mathcal{A}}$ which recognizes the set of terms that rewrite to a term recognized by $\mathcal{A}$) the rules of an automaton may be represented as a simple list of rules. But as soon as the construction is completed, the list of rules is converted into a hash-table as described above.

In general we use as much as possible "sharing" instead of "copying" data structures and use hash-tables instead of lists. When possible we use memoizing techniques to avoid recomputing several times identical calls. The latter may explain differences of timing when the same operations are performed in different order.

5 The outside of Autowrite

5.1 Autowrite specifications

Autowrite handles a set of specifications that can be loaded interactively. A specification consists of a signature, possibly a set of variables, followed by a list of Autowrite objects. Autowrite objects are TRSs, automata, sets of terms and single terms. Figure 5.1 shows an example of such a specification. That specification defines a signature in which integers and arithmetic expressions using + and * may be represented, a TRS named R that may be used to simplify arithmetic expressions, an automaton named EVEN which recognizes the set of even integers, two sets of terms named RS (for root-stable) and T(F), and four ground terms.

```
Ops 0:0 s:1 +:2 *:2
Vars x y
TRS R
; addition
+(0,x) -> x
+(s(x),y) -> s(+(x,y))
; product
*(0,x) -> 0
*(s(x),y) -> +(*(x,y),y)

Automaton EVEN
States odd even
Final States even
Transitions
0 -> even
s(even) -> odd
s(odd) -> even

Termset RS 0 s(x)
Termset "T(F)" x

Term *(*(0,s(0)),+(0,s(0)))
Term *(o,+(0,s(0)))
Term *(*(0,s(0)),o)
Term s(s(s(0)))
```

Fig. 1. Example of an Autowrite specification

5.2 Automata operations performed by `Autowrite`

5.2.1 Checking properties of an automaton

Here are the different decision problems about an automaton that can be solved with `Autowrite`.

- Given an automaton $\mathcal{A}$: decide whether $L(\mathcal{A})$ is empty.
- Given two automata $\mathcal{A}$ and $\mathcal{B}$: decide whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$.
- Given two automata $\mathcal{A}$ and $\mathcal{B}$: decide whether $L(\mathcal{A}) = L(\mathcal{B})$.
- Given two automata $\mathcal{A}$ and $\mathcal{B}$: decide whether $L(\mathcal{A}) \cap L(\mathcal{B})$ is empty.
  For this latter operation the intersection automaton is not computed, rather we incrementally compute its accessible states and stop as soon as a final state is found.

When a property is not satisfied `Autowrite` exhibits a ground term exposing the failure.

The screenshot of Figure 5.2.1 shows operations concerning the current term and the current automaton performed after loading the specification shown in Figure 5.1. The automaton recognizing $\mathsf{NF}(\mathcal{R})$ is first computed. Then we check that the current term is not recognized by the current automaton (as it is not a normal form). Next we compute the complement of the current automaton (which recognizes reducible terms) and check that the current term is recognized by the complement automaton. Finally, we check that the complement automaton does not recognize the empty language.

5.2.2 Building new automata

Here are the different automata transformations or constructions handled by `Autowrite`.

- Given an automaton $\mathcal{A}$: compute $Det(\mathcal{A})$, the determinized version of $\mathcal{A}$.
- Given an automaton $\mathcal{A}$: compute $\mathcal{A}^c$ recognizing $L(\mathcal{A})^c$ the complement of $L(\mathcal{A})$ in the whole set of ground terms.
- Given an automaton $\mathcal{A}$: compute $Red(\mathcal{A})$, the *reduced* version of $\mathcal{A}$ *i.e.* such that every state is accessible.
- Given two automata $\mathcal{A}$ and $\mathcal{B}$: compute $\mathcal{A} \cap \mathcal{B}$.
- Given two automata $\mathcal{A}$ and $\mathcal{B}$: compute $\mathcal{A} \cup \mathcal{B}$.
- Given a set of linear terms $\mathcal{L}$: compute an automaton $\mathcal{A}_\mathcal{L}$ such that $L(\mathcal{A}_\mathcal{L}) = \{\sigma(t) \mid t \in \mathcal{L} \text{ and } \sigma \text{ is a ground substitution }\}$.

The screenshot of Figure 5.2.2 shows how to perform boolean operations using `Autowrite`. We compute the intersection of the automaton recognizing normal forms and its complement. We check that the resulting automaton recognizes the empty language.

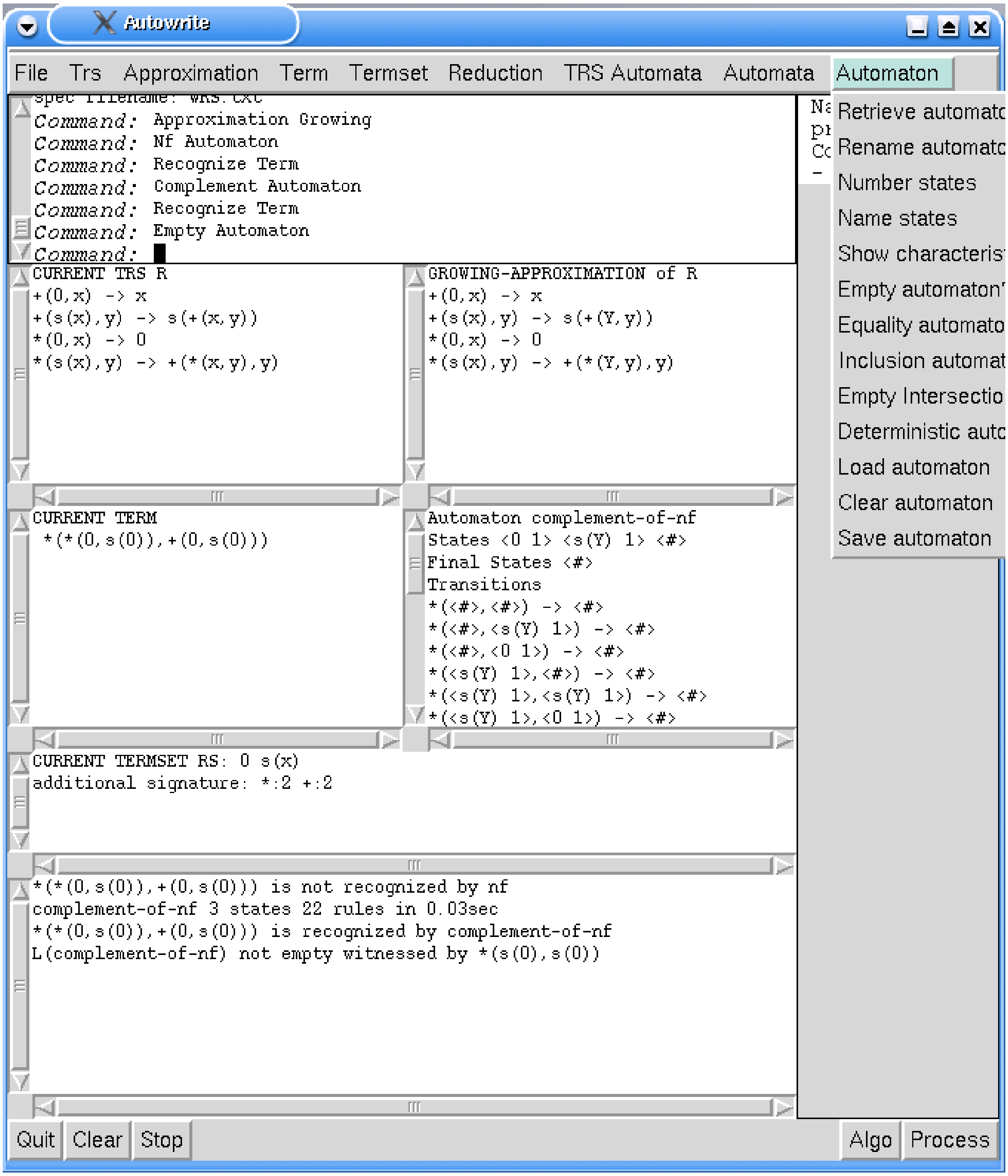

Fig. 2. Operations on the current term and the current automaton

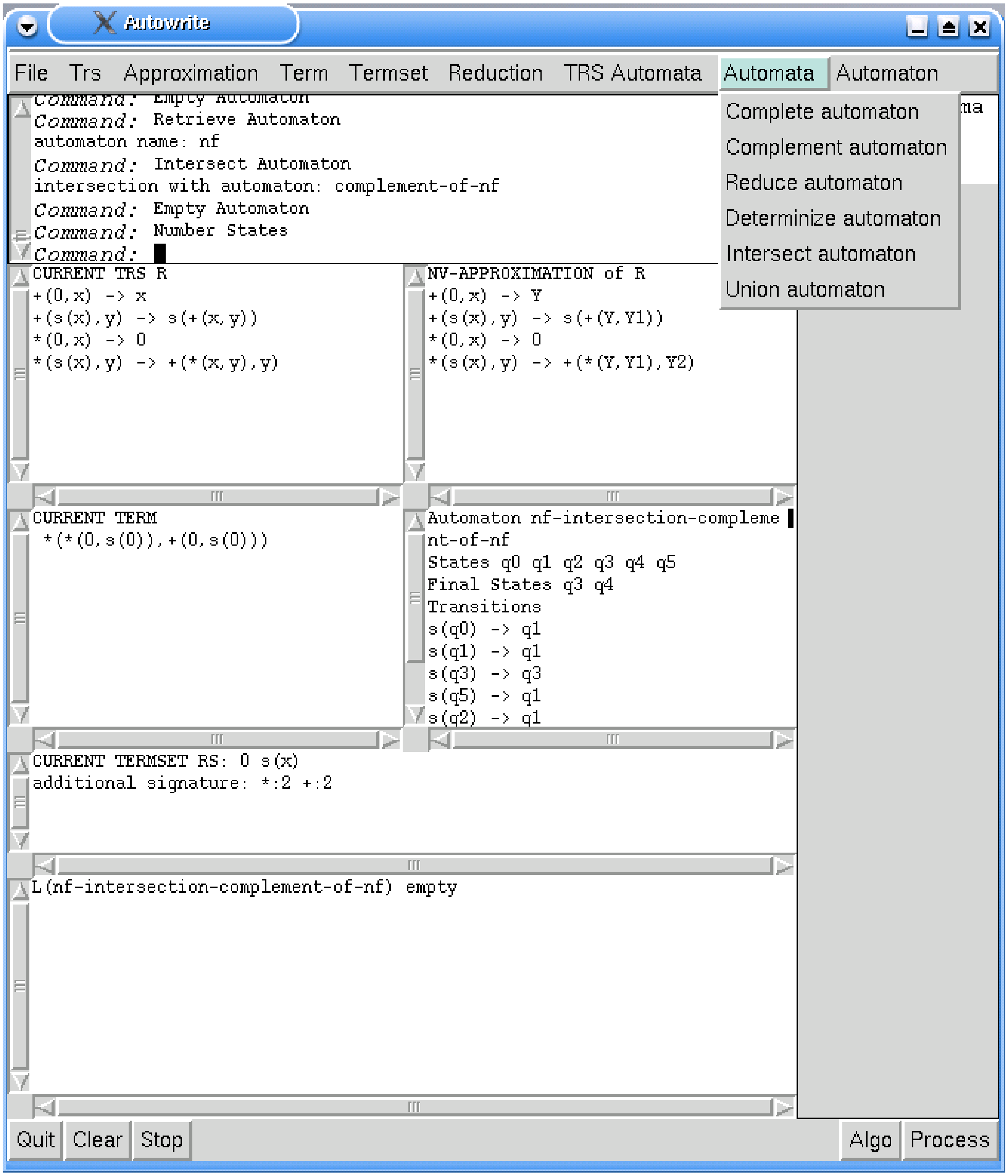

Fig. 3. Boolean operations on automata

5.3 Building automata related to a left-linear eTRSs

Let $\mathcal{R}$ be a left-linear eTRS. `Autowrite` can build the following automata:

- Build an automaton $\mathcal{A}_{\mathsf{NF}(\mathcal{R})}$ such that $L(\mathcal{A}_{\mathsf{NF}(\mathcal{R})}) = \mathsf{NF}(\mathcal{R})$.
- Build an automaton $\mathcal{A}_{\mathsf{ENF}(\mathcal{R})}$ such that $L(\mathcal{A}_{\mathsf{ENF}(\mathcal{R})}) = \mathsf{ENF}(\mathcal{R})$.

The two following automata can be constructed only if $\mathcal{R}$ also growing:

- Given a tree automaton $\mathcal{A}$: build a deterministic (Toyama and Nagaya's algorithm) or non-deterministic (Jaquemard's algorithm) automaton $\mathcal{C}_{\mathcal{R},\mathcal{A}}$ (as described in [14]) such that $L(\mathcal{C}_{\mathcal{R},\mathcal{A}}) = (\xrightarrow{*})[L(\mathcal{A})]$.
- Build an automaton $\mathcal{D}_{\mathcal{R}}$ such that $L(\mathcal{D}_{\mathcal{R}}) = \emptyset$ is equivalent to $\mathcal{R} \in \mathcal{CBN}$[6].

For $\mathcal{C}_{\mathcal{R},\mathcal{A}}$, both Jacquemard's algorithm for linear-growing TRSs and Toyama and Nagaya's for left-linear-growing TRSs have been implemented. For $\mathcal{D}_{\mathcal{R}}$, we have implemented the algorithm presented in [6]. In fact these three algorithms have been adapted in order to directly compute automata with only accessible states. This complicates the code but reduces considerably the size of the construction.

The main idea is to compute the automaton incrementally. We start building the rules having a constant left-hand side. This gives the first set of accessible states. Then we compute the rules whose left-hand sides contain the current accessible states which may give new accessible states. We stop when no new accessible state is created.

5.4 General properties of eTRS

These are easy properties but may be useful for checking big TRSs.

- Check whether a TRS is left-linear.
- Check whether a TRS is overlapping.
- Check whether a TRS is orthogonal.
- Check whether a TRS is collapsing.

5.5 Properties of left-linear eTRSs

Let $\mathcal{R}$ be a left-linear eTRS. The first set of properties concern only the left-hand sides of $\mathcal{R}$.

- Decide whether the set of normal forms is empty.
- Decide whether the set of external normal forms is empty.
  We have seen in section 3.2 that non-emptiness of $\mathsf{ENF}(\mathcal{R})$ is a sufficient condition for preserving the property that $\mathcal{R} \in \mathcal{CBN}$ when the signature is extended.

Much more interesting problems can be solved when we consider left-linear **growing** eTRSs:

- Given a tree automaton $\mathcal{A}$ and a term t, decide whether $t \in (\rightarrow^*_{\mathcal{R}})[L(\mathcal{A})]$.
  This is done by computing $\mathcal{C}_{\mathcal{R},\mathcal{A}}$ (see section 5.3) and checking whether t is recognized by $\mathcal{C}_{\mathcal{R},\mathcal{A}}$.
  Note that this solves the accessibility problem (given two terms t, s, does $t \rightarrow^*_{\mathcal{R}} s$) as a single term s forms a regular language recognizable by a tree automaton.
- Decide whether $\mathcal{R} \in \mathcal{CBN}$.
  The method consists of building the automaton $\mathcal{D}_{\mathcal{R}}$ (see section 5.3) and then check whether $L(\mathcal{D}_{\mathcal{R}}) = \emptyset$. If so `Autowrite` concludes that $\mathcal{R} \in \mathcal{CBN}$, otherwise it exhibits a ground term of $L(\mathcal{D}_{\mathcal{R}})$ which is a term with no $\mathcal{R}$-needed redex. Note that to build $\mathcal{D}_{\mathcal{R}}$, `Autowrite` must previously compute $\mathcal{C}_{\mathcal{R},\mathcal{A}_{\mathsf{NF}(\mathcal{R})}}$.
- Decide whether $\mathcal{R}$ is *arbitrary*, *i.e.* whether there exists a ground term $t \in \mathcal{T}(\mathcal{F})$ such that $t \rightarrow^*_{\mathcal{R},\mathcal{F}} \bullet$ (this means that there exists a term that may reduce to any other term).
  That latter property is relevant for the problem of signature extension (see section 3.2).

Concerning checking that $\mathcal{R} \in \mathcal{CBN}$, one cannot hope to use `Autowrite` for big TRSs because the size of the constructed automaton $\mathcal{D}_{\mathcal{R}}$ is in $\mathcal{O}(2^{2^{\|\mathcal{R}\|}})$ as shown in [6].

The screenshot of Figure 5.5 shows the use of `Autowrite` to decide membership to $\mathcal{CBN}_\alpha$ classes. We show that the current TRS $(\mathcal{R}, \mathcal{F})$ does not belong to $\mathcal{CBN}_{\mathrm{s}}$, that it belongs to $\mathcal{CBN}_{\mathrm{nv}}$ but that its extension $(\mathcal{R}, \mathcal{F}_{@})$ does not belong to $\mathcal{CBN}_{\mathrm{nv}}$.

6 Experimental results

In Table 1 (page 17), we present results obtained when testing membership of the above TRSs to $\mathcal{CBN}_\alpha$ classes with various approximations α. We present the number of states (st) and rules (rl) of the automata $\mathcal{C}_{\alpha(\mathcal{R}_\bullet)}$ and $\mathcal{D}_{\alpha(\mathcal{R})}$ built to decide whether $\mathcal{R} \in \mathcal{CBN}_\alpha$. If the TRS is not in $\mathcal{CBN}_\alpha$, we give the witness term with no $\alpha(\mathcal{R})$-needed redex found by `Autowrite`.

Table 2 (page 18) shows the results obtained for the computation of the non-deterministic automaton $\mathcal{C}_{\alpha(\mathcal{R}_\bullet)}$ with Jacquemard's algorithm which is applicable only in the linear case. The three last columns show the results given by the determinization of this automaton in order to obtain an automaton similar to the deterministic one given by Toyama and Nagaya's algorithm. "NA" means that the method is not applicable (as Jacquemard's construction is only applicable to linear TRSs).

In Table 3 (page 18), we report the results of tests of the type $\mathsf{WN}(\mathcal{R}, \mathcal{G}, \mathcal{F}) = \mathsf{WN}(\mathcal{R}, \mathcal{F})$ and $\mathsf{WN}_\bullet(\mathcal{R}, \mathcal{G}, \mathcal{F}) = \mathsf{WN}_\bullet(\mathcal{R}, \mathcal{F})$. The time needed for these computations may vary depending on the fact the automaton $\mathcal{C}_{\alpha(\mathcal{R})}$ has been computed or not by previous computations.

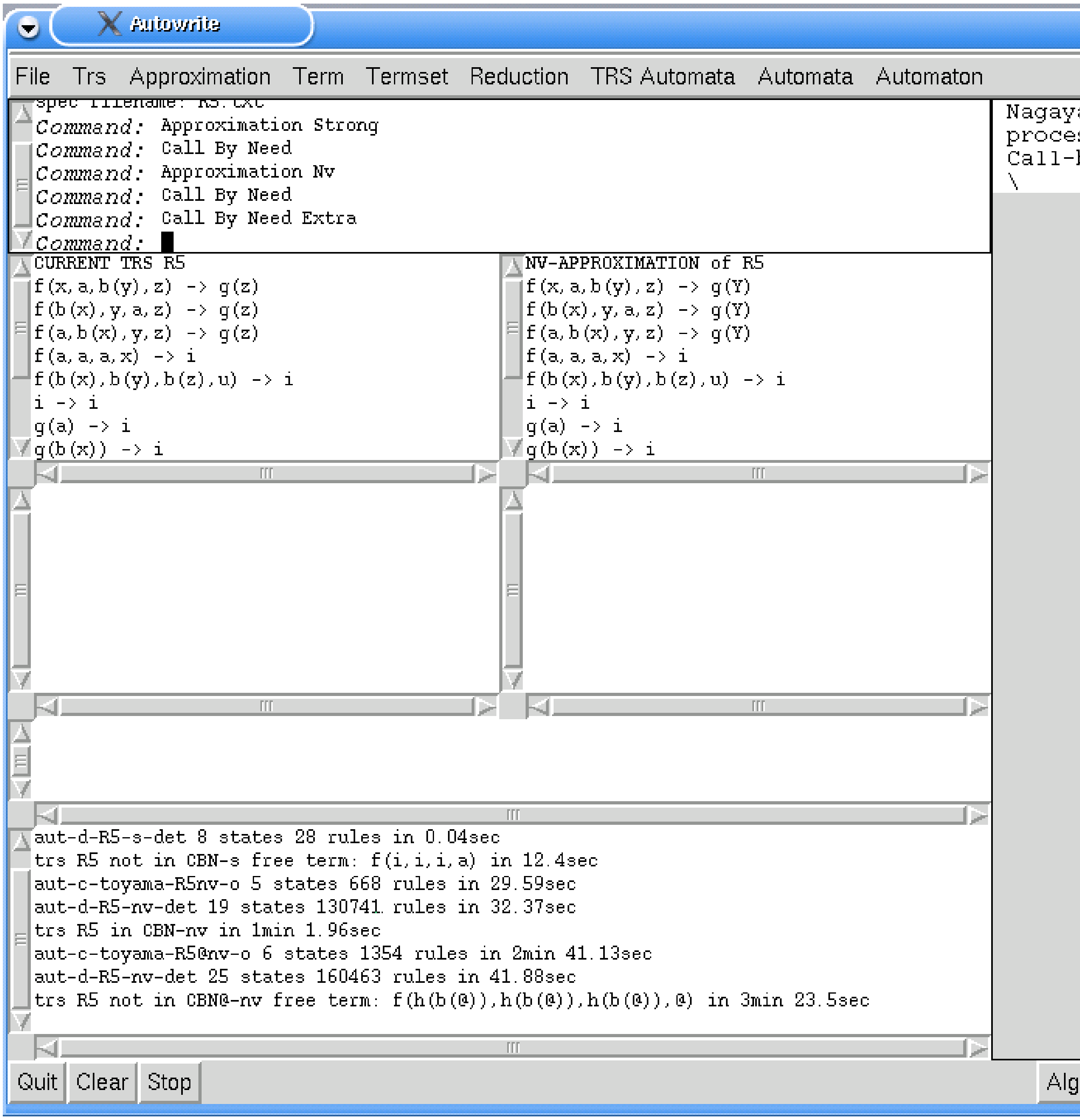

Fig. 4. Call by need queries

7 Comparison with other systems

We are aware of two other distributed tools implementing tree automata: Timbuk [9] and RX [17]. Timbuk requires the installation of `ocaml` and `RX` requires the installation of `ghc` while `Autowrite` comes self-contained. We were able able to use Timbuk (easier to install than RX). Timbuk was initially de-

Table 1
Call by need results

$\mathcal{R}$	α	$\mathcal{C}_{\alpha(\mathcal{R}_\bullet)}$ Toyama			$\mathcal{D}_{\alpha(\mathcal{R})}$			$(\mathcal{R},\mathcal{F}) \in \mathcal{CBN}_\alpha$
		st	rl	Time	st	rl	Time	
$\mathcal{R}_1, \mathcal{F}$	s	17	584	21s	32	727	0.12s	$\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{d},\mathsf{a}),\mathsf{f}(\mathsf{d},\mathsf{a})),\mathsf{f}(\mathsf{d},\mathsf{a}))$
	nv	21	888	41s	45	4055	0s76	Yes
	gr	29	1688	3m10	174	60557	48s	Yes
$\mathcal{R}_2, \mathcal{F}$	s	16	548	13s	31	687	13s	$\mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{b},\mathsf{b}),\mathsf{g}(\mathsf{b},\mathsf{b})),\mathsf{g}(\mathsf{b},\mathsf{b}))$
	nv	11	268	4s	17	615	0.08s	Yes
$\mathcal{R}_3, \mathcal{F}$	s	7	409	13s	11	162	0.03s	$\mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b}),\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b}),\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b}))$
	nv	9	831	50s	20	594	0.09	$\mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b}),\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b}),\mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b}))$
	gr	6	267	4s	17	5238	0.7s	Yes
$\mathcal{R}_4, \mathcal{F}$	s	14	3181	4m35s	12	24	0.11s	$\mathsf{f}(\mathsf{i},\mathsf{i},\mathsf{i})$
	nv	18	6537	31m36s	85	628832	7m46s	Yes
$\mathcal{R}_4, \mathcal{G}$	nv	26	19010	4h59m	115	353013	5m27s	$\mathsf{j}(\mathsf{f}(\mathsf{h}(\mathsf{g}(@)),\mathsf{h}(\mathsf{g}(@)),\mathsf{h}(\mathsf{g}(\mathsf{i}))),@)$
$\mathcal{R}_5, \mathcal{F}$	s	5	668	12s	8	28	0.04s	$\mathsf{f}(\mathsf{i},\mathsf{i},\mathsf{i},\mathsf{a})$
	nv	5	668	30s	19	130741	32s	Yes
$\mathcal{R}_5, \mathcal{G}$	nv	6	1354	2m41s	25	160463	3m24s	$\mathsf{f}(\mathsf{h}(\mathsf{b}(@)),\mathsf{h}(\mathsf{b}(@)),\mathsf{h}(\mathsf{b}(@)),@)$
$\mathcal{R}_6, \mathcal{F}$	s	5	183	1s	8	650	0.15	Yes

signed for computing over-approximations of the set of descendants $(\rightarrow^*_{\mathcal{R}})[L]$ for a regular language L and a TRS $\mathcal{R}$ and then, use it to prove unreachability. `Autowrite` can be used to check reachability, *i.e* whether $\exists t \in (\rightarrow^*_{\mathcal{R}})[L]$, but only for left-linear growing TRSs. Timbuk can handle some non-growing or non-linear cases. However, concerning efficiency of tree automata operations, `Autowrite` seems much faster: we have tried the determinization of the automaton $\mathcal{C}_{\mathrm{nv}(\mathcal{R}_5)}$ computed by Jacquemard's algorithm which runs in 0.16 seconds with `Autowrite` and took about 3 hours with Timbuk. The latest version of `Autowrite` is able to load Timbuk specifications defining TRSs, sets of terms and automata.

8 Practical information and perspectives

The `Autowrite` project has a web page:

`http://dept-info.labri.u-bordeaux.fr/~idurand/autowrite`

From that page one can download the graphical version of `Autowrite`. The file is rather big because it contains a big part of Common Lisp and McCLIM. But the advantage is that `Autowrite` is self-contained and requires no other software. The `Autowrite` sources contain about 6500 lines of Common Lisp

Table 2
Comparison between Jacquemard's and Toyama-Nagaya's automata

$\mathcal{R}$	α	$\mathcal{C}_{\alpha(\mathcal{R}_\bullet)}$ Jacquemard			$Det(\mathcal{C}_{\alpha(\mathcal{R}_\bullet)})$		
		st	rl	Time	st	rl	Time
$\mathcal{R}_1, \mathcal{F}$	s	12	343	20.84s	17	584	0.36s
	nv	12	333	19.09s	21	888	0.54s
	gr	12	201	5.14s	29	1688	1.14s
$\mathcal{R}_2, \mathcal{F}$	s	11	381	11.44s	16	548	0.27s
	nv	11	180	2.26s	11	268	0.76s
$\mathcal{R}_3, \mathcal{F}$	s	6	214	0.22s	7	409	1.34s
	nv	6	163	3.69s	9	831	0.32s
	gr	NA	NA	NA	NA	NA	NA
$\mathcal{R}_4, \mathcal{F}$	s	11	1330	1m51s	14	3181	4.26s
	nv	11	250	5.93s	18	6537	34m35s
$\mathcal{R}_5, \mathcal{F}$	s	4	287	12.95s	5	668	0.33s
	nv	4	95	1.0s	5	668	0.19s
$\mathcal{R}_6, \mathcal{F}$	s	4	126	1.52s	5	183	0.67s

Table 3
Preservation of normalizable terms by signature extension

$\mathcal{R}$	α	$\mathsf{WN}(\mathcal{R}, \mathcal{G}, \mathcal{F}) = \mathsf{WN}(\mathcal{R}, \mathcal{F})$	$\mathsf{WN}_\bullet(\mathcal{R}, \mathcal{G}, \mathcal{F}) = \mathsf{WN}_\bullet(\mathcal{R}, \mathcal{F})$	Time
$\mathcal{R}_5$	gr	Yes	Yes	2m33
$\mathcal{R}_6$	nv	Yes	Yes	21.7s
	gr	Yes	$\mathsf{h}(\bullet, \mathsf{i}, \mathsf{i})$	6.6s

(including the graphical interface). On the Web page one can find installation directives, an on-line User's Guide and useful links. The example of an `Autowrite` session should be useful for a new user. The code can still be improved for better performances. We plan to add the possibility of minimizing a tree automaton and extend the system to other classes of automata.

References

[1] G. Berry. Stable models of typed lambda-calculii. In *Proc. 5 th ICALP*, 1978.

[2] H. Comon. Sequentiality, monadic second-order logic and tree automata. *Information and Computation*, 157:25–51, 2000.

[3] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 1998. Draft, available from

http://www.grappa.univ-lille3.fr/tata/.

[4] I. Durand and A. Middeldorp. Decidable call by need computations in term rewriting (extended abstract). In *Proc. 14th CADE*, volume 1249 of *LNAI*, pages 4–18, 1997.

[5] Irène Durand. Autowrite: A tool for checking properties of term rewriting systems. In *Proceedings of the 13th International Conference on Rewriting Techniques and Applications*, volume 2378 of *Lecture Notes in Computer Science*, pages 371–375, Copenhagen, 2002. Springer-Verlag.

[6] Irène Durand and Aart Middeldorp. On the complexity of deciding call-by-need. Technical Report 1196–98, LaBRI, 1998.

[7] Irène Durand and Aart Middeldorp. On the modularity of deciding call-by-need. In *Foundations of Software Science and Computation Structures*, volume 2030 of *Lecture Notes in Computer Science*, pages 199–213, Genova, 2001. Springer-Verlag.

[8] Irène Durand and Aart Middeldorp. Decidable call-by-need computations in term rewriting. *To appear in Information and Computation*, 2004.

[9] Thomas Genêt and Valérie Viet Triem Tong. Reachability analysis of term rewriting systems with Timbuk. In *Proc. 8th LPAI*, volume 2250 of *LNAI*, pages 691–702. Springer-Verlag, 2001.

[10] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems, i and ii. In *Computational Logic, Essays in Honor of Alan Robinson*, pages 396–443. The MIT Press, 1991. Original version: Report 359, Inria, 1979.

[11] F. Jacquemard. Decidable approximations of term rewriting systems. In *Proc. 7th RTA*, volume 1103 of *LNCS*, pages 362–376, 1996.

[12] J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Vol. 2*, pages 1–116. Oxford University Press, 1992.

[13] J.W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, 12:161–195, 1991.

[14] T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems. *Information and Computation*, 178(2):499–514, 2002.

[15] M. Oyamaguchi. NV-sequentiality: A decidable condition for call-by-need computations in term rewriting systems. *SIAM Journal on Computation*, 22:114–135, 1993.

[16] Y. Toyama. Strong sequentiality of left-linear overlapping term rewriting systems. In *Proc. 7th LICS*, pages 274–284, 1992.

[17] J. Waldmann. RX: an interpreter for rational tree languages. http://www.informatik.uni-leipzig.de/~joe/rx/, 1998.

A Proof of Proposition 3.1

We recall two useful notions and a lemma from [8]. The subset of $\mathsf{WN}(\mathcal{R},\mathcal{G},\mathcal{F})$ consisting of those terms that admit a normalizing rewrite sequence in $(\mathcal{R},\mathcal{G})$ containing a root rewrite step is denoted by $\mathsf{WNR}(\mathcal{R},\mathcal{G},\mathcal{F})$. If $\mathcal{F}=\mathcal{G}$ then we just write $\mathsf{WNR}(\mathcal{R},\mathcal{F})$ or even $\mathsf{WNR}(\mathcal{R})$ if the signature is clear from the context. Given a TRS $\mathcal{R}$, a reducible term containing no $\mathcal{R}$-needed redex is called a *free* term.

Lemma A.1 *[8] Let $\mathcal{R}$ be a left-linear TRS and α an approximation mapping. Every minimal $\alpha(\mathcal{R})$-free term belongs to $\mathsf{WNR}(\alpha(\mathcal{R}))$.*

Two additional lemmas are needed for the proof of Proposition 3.1.

Lemma A.2 *Let $(\mathcal{R},\mathcal{F})$ be a left-linear eTRS. Let $\mathcal{G}\supseteq\mathcal{F}$. Let $t,s\in\mathcal{T}(\mathcal{G})$. Let $c\in\mathcal{T}(\mathcal{F})$. Let t^c and s^c be t and s where every outermost symbol in $\mathcal{G}\setminus\mathcal{F}$ have been replaced by c. If $t\rightarrow^*_{\mathcal{R},\mathcal{G}} s$ then $t^c\rightarrow^*_{\mathcal{R},\mathcal{F}} s^c$.*

Proof. Obvious as the symbols in $\mathcal{G}\setminus\mathcal{F}$ do not participate to the left-hand side of any redex. □

Lemma A.3 *Let $(\mathcal{R},\mathcal{F})$ be an eTRS. Let $\mathcal{G}\supseteq\mathcal{F}$. Let @ be a fresh constant symbol. Let $\mathcal{F}_@=\mathcal{F}\cup\{@\}$. Let $t\in\mathcal{T}(\mathcal{G})$. Let $t^@$ obtained from t by replacing every outermost subterm with root in $\mathcal{G}\setminus\mathcal{F}$ by @.*
(1) If $t\in\mathsf{NF}(\mathcal{R},\mathcal{G})$ then $t^@\in\mathsf{NF}(\mathcal{R},\mathcal{F}_@)$.
(2) If $t\in\mathsf{WN}(\mathcal{R},\mathcal{G})$ then $t^@\in\mathsf{WN}(\mathcal{R},\mathcal{F}_@)$.

Proof. (1) $t\in\mathsf{NF}(\mathcal{R},\mathcal{G})$. If $\mathsf{root}(t)\in\mathcal{G}\setminus\mathcal{F}$ then $t^@=@\in\mathsf{NF}(\mathcal{R},\mathcal{F}_@)$ otherwise $t^@$ cannot contain a redex otherwise t would also contain a redex. So (1) holds.
(2) $t\in\mathsf{WN}(\mathcal{R},\mathcal{G})$. Then $t\rightarrow^*_{\mathcal{R},\mathcal{G}} s$ for some $s\in\mathsf{NF}(\mathcal{R},\mathcal{G})$. Trivially, $t\rightarrow^*_{\mathcal{R},\mathcal{G}_@} s$ and $s\in\mathsf{NF}(\mathcal{R},\mathcal{G}_@)$. As $@\in\mathcal{F}_@$, Lemma A.2 yields $t^@\rightarrow^*_{\mathcal{R},\mathcal{F}_@} s^@$. (1) yields $s^@\in\mathsf{NF}(\mathcal{R},\mathcal{F}_@)$. It follows that $t^@\in\mathsf{WN}(\mathcal{R},\mathcal{F}_@)$. □

Proof of Proposition 3.1 (page 6). Let t be a minimal $(\alpha(\mathcal{R}),\mathcal{G})$-free term. By Lemma A.1, t is not $(\alpha(\mathcal{R}),\mathcal{G})$-root-stable. Let $t^@$ obtained from t by replacing every outermost subterm with root in $\mathcal{G}\setminus\mathcal{F}$ by @. $t^@$ is necessarily reducible otherwise t would be $(\alpha(\mathcal{R}),\mathcal{G})$-root-stable. Let p be a redex in $t^@$. p is also a redex in t and as t is free we get $t[\bullet]_p\in\mathsf{WN}(\alpha(\mathcal{R})_\bullet,\mathcal{G}_\bullet)$. By Lemma A.3, we get $t^@[\bullet]_p\in\mathsf{WN}(\alpha(\mathcal{R})_\bullet,\mathcal{F}_{@\bullet})$. We conclude that $t^@$ is a $(\alpha(\mathcal{R}),\mathcal{F}_@)$-free term so that $(\mathcal{R},\mathcal{F}_@)\notin\mathcal{CBN}_\alpha$. □