Autowrite: User's Guide

(Still being written)

Irène Durand

Université de Bordeaux I
33405 Talence, France
idurand@labri.fr

1 Introduction

Autowrite is a tool for handling term rewrite systems and tree automata. It was originally designed to check membership to Call-by-need (CBN) classes. For this purpose, it implements the tree automata constructions used in [7, 3, 4, 8] and many useful operations on terms, rewrite systems and tree automata. Now all these automata constructions are accessible from the graphical interface which makes Autowrite also a tool for handling tree automata.

The graphical interface (still under construction) is written using FreeCLIM, the free implementation of the CLIM specification. New functionnalities could be easily integrated upon demand.

The Autowrite tool was used to check membership to CBN_α for most of the examples presented in [5] and [6]. The latest version was presented in [2].

2 Installation

The system has been compiled to run on a PC x86 under Linux. The graphical interface is an X11 client so requires an X11 server.

The user should get the file Autowrite.tgz and install the system with the following commands.

```
tar xzvf Autowrite.tgz
cd Autowrite
./INSTALL
```

The system is started by typing

```
./autowrite &
```

Use the `Quit` button to exit Autowrite.

The directory `Autowrite` contains a `Data` directory were the specifications are stored.

3 Specification Files

The user may specify a set of Autowrite objects (Trss, automata, termsets) related to a common signature and set of variable, in a specification file whose name should have the extension `.txt`.

An Autowrite specification file starts with the definition of a signature, eventually followed by the definition of a set of variables. Next we may have in any order definitions of TRSs, automata, termsets, each one of them associated with a distinct name.

Example: `WRS.txt`

```
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)))
```

3.1 Comments

Any part of a line located after the character `;` is considered a comment and will be ignored.

3.2 Names

A name used for a symbol, a variable, a state, a trss or an automaton should not contain the following characters `;`, `:`, `(`, `)`, `"`, `-` unless the name is surrounded by `" "`. The names `o` and `@` are reserved (`o` represents the • symbol and `@` is the extra symbol used to extend a signature.

3.3 Symbols and signature

A *symbol* has a *name* and an *arity* separated by the character `:`.
A *signature* is introduced by the `Ops` keyword followed by a list of symbols.

3.4 Variables

The *variables* are introduced by the `Vars` keyword followed by a list of variables names which should be disjoint from the symbol names.

3.5 Terms

A *term* is introduced by the `Term` keyword followed by the term. Terms are represented in the usual infix notation and must be built upon the signature and the set of variables.

3.6 Trss

A *trs* is introduced by the `TRS` keyword followed by the name of the TRS and by the list of rules.

Each *rule* has a left-handside and a right-handside separated by `->`. Both the left-handside and the right-handside are terms.

3.7 Automata

An *automaton* is introduced by the `Automaton` keyword followed by the name of the automaton. Follow in that order states, final states and transitions.
The *states* are introduced by the `States` keyword followed by a list of the states names.
The *final states* are introduced by the `Final States` keyword followed by a list of the final states names.
The *transitions* are introduced by the `Transitions` keyword followed by a list of transitions. Each transition has a left-handside (a flat term with arguments being states) and a right-handside (a state) separated by `->`.

3.8 Termsets

A *termset* is introduced by the `Termset` keyword followed a list on terms (not necessarily ground).

4 Using Autowrite

4.1 Generalities

The top pane of the Autowrite window is an interactor pane from which the user interacts with the system. It prompts the user for commands and arguments.

At any time the user can get **help** about the current possibilities by clicking on the right-button.

Commands are accessible either thru the command line using **completion** (achieved with the `<TAB>` key or from the menus items. Some commands are also available from buttons.

All commands accessible are accessible by typing the command name (written on the item) to the *Command* prompt. The latter is recommended as thru completion it will save the user a lot of typing. Completion works for commands, data filenames, automaton names, trs names, termset names.

The menu are there to remind the user of every possible command or for people who don't like to type. In any case, at the moment, arguments of commands need to be typed in the interactor pane.

The **Read** commands are used to read Autowrite objects into the current specificationc directly from the command line. The **Load** commands are used to load Autowrite objects into the current specificationfrom a file. The **Save** commands are used to save Autowrite objects into a file. The **Retrieve** commands are used to set the current Autowrite object with an already computed object.

4.2 File Menu

From this menu, one can load a specification from a file or save the current specification to a file.

4.3 Trs Menu

This menu gathers all commands related to the current TRS or to the left-handsides of the current TRS.

Left-linearity One can check left-linearity for the current TRS $\mathcal{R}$ by using the *Left-linearity* command.

Overlappingness One can check overlappingness for the current TRS $\mathcal{R}$ by using the *Overlapping* command.

Orthogonality One can check orthogonality for the current TRS $\mathcal{R}$ by using the *Orthogonality* command.

Normal and external normal forms One can check whether the set of normal forms $\mathrm{NF}_{\mathcal{R}}$ or the set of external normal forms $\mathrm{ENF}_{\mathcal{R}}$ are empty using the *NF empty?* and *ENF empty?* commands. When the set is not empty, a term witnessing nonemptyness is shown.

Forward-branching One can check whether a system is forward-branching [1] by using either the cubic algorithm derived from the characterization of forward-branching system or the quadratic construction of a forward-branching index-tree. One can transform a forward-branching TRS to a constructor system in CBNs using the two algorithms presented in [9].

4.4 Approx Trs Menu

This menu gathers all commands related to the current approximation of the current TRS $\mathcal{S} = \mathcal{R}_\alpha$.

Changing the current approximation With the *Approximation Strong* *Approximation Nv* *Approximation Linear Growing* *Approximation Growing* commands, one can select the current approximation α for the current TRS $\mathcal{R}$. One may also use the *Approximation* item of the *Approx Trs* menu and select in the pop-up-menu the desired approximation.

Membership to CBN One can check membership to CBN_α for the current TRS $\mathcal{R}$ and the current approximation α by using the *Call-by-need* command.

The answer is either "the TRS is in CBN_α or "the TRS is not in CBN_α; in the latter case an "α-free-term" *i.e.* a term with no $\mathcal{R}_\alpha$-needed redex is shown.

The *Call-by-need* command checks membership to CBN_α but with the signature extended with a fresh constant symbol . Indeed a TRS may be in CBN_α with the signature consiting of all the symbols appearing in its rules but **not** in CBN_α with the same signature extended by just one fresh symbol. The preservation of membership thru "signature extension" has been investigated in [5].

Note that the membership to CBN_α problem having a double exponential time complexity, may run for a very long time for big TRS.
Use the *Stop* button to stop the computation.

Preservation of the set of normalizable terms As shown in [5], the sets of terms of $\mathcal{T}(\mathcal{F})$ that are normalizable may be augmented if the signature $\mathcal{F}$ is extended with just one fresh constant symbol (for instance). Using the command *WN(S,G,F) = WN(S,F)* one can check whether signature extension preserves the set of normalizable terms. If this is not the case, then a term witnessing that $\mathrm{WN}(\mathcal{R}_\alpha, \mathcal{G}, \mathcal{F}) \not\subseteq \mathrm{WN}(\mathcal{R}_\alpha, \mathcal{F})$ is shown.

The same can be done with $\mathcal{R} \cup \{\bullet \to \bullet\}$ as for the growing approximation we may have $\mathrm{WN}(\mathcal{R}_\alpha, \mathcal{G}, \mathcal{F}) = \mathrm{WN}(\mathcal{R}_\alpha, \mathcal{F})$ but $\mathrm{WN}_\bullet(\mathcal{R}_\alpha, \mathcal{G}, \mathcal{F}) \not\subseteq \mathrm{WN}_\bullet(\mathcal{R}_\alpha, \mathcal{F})$.
Use the command *WNo(S,G,F) = WNo(S,F)* for that purpose.

Arbitraryness The approximated TRS is *arbitrary* if there exists a term $t \in \mathcal{T}(\mathcal{F})$ such that $t \to_{\mathcal{R}_\alpha, \mathcal{F}\cup\{@\}}$ @. If the approximated TRS $\mathcal{R}_\alpha$ is arbitrary a witness term t is given. One can check whether the current approximated TRS $\mathcal{R}_\alpha$ is arbitrary by using the *Arbitrary* command of the *Approx-TRS* menu.

4.5 Term Menu

Redex positions With the *Redexes* command, one may get the redexes positions (according to the current TRS) of the current term.

Needed redex positions With the *Needed redexes* command, one may get the needed-redexes positions (according to the current approximation of the current TRS) of the current term.

Normalizability With the *Normalizability* command, one may check whether a term is normalizable with regard to the approximated TRS.
To check whether a redex in a term is needed in a term (according to the current TRS and approximation), just enter the term where the redex has been replaced with the `o` symbol and check whether it is normalizable.

Recognizability With the *Recognizability* command of the *Term* menu, one may check whether the current term is recognized by the current automaton.

4.6 Termset

A *termset* is a set of (not necessarily ground) terms built upon the current signature and set of variables.

Accessibility With the *Accessibility* command, one may check whether a termset is accessible from the current term with regard to the current approximation.
To check accessibility to a single term, just enter a termset containing that term.

Termset automaton With the *Termset automaton* command, one may get obtain an automaton recognizing the termset. With the *Accessibility* command one may check whether the set of instances of the current termset is accessible from the current term using the current approximation.

4.7 Reduction and Needed Reduction

The **Reduction** menu contains choices for reducing the current term.

Classical reduction One can perform a reduction step using either the leftmost-outermost strategy or parallel-outermost strategy. The "Reduction to normal form" command applies the parallel-outermost strategy until a normal form is reached Use the *Stop* button (or command) to stop a looping computation.

Needed reduction One can perform a needed-reduction step using the leftmost-outermost needed redex. The "Needed reduction to normal form" command applies the needed strategy until a normal form is reached (may loop if term does not have a normal form).
Use the Stop button (or command) to stop a looping computation.

4.8 TRS Automata

The *TRS Automata* menu, gathers commands to compute automata related to the left-handsides of the current TRS or to its approximation. The *Redex automaton* , *Reducible automaton* , *Nf automaton* compute the automata recognizing the sets of redexes, reducible terms, normal forms.

We have the possibilities of computing Toyama's or Jacquemard's $\mathcal{C}_{\mathcal{R}_\alpha,\mathcal{L}}$ automaton recognizing $(\rightarrow^*_{\mathcal{R}_\alpha})[\mathcal{L}]$ with either

- $\mathcal{L} = \mathrm{NF}$: command *Automaton C NF* ,
- $\mathcal{L} = L(\mathcal{A}_A)$: command *Automaton C A* ,
- $\mathcal{L} = L(\mathcal{A}_S)$: command *Automaton C S* , S being the instances of the current termset.

We can also compute the automaton $\mathcal{D}(\mathcal{R}_\alpha, \mathcal{F})$ which recognizes the set of reducible terms without $\mathcal{R}_\alpha$-needed redex: command *Automaton D* .

4.9 Automata

From this menu one can build new automata from existing ones. For instance compute the complement of the current automaton (*Complement automaton*), complete the current automaton (*Complete automaton*), compute the intersection (or the union) of the current automaton with an existing one (*Intersect automaton* , *Union automaton*).

4.10 Automaton

The operations of this menu apply to the current automaton. Most of these operations check a property of the current automaton like:

– “does it recognize the empty language?” (*Empty automaton?*),
– “is it included into another automaton” (*Inclusion automaton*),
– “is it equal (recognizes the same language) to another automaton” (*Equality automaton*),
– “is its intersection with another automaton empty” (*Intersection emptiness*).

5 How to exit if Autowrite crashes

In that case, you should end up in the Lisp debugger. Just type *(cl::quit)* to exit Lisp.

6 Example of a session

Command: ***Load Spec***
spec filename: ***WRS.txt***
Command: ***Nf Automaton***
Command: ***Redex Automaton***
Command: ***Reducible Automaton***
Command: ***Complement Automaton***
Command: ***Equality Automaton***
equality with automaton: ***nf***
Command: Retrieve automaton
automaton name: ***redex***
Command: Inclusion automaton
inclusion in automaton: ***reducible***
Command: Empty intersection?
emptyness of intersection with automaton: ***nf***
Command: Intersection automaton
Intersection with automaton: ***nf***
Command: Empty automaton?
Command: ***Load Spec***
spec filename: ***S04.txt***
Command: Call by need
Command: Approximation Nv
Command: Call by need
Command: Call by need Extra
Command: Redexes
Command: Needed redexes
Command: Automaton D
Command: Recognized by automaton
Command: Retrieve spec
spec name: ***WRS.txt***
Command: Termset automaton
Command: Accessibility automaton
Command: Needed reduction step
Command: Needed reduction to nf
Command: Retrieve term

term: ****(*(0,s(0)),+(0,s(0)))***
Command: Parallel outermost step

7 Snapshots of a session

Figures 1, 2 and 3 at the end of the document show some snapshots of the graphical interface.

References

1. I. Durand. Bounded, strongly sequential and forward-branching term rewriting systems. *Journal of Symbolic Computation*, 18:319–352, 1994.
2. I. Durand. Autowrite: A tool for term rewrite systems and tree automata. In *Workshop on Rewriting Strategies*, Electronics Notes in Theorical Computer Science, Aachen, June 2004. To appear in ENTCS.
3. I. Durand and A. Middeldorp. Decidable call by need computations in term rewriting (extended abstract). In *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 4–18, 1997.
4. I. Durand and A. Middeldorp. On the complexity of deciding call-by-need. Technical Report 1194-98, LaBRI, Université de Bordeaux I, 1998.
5. I. Durand and A. Middeldorp. On the modularity of deciding call-by-need. In *Proceedings of the Internation Conference on Foundations of Software Science and Computation Structures*, volume 2030 of *Lecture Notes in Computer Science*, pages 199–213, 2001.
6. I. Durand and Aart Middeldorp. Decidable call-by-need computations in term rewriting. *Information and Computation*, 196:95–126, 2005.
7. F. Jacquemard. Decidable approximations of term rewriting systems. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376, 1996.
8. T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems. *Information and Computation*, 178(2):499–514, 2002.
9. B. Salinier and R. Strandh. Efficient simulation of forward-branching systems. *Journal of Symbolic Computation*, 1996.

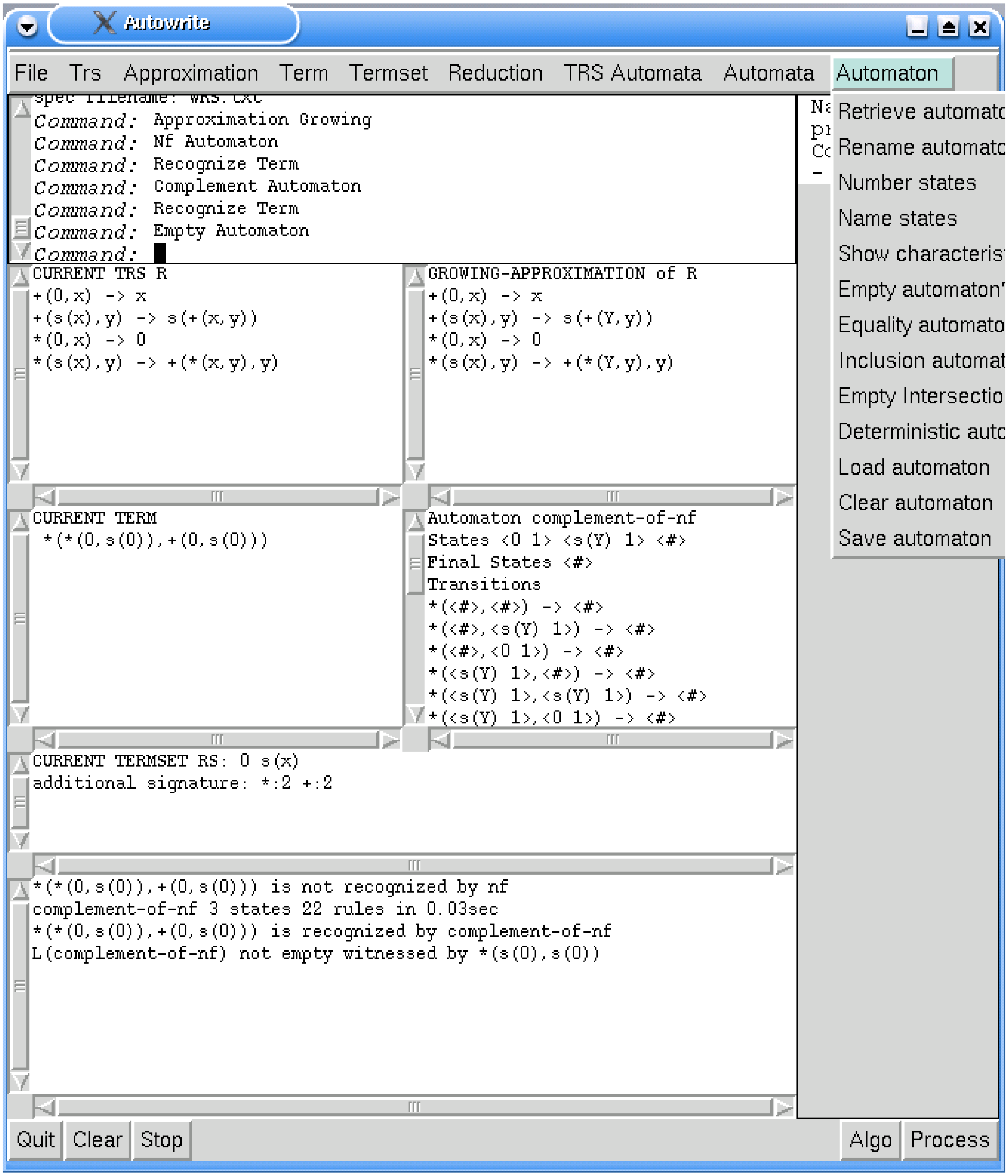

Fig. 1. Operations on the current term and the current automaton

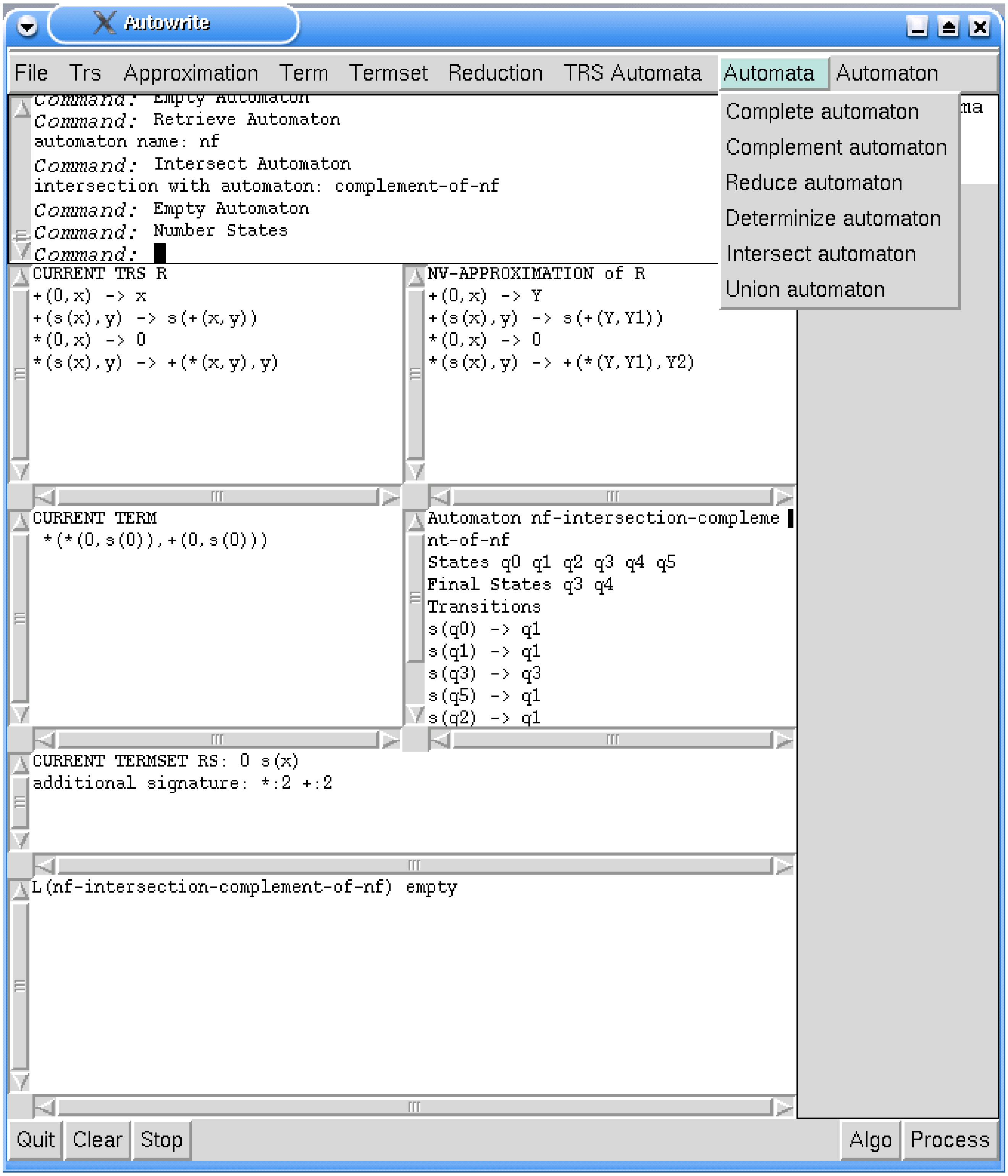

Fig. 2. Boolean operations on automata

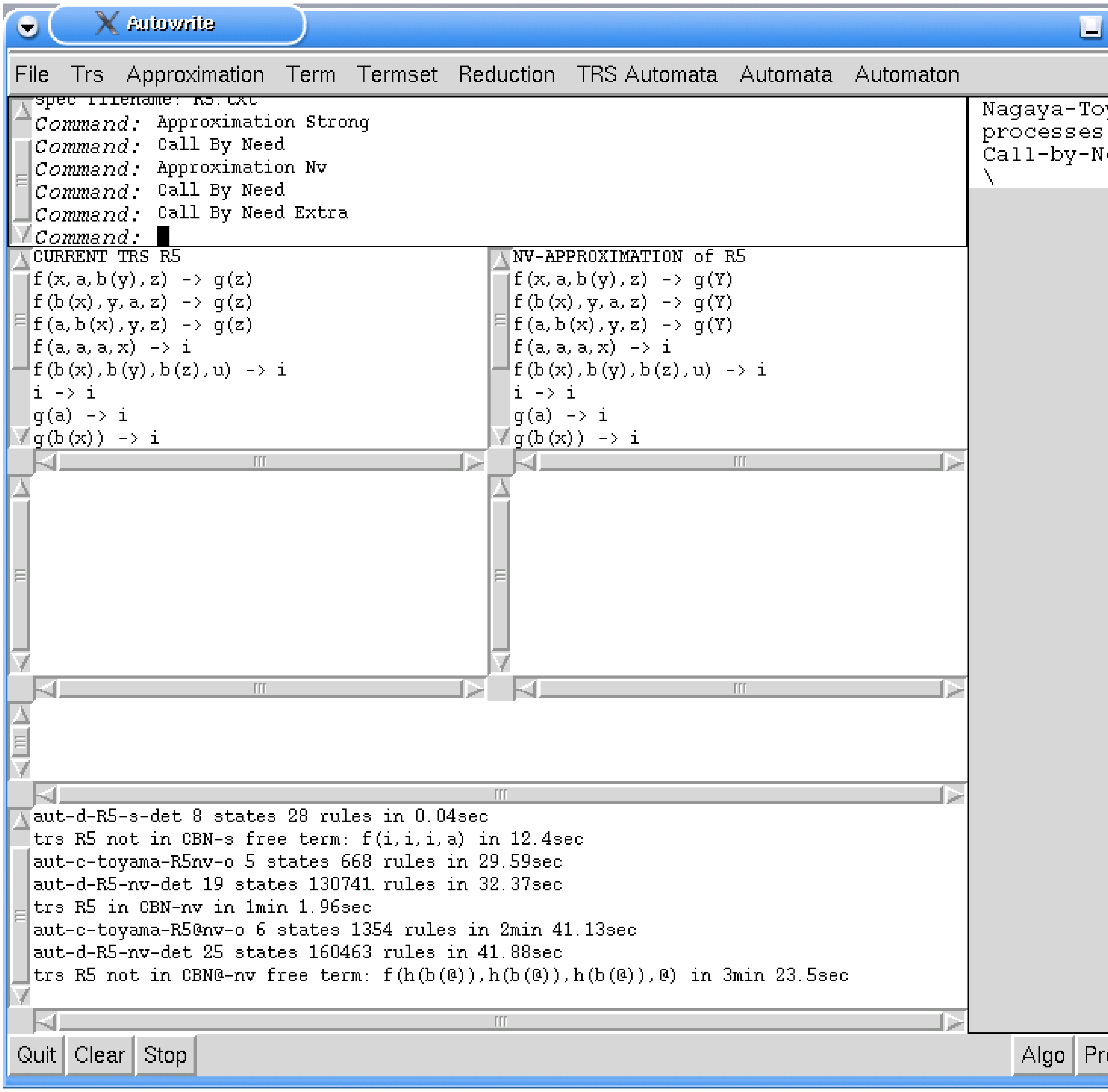

Fig. 3. Call by need queries