Autowrite: A tool for checking properties of term rewriting systems (category: system description)

Irène Durand

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

1 Introduction

Huet and Lévy [6] showed that for the class of orthogonal term rewriting 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 and showed that strong sequentiality is a decidable property of orthogonal TRSs.

Several authors [2, 3, 7, 9–12] proposed decidable extensions of the class of strongly sequential TRSs. Since Comon's [1, 2] and Jacquemard's [7] 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 sequentiality is described in [3] where classes of TRSs are parameterized by approximation mappings. Nowdays the best known approximation for which sequentiality is decidable is the growing approximation studied in [10]. We assume that the reader is familiar with the basics of term rewriting [8]. Familiarity with the theory of sequentiality ([1, 3–7, 10, 11]) is helpfuf.

Autowrite is an experimental tool written in Common Lisp for checking properties of TRSs. It was initially designed to check sequentiality properties of TRSs. For this purpose, it implements the tree automata constructions used in [7, 3, 4, 10] and many useful operations on terms, TRSs and tree automata (unfortunaletly not all yet integrated into the graphical interface). A graphical interface (still under construction) is written using FreeCLIM, the free implementation of the CLIM specification. Using this interface, one can check sequentiality of the different approximations of a given system. Many other functionalities are available directly from the Common Lisp interactive environment and could easily be integrated into the graphical interface. The Autowrite tool was used to check sequentiality for most of the examples presented in [5].

Our term rewriting systems consist of *left-linear* rewrite rules $l \to r$ that satisfy $\mathsf{root}(l) \notin \mathcal{V}$ and $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$. The following example is adapted from [9].

$$\mathcal{R}_1 = \begin{cases} \mathsf{f}(\mathsf{a}, \mathsf{g}(x_0, \mathsf{a})) & \to & \mathsf{a} \\ \mathsf{f}(\mathsf{b}, \mathsf{g}(\mathsf{a}), x_0) & \to & \mathsf{a} \\ \mathsf{f}(x_0, \mathsf{a}) & \to & x_0 \\ \mathsf{g}(\mathsf{b}, \mathsf{b}) & \to & \mathsf{a} \end{cases}$$

If the condition $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$ is not imposed, we speak of *extended* TRSs (eTRSs). Such TRSs arise naturally when we approximate TRSs, as explained below. Let $\mathcal{R}$ be an eTRS over a signature $\mathcal{F}$. The set of ground normal forms of $\mathcal{R}$ is denoted by $\mathsf{NF}(\mathcal{R})$. 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 *sequential* if every reducible term in $\mathcal{T}(\mathcal{F})$ contains a $\mathcal{R}$-needed redex.

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 α-*sequential* if $\alpha(\mathcal{R})$ is sequential. Next we define the approximation mappings s, nv, and gr. Let $\mathcal{R}$ be a TRS. The *strong* approximation $\mathrm{s}(\mathcal{R})$ 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 *nv* approximation $\mathrm{nv}(\mathcal{R})$ 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 $\mathrm{gr}(\mathcal{R})$ 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 sequential. However the decision procedures are very complex (exponential for s, doubly exponential for nv and gr [4]) so it is impossible to show by hand that a particular system is sequential, even for very small systems. However showing that a system is not sequential can be done more easily by exibiting a term with no needed redex. This later property of a term can be proved in polynomial time. For $\mathcal{R}_1$, we obtain the following approximated TRSs:

$$\mathrm{s}(\mathcal{R}_1) = \begin{cases} \mathsf{f}(\mathsf{a}, \mathsf{g}(x_0, \mathsf{a})) & \to & y_0 \\ \mathsf{f}(\mathsf{b}, \mathsf{g}(\mathsf{a}, x_0)) & \to & y_0 \\ \mathsf{f}(x_0, \mathsf{a}) & \to & y_0 \\ \mathsf{g}(\mathsf{b}, \mathsf{b}) & \to & y_0 \end{cases} \qquad \mathrm{nv}(\mathcal{R}_1) = \begin{cases} \mathsf{f}(\mathsf{a}, \mathsf{g}(x_0, \mathsf{a})) & \to & \mathsf{a} \\ \mathsf{f}(\mathsf{b}, \mathsf{g}(\mathsf{a}, x_0)) & \to & \mathsf{a} \\ \mathsf{f}(x_0, \mathsf{a}) & \to & y_0 \\ \mathsf{g}(\mathsf{b}, \mathsf{b}) & \to & \mathsf{a} \end{cases}$$

$$\mathrm{gr}(\mathcal{R}_1) = \mathcal{R}_1$$

The next example ($\mathcal{R}_2$) is adapted from [11].

$$\mathcal{R}_2 = \begin{cases} \mathsf{f}(\mathsf{g}(\mathsf{a}, x_0), \mathsf{a}) & \to & \mathsf{a} \\ \mathsf{f}(\mathsf{g}(x_0, \mathsf{a}), \mathsf{k}(\mathsf{a})) & \to & \mathsf{a} \\ \mathsf{f}(\mathsf{k}(\mathsf{c}), x_0) & \to & \mathsf{a} \\ \mathsf{g}(\mathsf{k}(\mathsf{a}), \mathsf{k}(\mathsf{a})) & \to & \mathsf{h}(\mathsf{k}(\mathsf{a})) \\ \mathsf{k}(\mathsf{k}(\mathsf{a})) & \to & \mathsf{a} \\ \mathsf{h}(x_0) & \to & \mathsf{k}(x_0) \end{cases} \qquad \mathrm{nv}(\mathcal{R}_2) = \begin{cases} \mathsf{f}(\mathsf{g}(\mathsf{a}, x_0), \mathsf{a}) & \to & \mathsf{a} \\ \mathsf{f}(\mathsf{g}(x_0, \mathsf{a}), \mathsf{k}(\mathsf{a})) & \to & \mathsf{a} \\ \mathsf{f}(\mathsf{k}(\mathsf{c}), x_0) & \to & \mathsf{a} \\ \mathsf{g}(\mathsf{k}(\mathsf{a}), \mathsf{k}(\mathsf{a})) & \to & \mathsf{h}(\mathsf{k}(\mathsf{a})) \\ \mathsf{k}(\mathsf{k}(\mathsf{a})) & \to & \mathsf{a} \\ \mathsf{h}(x_0) & \to & \mathsf{k}(y_0) \end{cases}$$

$$\mathrm{gr}(\mathcal{R}_2) = \mathcal{R}_2$$

The last example ($\mathcal{R}_3$) is adapted from Berry's example (the first three rules).

$$\mathcal{R}_3 = \begin{cases} \mathsf{f}(x_0, \mathsf{a}, \mathsf{b}) & \to \mathsf{h}(x_0) \\ \mathsf{f}(\mathsf{b}, x_0, \mathsf{a}) & \to \mathsf{h}(x_0) \\ \mathsf{f}(\mathsf{a}, \mathsf{b}, x_0) & \to \mathsf{h}(x_0) \\ \mathsf{h}(\mathsf{k}(x_0)) & \to \mathsf{g}(x_0, x_0) \\ \mathsf{g}(\mathsf{a}, \mathsf{a}) & \to \mathsf{g}(\mathsf{a}, \mathsf{a}) \\ \mathsf{g}(\mathsf{a}, \mathsf{b}) & \to \mathsf{a} \\ \mathsf{g}(\mathsf{b}, \mathsf{a}) & \to \mathsf{b} \end{cases} \qquad \mathrm{gr}(\mathcal{R}_3) = \begin{cases} \mathsf{f}(x_0, \mathsf{a}, \mathsf{b}) & \to \mathsf{h}(x_0) \\ \mathsf{f}(\mathsf{b}, x_0, \mathsf{a}) & \to \mathsf{h}(x_0) \\ \mathsf{f}(\mathsf{a}, \mathsf{b}, x_0) & \to \mathsf{h}(x_0) \\ \mathsf{h}(\mathsf{k}(x_0)) & \to \mathsf{g}(y_0, y_0) \\ \mathsf{g}(\mathsf{a}, \mathsf{a}) & \to \mathsf{g}(\mathsf{a}, \mathsf{a}) \\ \mathsf{g}(\mathsf{a}, \mathsf{b}) & \to \mathsf{a} \\ \mathsf{g}(\mathsf{b}, \mathsf{a}) & \to \mathsf{b} \end{cases}$$

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

2 Automata

Here are the different kinds of automata that Autowrite can build:

1. Given a set of ground terms $\mathcal{L}$: an automaton $\mathcal{A}_{\mathcal{L}}$ such that $L(\mathcal{A}_{\mathcal{L}}) = \mathcal{L}$.
2. Given a set of linear terms $\mathcal{L}$: an automaton $\mathcal{A}_{\mathcal{L}}$ such that $L(\mathcal{A}_{\mathcal{L}}) = \{\sigma(t) \mid t \in \mathcal{L} \text{ and } \sigma \text{ is a ground substitution}\}$.
3. Given the set of left-hand sides of a left-linear TRS $\mathcal{R}$: an automaton $\mathcal{A}_{\mathsf{NF}(\mathcal{R})}$ such that $L(\mathcal{A}_{\mathsf{NF}(\mathcal{R})}) = \mathsf{NF}(\mathcal{R})$.
4. Given any regular tree language $\mathcal{L}$ recognized by an automaton $\mathcal{A}_{\mathcal{L}}$ and a left-linear growing e-TRS $\mathcal{R}$: a deterministic automaton $\mathcal{C}_{\mathcal{R},\mathcal{L}}$ (as described in [10]) such that $L(\mathcal{C}_{\mathcal{R},\mathcal{L}}) = (\xleftarrow{*})(\mathcal{L})$.
5. Given a left-linear growing e-TRS $\mathcal{R}$: an automaton $\mathcal{D}_{\mathcal{R}}$ such that $L(\mathcal{D}_{\mathcal{R}}) = \emptyset$ is equivalent to $\mathcal{R}$ is sequential [4].

Autowrite may check the following properties about tree automata:

1. Given an automaton $\mathcal{A}$: does $L(\mathcal{A}) = \emptyset$? If not, it may exhibit ground terms belonging to the language.
2. Given a ground term t and an automaton $\mathcal{A}$: does $t \in L(\mathcal{A})$?

To decide whether a TRS $\mathcal{R}$ is α-sequential, Autowrite computes $\alpha(\mathcal{R})$ then builds the automaton $\mathcal{D}_{\alpha(\mathcal{R})}$ and finally checks whether $L(\mathcal{D}_{\alpha(\mathcal{R})}) = \emptyset$. If so it concludes that $\mathcal{R}$ is α-sequential otherwise it exhibits a ground term of $L(\mathcal{D}_{\alpha(\mathcal{R})})$ which is a term with no $\alpha(\mathcal{R})$-needed redex. Note that to build $\mathcal{D}_{\alpha(\mathcal{R})}$, Autowrite must previously compute $\mathcal{C}_{\alpha(\mathcal{R})} = \mathcal{C}_{\alpha(\mathcal{R}),\mathsf{NF}(\mathcal{R})}$. For $\mathcal{C}_{\alpha(\mathcal{R})}$, both Jacquemard's algorithm for growing-linear systems and Toyama and Nagaya's for linear-growing systems have been implemented. For $\mathcal{D}_{\alpha(\mathcal{R})}$, we have implemented the algorithm presented in [4]. In fact these three algorithms have been adapted in order to compute directly automata with only accessible states. This complicates the code but reduces considerably the size of the constructed automata.

3 Experimental results

The graphical version of Autowrite can be found at
`http://dept-info.labri.u-bordeaux.fr/~idurand/autowrite`
Autowrite has been used to check α-sequentiality for many linear TRSs. It can also check many other properties of TRSs and perform computations on tree automata; although these are not yet accessible from the graphical interface, they can be called using Lisp. However, with regard to sequentiality, 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 [4].

With Autowrite, we were able to check sequentiality for many examples found in the literature. In the following table we present a few sequentiality results for some TRSs $\mathcal{R}$ with different approximations. We present the number of states (st) and rules (rl) of the automata $\mathcal{C}_{\alpha(\mathcal{R})}$ and $\mathcal{D}_{\alpha(\mathcal{R})}$ built to decide α-sequentiality. If the TRS is not α-sequential we present the first term with no $\alpha(\mathcal{R})$-needed redex computed by Autowrite.

	α	Time in sec	$\mathcal{C}_{\alpha(\mathcal{R})}$ st	$\mathcal{C}_{\alpha(\mathcal{R})}$ rl	$\mathcal{D}_{\alpha(\mathcal{R})}$ st	$\mathcal{D}_{\alpha(\mathcal{R})}$ rl	Sequentiality results
$\mathcal{R}_1$	s	13	20	412	107	2181	f(g(b, b), g(g(b, b), g(b, b)))
	nv	101	25	742	243	7463	f(f(a, a), g(f(a, a), f(a, a)))
	gr	98	15	182	114	16477	Sequential
$\mathcal{R}_2$	s	19	24	573	109	2460	f(g(h(a), h(a)), h(a))
	nv	69	33	1329	400	4391	f(g(h(a), h(a)), h(h(a)))
	gr	145	19	292	117	17469	Sequential
$\mathcal{R}_3$	s	3	12	435	15	179	f(g(b, a), g(b, a), g(b, a))
	nv	16	14	857	27	716	f(f(b, b, a), f(b, b, a), f(b, b, a))
	gr	263	11	293	30	8187	Sequential

References

1. H. Comon. Sequentiality, second-order monadic logic and tree automata. In *Proc. 10 th LICS*, pages 508–517, 1995.
2. H. Comon. Sequentiality, monadic second-order logic and tree automata. *Information and Computation*, 1999. Full version of [1].
3. 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.
4. Irène Durand and Aart Middeldorp. On the complexity of deciding call-by-need. Technical Report 1196–98, LaBRI, 1998.
5. 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.
6. 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.
7. F. Jacquemard. Decidable approximations of term rewriting systems. In *Proc. 7th RTA*, volume 1103 of *LNCS*, pages 362–376, 1996.
8. J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Vol. 2*, pages 1–116. Oxford University Press, 1992.
9. J.W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, 12:161–195, 1991.
10. Takashi Nagaya and Yoshihito Toyama. Decidability for left-linear growing term rewriting systems. In *Proc. 10 th RTA*, volume 1631 of *LNCS*, 1999.
11. M. Oyamaguchi. NV-sequentiality: A decidable condition for call-by-need computations in term rewriting systems. *SIAM Journal on Computation*, 22:114–135, 1993.
12. Y. Toyama. Strong sequentiality of left-linear overlapping term rewriting systems. In *Proc. 7th LICS*, pages 274–284, 1992.