Electronic Notes in Theoretical Computer Science 124 (2005) 29–49 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Autowrite: A Tool for Term Rewrite Systems and Tree Automata

# Ir`ene Durand[1](#_bookmark0)

*LaBRI*

*Universit´e 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 graphi- cal 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 de- signed to check call-by-need properties of term rewrite systems. For this purpose, it implements the tree automata constructions used in [[11](#_bookmark22),[4](#_bookmark15),[6](#_bookmark17),[14](#_bookmark23)] and many useful operations on terms, term rewrite systems and tree automata.

*Keywords:* Tree automata, Term rewriting

# Introduction

Huet and L´evy [[10](#_bookmark21)] showed that for the class of orthogonal term rewrite sys- tems (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 approx-

imation to neededness, Huet and L´evy introduced the subclass of *strongly sequential* TRSs.

1 Email: [idurand@labri.fr](mailto:idurand@labri.fr)

1571-0661 © 2005 Elsevier B.V. Open access under [CC BY-NC-ND license](http://creativecommons.org/licenses/by-nc-nd/3.0/).

doi:10.1016/j.entcs.2004.11.019

In a strongly sequential TRS, at least one of the needed redexes in every reducible term can be effectively computed. Moreover, Huet and L´evy 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.

Several authors [[2](#_bookmark12),[4](#_bookmark15),[11](#_bookmark22),[13](#_bookmark24),[14](#_bookmark23),[15](#_bookmark25),[16](#_bookmark26)] proposed decidable extensions of the class of strongly sequential TRSs. Since Comon’s [[2](#_bookmark12)] and Jacquemard’s [[11](#_bookmark22)] work, all the corresponding decidability proofs have been expressed using tree automata techniques: typically a property is satisfied if and only if some as- sociated automaton recognizes the empty language. A uniform framework for the study of call-by-need (sequentiality) is described in [[4](#_bookmark15)] where classes of term rewrite systems are parameterized by approximation mappings. Nowa- days the best known approximation (easily definable) for which call-by-need is decidable is the growing approximation studied in [[14](#_bookmark23)]. No TRS is strong (equal to its strong approximation) because of the requirement that every vari- able 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 approxima- tion). 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](#_bookmark18)]. 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](#_bookmark16)], 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 informa- tion. 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](#_bookmark19)] for which no easy proof can be written.

# Preliminaries

We assume that the reader is familiar with the basics of term rewriting [[12](#_bookmark27)] and tree automata [[3](#_bookmark13)]. Familiarity with the theory of call-by-need [[10](#_bookmark21),[15](#_bookmark25),[11](#_bookmark22),[4](#_bookmark15)] [[6](#_bookmark17),[2](#_bookmark12),[7](#_bookmark18),[14](#_bookmark23)] is helpful.

A *term rewrite system* (TRS for short) R over a finite *signature* F (set of symbols with fixed arity) consists of *rewrite rules l* → *r* between terms in T (F *,* V) that satisfy *l* ∈*/* V and V(*r*) ⊆ V(*l*). Here 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](#_bookmark1).

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 R is denoted by NF(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](#_bookmark13)]. A (finite bottom- up) *tree automaton* is a quadruple A = (F *, Q, Qf ,* ∆) consisting of a finite signature F, a finite set *Q* of states, disjoint from F, a subset *Qf* ⊆ *Q* of final states, and a set of transition rules ∆. Every transition rule is of the

form *f* (*q*1*,... , qn*) → *q* with *f* ∈ F and *q*1*,... , qn,q* ∈ *Q* or *q* → *q*' with *q, q*' ∈ *Q*. The latter rules are called *ϵ*-transitions. So a tree automaton A = (F *, Q, Qf ,* ∆) is simply a finite ground TRS ∆ over the signature F ∪ *Q*

whose rewrite rules have a special shape, together with a subset *Qf* of *Q*. Let T (F) be the set of ground terms. A term *t* ∈ T (F) is accepted by

A if *t* →+ *q* for some *q* ∈ *Qf* . The set of all such terms is denoted by

A

*L*(A). A subset *L* ⊆ T (F) is called *regular* if there exists a tree automaton

A = (F *, Q, Qf ,* ∆) such that *L* = *L*(A). It is well-known and easy to build an automaton recognizing T (F).

* 1. *Call-by-need*

Let R• be the eTRS R∪ {• → •} over the extended signature F• = F ∪ {•}. We say that redex ∆ in *C*[∆] ∈ T (F) is R*-needed* if there is no term *t* ∈ NF(R)

such that *C*[•] →∗

R

*t*. Finally, we say that R is *in* CBN (*Call-By-Need* ) if every

reducible term in T (F) contains a R-needed redex.

Theorem 2.1 *[*[*10*](#_bookmark21)*] Let* R *be an orthogonal TRS.*

1. *Every reducible term contains a needed redex.*
2. *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).

* 1. *Approximation mappings*

Let R and S be eTRSs over the same signature. We say that S approximates R

S

if →∗

R

⊆ →∗

and NF(R) = NF(S). An approximation mapping is a mapping

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

Next we define the approximation mappings s, nv, and gr. Let R be a TRS. The *strong* approximation [[10](#_bookmark21)] s(R) is obtained from 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](#_bookmark25)] nv(R) is obtained from R by replacing the variables in the right-hand sides of the

rewrite rules by pairwise distinct variables that do not occur in the corre- sponding left-hand sides. The *growing* approximation [[11](#_bookmark22),[14](#_bookmark23)] gr(R) is ob- tained from 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 R and *α* ∈ {s*,* nv*,* gr}, it is decidable whether *α*(R) is in CBN . However the

decision procedures are very complex (exponential for s, doubly exponential for nv and gr [[6](#_bookmark17)]) so in general it is impossible to show by hand that a partic- ular TRS is in CBN , even for very small TRSs. However showing that a TRS is not in CBN can be done more easily by exhibiting a term with no needed redex.

Example 2.2 The following example is taken from [[13](#_bookmark24)].

R = ⎪⎨ f(g(*x,* a)*,* c) → *x*

⎧⎪ f(g(a*, x*)*,* b) → *x*

1

f(d*, x*) → *x*

⎪⎪⎩ g(e*,* e) → e

For R1, we obtain the following approximated TRSs:

s(R

⎧⎪ f(g(a*, x*)*,* b) → *y*

⎪⎨ f(g(*x,* a)*,* c) → *y*

⎧⎪ f(g(*x,* a)*,* b) → *y*

nv(R ) = ⎪⎨ f(g(a*, x*)*,* c) → *y*

1) =

f(d*, x*) → *y*

⎪⎪⎩ g(e*,* e) → *y*

⎧⎪ f(g(*x,* a)*,* b) → *y*

1

f(d*, x*) → *y*

⎪⎪⎩ g(e*,* e) → e

gr(R

⎪⎨ f(g(a*, x*)*,* c) → *y*

1) =

f(d*, x*) → *x*

⎪⎪⎩ g(e*,* e) → e

Example 2.3 The next example (R2) comes from [[15](#_bookmark25)].

⎧⎪ f(g(a*, x*)*,* a) → c

⎪⎪⎨ f(g(*x,* a)*,* b) → c

R2 =

f(k(a)*, x*) → c

g(b*,* b) → h(b)

⎪

⎪⎩ h(*x*) → k(*x*)

⎧⎪ f(g(a*, x*)*,* a) → c

⎪⎪⎨ f(g(*x,* a)*,* b) → c

nv(R2) =

f(k(a)*, x*) → c

gr(R2) = R2

g(b*,* b) → h(b)

⎪⎩ h(*x*) → k(*y*)

⎪

Example 2.4 The last example (R3) is an extension of Berry’s example [[1](#_bookmark14)].

⎧⎪ f(*x,* a*,* b) → h(*x*)

⎪ f(b*, x,* a) → h(*x*)

f(a*,* b*, x*) → h(*x*)

⎪⎨

R3 =

h(k(*x*)) → g(*x, x*)

⎧⎪ f(*x,* a*,* b) → h(*x*)

⎪ f(b*, x,* a) → h(*x*)

f(a*,* b*, x*) → h(*x*)

⎪⎨

gr(R3) =

h(k(*x*)) → g(*y, y*)

⎪ g(a*,* a) → g(a*,* a)

⎪ g(a*,* b) → a

⎪

⎪⎩ g(b*,* a) → b

⎪ g(a*,* a) → g(a*,* a)

⎪ g(a*,* b) → a

⎪

⎪⎩ g(b*,* a) → b

Autowrite computes *α*(R) for any approximation *α* in {s*,* nv*,* gr}.

Theorem 2.5 *The approximation mappings* s*,* nv*, and* gr *preserve recogniz- ability.*

In other words, given a recognizable tree language *L*, the set (→∗

*α*(R)

)[*L*]

of terms that *α*(R)-rewrite to a term in *L* is recognizable.

Nagaya and Toyama [[14](#_bookmark23)] proved the above result for the growing approx- imation; the tree automaton that recognizes (→∗ )[*L*] is defined as the limit of a finite saturation process. This saturation process is similar to the ones

gr

defined in Comon [[2](#_bookmark12)] and Jacquemard [[11](#_bookmark22)], but by working exclusively with deterministic tree automata, non-right-linear rewrite rules can be handled.

# Real problems solved by Autowrite

* 1. *Convince someone that* R ∈ CBN *for a given* R

It is quite easy to convince someone that a TRS is not in CBN by exhibiting a term with no R-needed redex. However convincing someone that a TRS is in 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 con- sidered. Often, in papers about Call-By-Need (or Sequentiality) the authors always prove that some R /∈ CBN but never that some R ∈ CBN . Usually, they just conjecture or say they think that the TRS is in CBN .

For TRSs of reasonable size, we can use Autowrite to comfort the reader with his intuition that a TRSs is in CBN . Also, in the search of a TRS in CBN having particular properties, we were often surprised to learn from Autowrite that the candidate TRS was not in CBN contrary to our intuition. With the term with no 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](#_bookmark25)] conjectured that the TRS R2 is nv-sequential. With Autowrite one can easily check that R2 ∈ CBN nv. This does not imply that R2 is nv-sequential as CBN nv properly includes the class of nv-sequential TRSs but shows that there exists an optimal and computable strategy for R2.

* 1. *Properties related to signature extension*

Let R be a left-linear growing eTRS. In [[7](#_bookmark18),[8](#_bookmark19)] we have studied the question whether the property that R ∈ CBN is preserved after adding new function symbols. For that problem, we need to specify the underlying signature in

our notation. We write (R*,* G) instead of just R to indicate which signature is used. We write NF(R*,* F) for the set of ground normal forms of an eTRS R over a signature F. We denote by WN(R*,* F) the set of all ground terms in T (F) that rewrite in R to a normal form in NF(R*,* F). Let F ⊆ G. We denote by WN(R*,* G*,* F) the set of terms in T (F) that have a normal form with respect to (R*,* G). We write WN•(R*,* G*,* F) for WN(R•*,* G•*,* F•).

The following proposition (whose proof can be found in the appendix) states that for (R*,* F) ∈ CBN , if (R*,* {F ∪@}) ∈ CBN (for some fresh constant @) then (R*,* G) ∈ CBN for any G such that F ⊆ G.

Proposition 3.1 *Let α be an arbitrary approximation mapping. Let* (R*,* F) *a TRS such that* R ∈ CBN *α. Let* G ⊇ F *. Let* @ *be fresh constant symbol (not in* G*). Let* F@ = F ∪ {@}*. If* (R*,* G) /∈ CBN *α then* (R*,* F@) /∈ CBN *α.*

This is why Autowrite provides the possibility of testing whether (R*,* G) ∈ CBN with G = F ∪ {@}.

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 R. The set of all ground external normal forms of a TRS R is denoted by ENF(R). ENF(R) /= ∅ is a

sufficient condition for R ∈ CBN being preserved under signature extension. When ENF(R) = ∅, orthogonality is needed in all the sufficient conditions that we have obtained.

A rewrite rule *l* → *r* is *collapsing* if *r* ∈ V and so is an eTRS containing a collapsing rule. For orthogonal nv eTRSs, the condition that R is collapsing and WN(R*,* G*,* F) = WN(R*,* F) is sufficient for having R ∈ 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 R4 be the following orthogonal TRS:

f(*x,* a*,* b(*y, z*)) → c(i) f(c(*x*)*,* c(*y*)*, z*) → i

f(*x,* a*,* c(*y*)) → i g(*x*) → b(*x,* i) f(a*,* a*,* a) → i h(a) → i

f(a*,* b(*x, y*)*, z*) → a h(b(a*, x*)) → a

f(a*,* c(*x*)*, y*) → i h(b(b(*x, y*)*, z*)) → b(i*,* i)

f(b(*x, y*)*, z,* a) → a h(b(c(*x*)*, y*)) → i

f(b(*x, y*)*,* b(*z, u*)*,* b(*v, w*)) → i h(c(*x*)) → i

f(b(*x, y*)*,* b(*z, u*)*,* c(*v*)) → i j(a*,* a) → i

f(b(*x, y*)*,* c(*z*)*,* b(*u, v*)) → i j(a*,* b(*x, y*)) → i

f(b(*x, y*)*,* c(*z*)*,* c(*u*)) → i j(a*,* c(*x*)) → i

f(c(*x*)*,* a*,* a) → i j(b(*x, y*)*, z*) → i

f(c(*x*)*,* b(*y, z*)*,* a) → i j(c(*x*)*, y*) → a

f(c(*x*)*,* b(*y, z*)*,* c(*u*)) → i i → i

f(c(*x*)*,* b(*y, z*)*,* b(*u, v*)) → i

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

Autowrite is able to check that

* ENF(R4) = ∅,
* (R4*,* F) ∈ CBN nv,
* (R4*,* G) /∈ CBN nv as shown by the term with no (nv(R4)*,* G)-needed redex

j(f(∆*,* ∆*,* ∆)*,* @) with ∆ = h(g(@)),

* WN(nv(R4)*,* G*,* F) = WN(nv(R4)*,* F).

One can verify easily that j(f(∆*,* ∆*,* ∆)*,* @) has no nv(R4)*,* G)-needed redex:

∆ = h(g(@)) →nv h(b(a*,* i)) →nv a ∆ = h(g(@)) →nv h(b(b(a*,* a)*,* i)) →nv

b(i*,* i)

j(f(•*,* ∆*,* ∆)*,* @) →nv j(f(•*,* a*,* b(i*,* i))*,* @) →nv j(c(i)*,* @) →nv a ∈ NF(R*,* G)

j(f(∆*,* •*,* ∆)*,* @) →nv j(f(b(i*,* i)*,* •*,* a)*,* @) →nv j(a*,* @) ∈ NF(R*,* G)

j(f(∆*,* ∆*,* •)*,* @) →nv j(f(a*,* b(i*,* i)*,* •)*,* @) →nv j(a*,* @) ∈ NF(R*,* G)

The next example in this section shows the necessity of the restriction to

*α* ∈ {s*,* nv}.

Example 3.3 Let R5 be the following orthogonal eTRS:

f(*x,* a*,* b(*y*)*, z*) → g(*z*) g(a) → i

f(b(*x*)*, y,* a*, z*) → g(*z*) g(b(*x*)) → i

f(a*,* b(*x*)*, y, z*) → g(*z*) h(a) → i

f(a*,* a*,* a*, x*) → i h(b(*x*)) → j(i*, x*)

f(b(*x*)*,* b(*y*)*,* b(*z*)*, u*) → i j(*x,* a) → a

i → i j(*x,* b(*y*)) → b(a)

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

Autowrite is able to check that

* ENF(R5) = ∅,
* (R5*,* F) ∈ CBN g,
* (R5*,* G) ∈*/* CBN g as shown by the term with no gr(R5)-needed redex f(∆*,* ∆*,* ∆*,* @), with ∆ = h(j(@)),
* WN(gr(R5)*,* F) = WN(gr(R5)*,* G*,* F).

Note that R is not collapsing. This is not essential, since adding the single collapsing rule k(*x*) → *x* to R does not affect any of the above properties.

For an nv eTRS R, we have the nice property that

WN(R*,* G*,* F) = WN(R*,* F) ⇒ WN•(R*,* G*,* F) = WN•(R*,* F)

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

Example 3.4 Let R6 be the following orthogonal TRS:

f(*x,* a) → a h(*x,* a*,* a) → i

f(a*,* b(*x*)) → i h(*x,* a*,* b(*y*)) → i

f(b(*x*)*,* b(*y*)) → i h(*x,* b(*y*)*,* a) → i

g(a*,* a) → i h(*x,* b(*y*)*,* b(*z*)) → b(g(*y,* f(*x, z*)))

g(b(*x*)*,* a) → i i → b(i)

g(*x,* b(*y*)) → a

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

Autowrite is able to check

* ENF(Y6) = ∅,
* WN(gr(Y6)*,* G*,* J) = WN(gr(Y6)*,* J),
* WN•(gr(Y6)*,* G*,* J) /= WN•(gr(Y6)*,* J) as shown by the term *t* = h(•*,* i*,* i).

# The inside of Autowrite

The most important object in Autowrite is the tree automaton. Since the first version of Autowrite [[5](#_bookmark16)], 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*,... , qn*), the corresponding key consists of a list containing the root symbol *f* followed by the references of the states *q*1*,... , qn*.

During the construction of an automaton (for instance during the con- struction of the CR*,*A which recognizes the set of terms that rewrite to a term recognized by 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 memo- izing techniques to avoid recomputing several times identical calls. The latter may explain differences of timing when the same operations are performed in different order.

# The outside of Autowrite

* 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](#_bookmark4) shows an example of such a specification. That specification defines a signature in which integers and arithmetic expres-

sions 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

* 1. *Automata operations performed by Autowrite*
     1. *Checking properties of an automaton*

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

* + - * Given an automaton A: decide whether *L*(A) is empty.
      * Given two automata A and B: decide whether *L*(A) ⊆ *L*(B).
      * Given two automata A and B: decide whether *L*(A) = *L*(B).
      * Given two automata A and B: decide whether *L*(A) ∩ *L*(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](#_bookmark5) shows operations concerning the current term and the current automaton performed after loading the specification shown in Figure [5.1](#_bookmark4). The automaton recognizing NF(Y) is first computed.

Then we check that the current term is not recognized by the current au- tomaton (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.

* + 1. *Building new automata*

Here are the different automata transformations or constructions handled by

Autowrite.

* + - * Given an automaton A: compute *Det*(A), the determinized version of A.
      * Given an automaton A: compute A*c* recognizing *L*(A)*c* the complement of

*L*(A) in the whole set of ground terms.

* + - * Given an automaton A: compute *Red*(A), the *reduced* version of A *i.e.*

such that every state is accessible.

* + - * Given two automata A and B: compute A∩ B.
      * Given two automata A and B: compute A∪ B.
      * Given a set of linear terms L: compute an automaton AL such that

*L*(AL) = {*σ*(*t*) | *t* ∈ L and *σ* is a ground substitution }.

The screenshot of Figure [5.2.2](#_bookmark6) 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.

* 1. *Building automata related to a left-linear eTRSs*

Let Y be a left-linear eTRS. Autowrite can build the following automata:

* Build an automaton ANF(R) such that *L*(ANF(R)) = NF(Y).

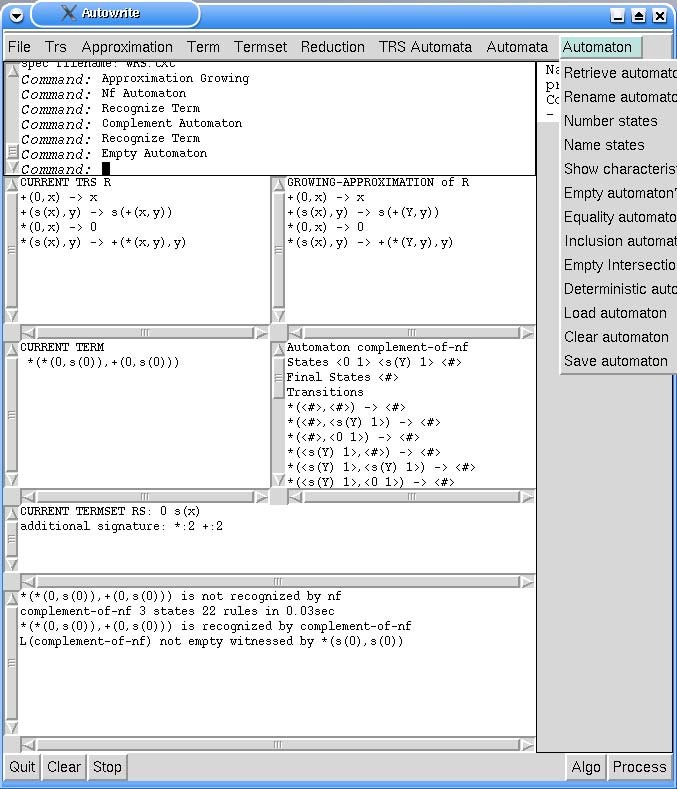


Fig. 2. Operations on the current term and the current automaton

* Build an automaton AENF(R) such that *L*(AENF(R)) = ENF(Y).

The two following automata can be constructed only if Y also growing:

* Given a tree automaton A: build a deterministic (Toyama and Nagaya’s algorithm) or non-deterministic (Jaquemard’s algorithm) automaton CR*,*A

∗

(as described in [[14](#_bookmark23)]) such that *L*(CR*,*A) = (→)[*L*(A)].

* Build an automaton DR such that *L*(DR) = ∅ is equivalent to Y ∈ CBN [[6](#_bookmark17)].

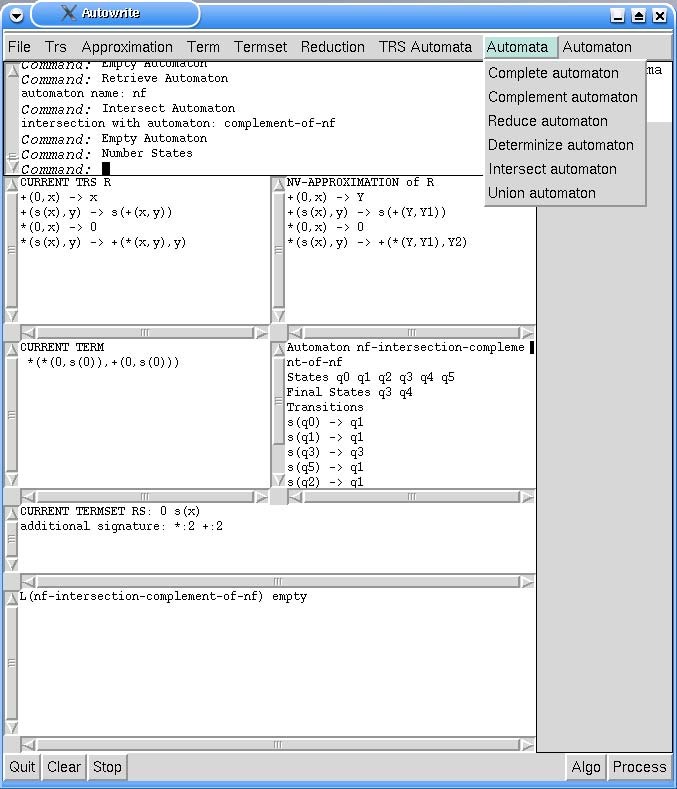


Fig. 3. Boolean operations on automata

For CR*,*A, both Jacquemard’s algorithm for linear-growing TRSs and Toyama and Nagaya’s for left-linear-growing TRSs have been implemented. For DR, we have implemented the algorithm presented in [[6](#_bookmark17)]. In fact these three algo-

rithms 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 build- ing the rules having a constant left-hand side. This gives the first set of ac- cessible 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.

* 1. *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.
  1. *Properties of left-linear eTRSs*

Let Y be a left-linear eTRS. The first set of properties concern only the left- hand sides of Y.

* 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](#_bookmark2) that non-emptiness of ENF(Y) is a sufficient condition for preserving the property that Y ∈ CBN when the signature is extended.

Much more interesting problems can be solved when we consider left-linear

growing eTRSs:

* Given a tree automaton A and a term *t*, decide whether *t* ∈ (→∗ )[*L*(A)]. This is done by computing CR*,*A (see section [5.3](#_bookmark7)) and checking whether *t* is recognized by CR*,*A.

R

Note that this solves the accessibility problem (given two terms *t, s*, does

*t* →∗

R

*s*) as a single term *s* forms a regular language recognizable by a tree

automaton.

* Decide whether Y ∈ CBN .

The method consists of building the automaton DR (see section [5.3](#_bookmark7)) and then check whether *L*(DR) = ∅. If so Autowrite concludes that Y ∈ CBN , otherwise it exhibits a ground term of *L*(DR) which is a term with no Y- needed redex. Note that to build DR, Autowrite must previously compute

CR*,*ANF(R) .

* Decide whether Y is *arbitrary*, *i.e.* whether there exists a ground term

*t* ∈ T (J) such that *t* →∗

R*,*F

reduce to any other term).

* (this means that there exists a term that may

That latter property is relevant for the problem of signature extension (see section [3.2](#_bookmark2)).

Concerning checking that Y ∈ CBN , one cannot hope to use Autowrite for big TRSs because the size of the constructed automaton D is in O(22R) as shown in [[6](#_bookmark17)].

R

The screenshot of Figure [5.5](#_bookmark8) shows the use of Autowrite to decide mem- bership to CBN *α* classes. We show that the current TRS (Y*,* J) does not belong to CBN s, that it belongs to CBN nv but that its extension (Y*,* J@) does not belong to CBN nv.

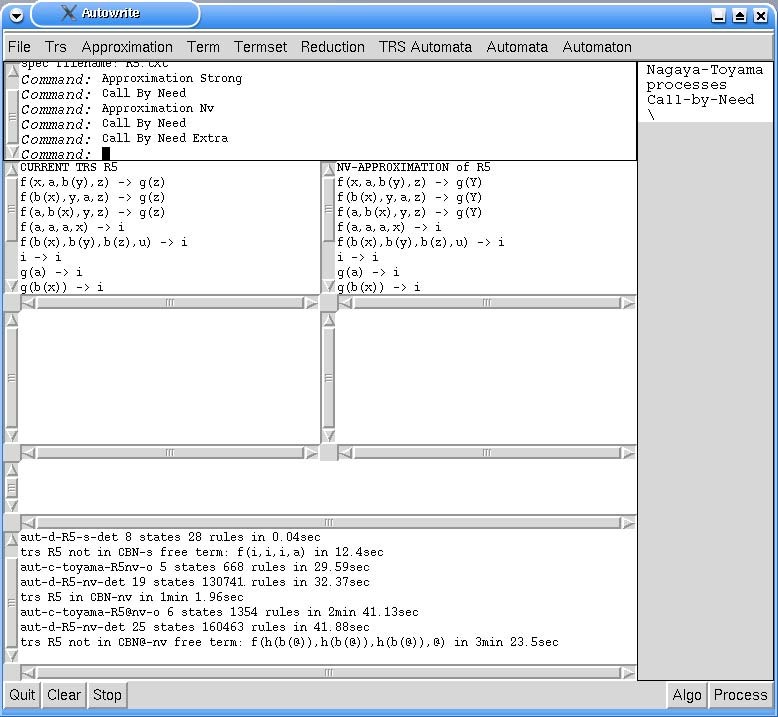


Fig. 4. Call by need queries

# Experimental results

In Table [1](#_bookmark9) (page [17](#_bookmark9)), we present results obtained when testing membership of the above TRSs to CBN *α* classes with various approximations *α*. We present

the number of states (st) and rules (rl) of the automata C*α*(R•) and D*α*(R) built

to decide whether Y ∈ CBN *α*. If the TRS is not in CBN *α*, we give the witness term with no *α*(Y)-needed redex found by Autowrite.

Table 1

Call by need results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | *α* | C*α*(R•) Toyama | | | D*α*(R) | | | (R*,* F) ∈ CBN *α* |
| st | rl | Time | st | rl | Time |
| R1 *,* F | s | 17 | 584 | 21s | 32 | 727 | 0.12s | f(g(f(d*,* a)*,* f(d*,* a))*,* f(d*,* a)) |
|  | nv | 21 | 888 | 41s | 45 | 4055 | 0s76 | Yes |
|  | gr | 29 | 1688 | 3m10 | 174 | 60557 | 48s | Yes |
| R2 *,* F | s | 16 | 548 | 13s | 31 | 687 | 13s | f(g(g(b*,* b)*,* g(b*,* b))*,* g(b*,* b)) |
|  | nv | 11 | 268 | 4s | 17 | 615 | 0.08s | Yes |
| R3 *,* F | s | 7 | 409 | 13s | 11 | 162 | 0.03s | f(f(a*,* a*,* b)*,* f(a*,* a*,* b)*,* f(a*,* a*,* b)) |
|  | nv | 9 | 831 | 50s | 20 | 594 | 0.09 | f(f(a*,* a*,* b)*,* f(a*,* a*,* b)*,* f(a*,* a*,* b)) |
|  | gr | 6 | 267 | 4s | 17 | 5238 | 0.7s | Yes |
| R4 *,* F | s | 14 | 3181 | 4m35s | 12 | 24 | 0.11s | f(i*,* i*,* i) |
|  | nv | 18 | 6537 | 31m36s | 85 | 628832 | 7m46s | Yes |
| R4 *,* G | nv | 26 | 19010 | 4h59m | 115 | 353013 | 5m27s | j(f(h(g(@))*,* h(g(@))*,* h(g(i)))*,* @) |
| R5 *,* F | s | 5 | 668 | 12s | 8 | 28 | 0.04s | f(i*,* i*,* i*,* a) |
|  | nv | 5 | 668 | 30s | 19 | 130741 | 32s | Yes |
| R5 *,* G | nv | 6 | 1354 | 2m41s | 25 | 160463 | 3m24s | f(h(b(@))*,* h(b(@))*,* h(b(@))*,* @) |
| R6 *,* F | s | 5 | 183 | 1s | 8 | 650 | 0.15 | Yes |

Table [2](#_bookmark10) (page [18](#_bookmark10)) shows the results obtained for the computation of the non-deterministic automaton C*α*(R•) with Jacquemard’s algorithm which is ap-

plicable 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](#_bookmark11) (page [18](#_bookmark11)), we report the results of tests of the type WN(Y*,* G*,* J)

= WN(Y*,* J) and WN•(Y*,* G*,* J) = WN•(Y*,* J ). The time needed for these computations may vary depending on the fact the automaton C*α*(R) has been computed or not by previous computations.

Table 2

Comparison between Jacquemard’s and Toyama-Nagaya’s automata

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Y | *α* | Cα(R•) Jacquemard | | | *Det*(Cα(R•)) | | |
| st | rl | Time | st | rl | Time |
| Y1*,* J | 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 |
| Y2*,* J | s | 11 | 381 | 11.44s | 16 | 548 | 0.27s |
|  | nv | 11 | 180 | 2.26s | 11 | 268 | 0.76s |
| Y3*,* J | 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 |
| Y4*,* J | s | 11 | 1330 | 1m51s | 14 | 3181 | 4.26s |
|  | nv | 11 | 250 | 5.93s | 18 | 6537 | 34m35s |
| Y5*,* J | s | 4 | 287 | 12.95s | 5 | 668 | 0.33s |
|  | nv | 4 | 95 | 1.0s | 5 | 668 | 0.19s |
| Y6*,* J | s | 4 | 126 | 1.52s | 5 | 183 | 0.67s |

Table 3

Preservation of normalizable terms by signature extension

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Y | *α* | WN(Y*,* G*,* J)= WN(Y*,* J) | WN•(Y*,* G*,* J)= WN•(Y*,* J) | Time |
| Y5 | gr | Yes | Yes | 2m33 |
| Y6 | nv | Yes | Yes | 21.7s |
|  | gr | Yes | h(•*,* i*,* i) | 6.6s |

# Comparison with other systems

We are aware of two other distributed tools implementing tree automata: Timbuk [[9](#_bookmark20)] and RX [[17](#_bookmark28)]. Timbuk requires the installation of ocaml and RX re- quires 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- signed for computing over-approximations of the set of descendants (→∗ )[*L*] for a regular language *L* and a TRS Y and then, use it to prove unreachabil-

R

ity. Autowrite can be used to check reachability, *i.e* whether ∃*t* ∈ (→∗ )[*L*],

R

but only for left-linear growing TRSs. Timbuk can handle some non-growing or non-linear cases. However, concerning efficiency of tree automata opera-

tions, Autowrite seems much faster: we have tried the determinization of the automaton Cnv(R5 ) 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.

# Practical information and perspectives

The Autowrite project has a web page, which can be found at the URL: <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 (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 ses- sion 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. Ir`ene 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. Ir`ene Durand and Aart Middeldorp. On the complexity of deciding call-by-need. Technical Report 1196–98, LaBRI, 1998.
7. Ir`ene 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. Ir`ene Durand and Aart Middeldorp. Decidable call-by-need computations in term rewriting.

*To appear in Information and Computation*, 2004.

1. Thomas Genˆet and Val´erie 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.
2. G. Huet and J.-J. L´evy. 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.
3. F. Jacquemard. Decidable approximations of term rewriting systems. In *Proc. 7th RTA*, volume 1103 of *LNCS*, pages 362–376, 1996.
4. J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Vol. 2*, pages 1–116. Oxford University Press, 1992.
5. J.W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal* *of Symbolic Computation*, 12:161–195, 1991.
6. T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems.

*Information and Computation*, 178(2):499–514, 2002.

1. M. Oyamaguchi. NV-sequentiality: A decidable condition for call-by-need computations in term rewriting systems. *SIAM Journal on Computation*, 22:114–135, 1993.
2. Y. Toyama. Strong sequentiality of left-linear overlapping term rewriting systems. In *Proc. 7th LICS*, pages 274–284, 1992.
3. J. Waldmann. RX: an interpreter for rational tree languages.

<http://www.informatik.uni-leipzig.de/~joe/rx/>, 1998.

# A Proof of Proposition [3.1](#_bookmark3)

We recall two useful notions and a lemma from [[8](#_bookmark19)]. The subset of WN(Y*,* G*,* J) consisting of those terms that admit a normalizing rewrite sequence in (Y*,* G) containing a root rewrite step is denoted by WNR(Y*,* G*,* J). If J = G then we just write WNR(Y*,* J) or even WNR(Y) if the signature is clear from the context. Given a TRS Y, a reducible term containing no Y-needed redex is called a *free* term.

Lemma A.1 *[*[*8*](#_bookmark19)*] Let* Y *be a left-linear TRS and α an approximation mapping. Every minimal α*(Y)*-free term belongs to* WNR(*α*(Y))*.*

Two additional lemmas are needed for the proof of Proposition [3.1](#_bookmark3).

Lemma A.2 *Let* (Y*,* J) *be a left-linear eTRS. Let* G ⊇ J *. Let t, s* ∈ T (G)*. Let c* ∈ T (J)*. Let tc and sc be t and s where every outermost symbol in* G\ J

*have been replaced by c. If t* →∗

R*,*G

R*,*F

*s then tc* →∗

*sc.*

Proof. Obvious as the symbols in G \ J do not participate to the left-hand side of any redex.

Lemma A.3 *Let* (Y*,* J) *be an eTRS. Let* G ⊇ J *. Let* @ *be a fresh constant symbol. Let* J@ = J ∪ {@}*. Let t* ∈ T (G)*. Let t*@ *obtained from t by replacing every outermost subterm with root in* G \ J *by* @*.*

1. *If t* ∈ NF(Y*,* G) *then t*@ ∈ NF(Y*,* J@)*.*
2. *If t* ∈ WN(Y*,* G) *then t*@ ∈ WN(Y*,* J@)*.*

Proof. (1) *t* ∈ NF(Y*,* G). If root(*t*) ∈ G \ J then *t*@ = @ ∈ NF(Y*,* J@) otherwise *t*@ cannot contain a redex otherwise *t* would also contain a redex. So (1) holds.

(2) *t* ∈ WN(Y*,* G). Then *t* →∗ *s* for some *s* ∈ NF(Y*,* G). Trivially, *t* →∗ *s*

R*,*G

R*,*G@

and *s* ∈ NF(Y*,* G@). As @ ∈ J@, Lemma [A.2](#_bookmark30) yields *t*@ →∗ *s*@. (1) yields

R*,*F@

*s*@ ∈ NF(Y*,* J@). It follows that *t*@ ∈ WN(Y*,* J@).

Proof of Proposition [3.1](#_bookmark3) (page [7](#_bookmark3)). Let *t* be a minimal (*α*(Y)*,* G)-free term. By Lemma [A.1](#_bookmark29), *t* is not (*α*(Y)*,* G)-root-stable. Let *t*@ obtained from *t* by replacing every outermost subterm with root in G \ J by @. *t*@ is necessarily reducible otherwise *t* would be (*α*(Y)*,* 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*[•]*p* ∈ WN(*α*(Y)•*,* G•). By Lemma [A.3](#_bookmark31), we get *t*@[•]*p* ∈ WN(*α*(Y)•*,* J@•). We conclude that *t*@ is a (*α*(Y)*,* J@)-free term so that (Y*,* J@) /∈ CBN *α*.