Termination of Computational Restrictions of Rewriting and Termination of Programs[*]

Salvador Lucas

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, E-46022 Valencia, Spain
e.mail: slucas@dsic.upv.es

Termination of rewriting [Der87,Zan03] is often proposed as a suitable theory for proving termination of programs which are executed by rewriting. Programming languages and systems whose operational principle is based on reduction (e.g., functional, algebraic, and equational programming languages as well as theorem provers based on rewriting techniques) need, however, to break down the non-determinism which is inherent to reduction relations to make computations feasible. This is usually done by means of some *reduction strategy*, i.e., a concrete rule to specify the (non-empty set of) reduction steps which can be issued on any term which is not a normal form. Thus, termination of a program $\mathcal{R}$ (where $\mathcal{R}$ is a TRS) can be more precisely defined as the *termination of the strategy* $\mathbb{S}$ which is used to execute $\mathcal{R}$. Here, by termination of a strategy $\mathbb{S}$ for a TRS $\mathcal{R}$, we mean the termination of the reduction relation $\to_{\mathbb{S}} \subseteq \to^{+}$ associated to $\mathbb{S}$.

Traditionally, the most important question about a rewriting strategy $\mathbb{S}$ is whether it is *normalizing*, i.e., no infinite $\mathbb{S}$-sequence $t \to_{\mathbb{S}} t' \to_{\mathbb{S}} \cdots$ starts from a term t having a normal form. Obviously, every rewriting strategy $\mathbb{S}$ is forced to run forever when faced to terms t having no normal form. Then, the following property is obvious.

Proposition 1. *Let $\mathcal{R}$ be a TRS and $\mathbb{S}$ be a strategy for $\mathcal{R}$. Then, $\mathbb{S}$ is terminating if and only if $\mathcal{R}$ is normalizing and $\mathbb{S}$ is normalizing.*

This proposition says that we *cannot* obtain a terminating behavior for a program $\mathcal{R}$ running under a given strategy $\mathbb{S}$ unless the program is normalizing, i.e., *every* term t has a normal form. However, many interesting programs are not normalizing. In particular, those which can be used to deal with 'infinite' data structures. For instance, the TRS that corresponds to the following Maude program:

```
fmod SEL-FIRST-PRIMES is
  sorts Nat LNat .
  op 0     : -> Nat .
  op s     : Nat -> Nat .
  ops nil primes : -> LNat .
  op cons  : Nat LNat -> LNat [strat (1 0)] .
  op sel   : Nat LNat -> Nat .
  op first : Nat LNat -> LNat .
```

```
    op nats  : Nat -> LNat .
    op sieve : LNat -> LNat .
    op filter : LNat Nat Nat -> LNat .
    vars X Y M N : Nat .
    var Z : LNat .
    eq filter(cons(X,Z),0,M) = cons(0,filter(Z,M,M)) .
    eq filter(cons(X,Z),s(N),M) = cons(X,filter(Z,N,M)) .
    eq sieve(cons(0,Z)) = sieve(Z) .
    eq sieve(cons(s(N),Z)) = cons(s(N),sieve(filter(Z,N,N))) .
    eq nats(N) = cons(N,nats(s(N))) .
    eq primes = sieve(nats(s(s(0)))) .
    eq sel(s(X),cons(Y,Z)) = sel(X,Z) .
    eq sel(0,cons(X,Z)) = X .
    eq first(0,Z) = nil .
    eq first(s(X),cons(Y,Z)) = cons(Y,first(X,Z)) .
  endfm
```

is not normalizing: the expression `primes` is intended to arbitrarily approximate the list of prime numbers (see [KdV03]) and has no normal form. As mentioned before, the problem is that rewriting strategies must rewrite terms which are not normal forms. The following notion permits us to avoid the limitation expressed by Proposition 1. In the following definition, $(A, \rightarrow)$ is an abstract reduction system (ARS), where $\rightarrow \subseteq A \times A$ for a given set A; also, for a reduction relation $\rightarrow' \subseteq A \times A$, $\mathsf{NF}_{\rightarrow'}$ is the set of all $\rightarrow'$-normal forms.

Definition 1. *Let* $(A, \rightarrow)$ *be an ARS. A binary relation* $\rightharpoonup$ *on* A *is a* computational restriction *of* $\rightarrow$ *if* $\rightharpoonup \subseteq \rightarrow^+$ *and* $\mathsf{NF}_{\rightharpoonup} \neq \mathsf{NF}_{\rightarrow}$.

When considering a TRS $\mathcal{R} = (\mathcal{F}, R)$, $\rightarrow$ is the (one-step) rewrite relation $\rightarrow_{\mathcal{R}}$ induced by $\mathcal{R}$ and we write $\mathsf{NF}_{\mathcal{R}}$ rather than $\mathsf{NF}_{\rightarrow_{\mathcal{R}}}$. Note that the condition $\mathsf{NF}_{\rightharpoonup} \neq \mathsf{NF}_{\mathcal{R}}$ (or, equivalently, $\mathsf{NF}_{\rightharpoonup} \supset \mathsf{NF}_{\mathcal{R}}$, since $\rightharpoonup \subseteq \rightarrow^+$ implies that $\mathcal{R}$-normal forms are $\rightharpoonup$-normal forms) makes the difference between the notion of rewriting strategy and that of computational restriction of rewriting:

1. every rewriting strategy $\mathbb{S}$ for a TRS $\mathcal{R}$ satisfies $\rightarrow_{\mathbb{S}} \subseteq \rightarrow^+$; however, $\mathsf{NF}_{\rightarrow_{\mathbb{S}}} = \mathsf{NF}_{\mathcal{R}}$ by definition of strategy. On the other hand,
2. every subset $\rightharpoonup$ of $\rightarrow^+$ satisfying $\mathsf{NF}_{\rightharpoonup} = \mathsf{NF}_{\mathcal{R}}$ can be just considered as a rewriting strategy $\mathbb{S}_{\rightharpoonup}$ given by

$$\mathbb{S}_{\rightharpoonup}(t) = \{t = t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n = s \mid t \rightharpoonup s\}$$

for each term t.

Using *restrictions of rewriting*, we are still able to define (restricted) strategies which only consider $\rightharpoonup$-steps. In fact, the notion of computational restriction (of rewriting) as given in Definition 1 is new but already used (as well as the corresponding strategies) in the literature.

Computational restriction	Related Strategies
Demand-driven reduction [MR92]	Demand-driven strategies [AL02]
Outermost-needed reduction [Ant92]	Outermost-needed strategy [Ant92]
Context-Sensitive Rewriting [Luc98]	Context-Sensitive Strategies [Luc02a]
”	*E*-strategies [Eke98],
”	*Just-in-time* [Pol01]
”	Non-strict evaluation [GM03]
Lazy Rewriting [FKW00,Luc02b]	On-Demand Strategies [AEGL02]
On-demand rewriting [Luc01a]	On-Demand Strategies [AEGL02,NO01]

The use of computational restrictions of rewriting in the computational model of programs is, then, interesting. For instance, the program SEL-FIRST-PRIMES above is terminating thanks to the strategy annotation $(1\ 0)$ for symbol cons which ensures that reductions on the second argument of calls to cons are not allowed. In particular, the evaluation of expression primes with SEL-FIRST-PRIME yields[1]:

```
Maude> red primes .
reduce in SEL-FIRST-PRIMES : primes .
rewrites: 3 in 0ms cpu (10ms real) (~ rewrites/second)
result LNat: cons(s(s(0)), sieve(filter(nats(s(s(s(0)))), s(0), s(0))))
```

However, we can easily obtain the fourth prime number without any risk of nontermination:

```
Maude> red sel(s(s(s(0))),primes) .
reduce in SEL-FIRST-PRIMES : sel(s(s(s(0))), primes) .
rewrites: 28 in -10ms cpu (0ms real) (~ rewrites/second)
result Nat: s(s(s(s(s(s(0))))))
```

Of course, further semantic issues should also be addressed (see [Luc03]). In this paper, we are only concerned with termination.

We argue that termination of programs could be more appropriately studied as *termination of (strategies for) computational restrictions of rewriting*.

Termination of computational restrictions of rewriting is a challenging problem. To motivate this claim, think of Lankford's theorem establishing that termination of TRSs is equivalent to the existence of a well-founded ordering $>$ on terms such that $t > s$ whenever $t \to s$. This is easily generalized to computational restrictions of rewriting $\rightharpoonup$, i.e., orderings on terms are also the basis of termination analysis of computational restrictions of rewriting. In practice, however, we only want to compare the left- and right-hand sides of the rules of the TRS by using some *reduction ordering*, i.e., a stable, monotonic, and well-founded ordering on terms. These properties correspond to well-known properties of the rewriting relation $\to$ that computational restrictions of rewriting do not need to fulfill. For instance, context-sensitive rewriting (*CSR* [Luc98]) is not monotonic; lazy rewriting (*LR* [FKW00]) and on-demand rewriting (*ODR* [Luc01a]) are not stable or monotonic. Thus, reduction orderings are not completely suitable to prove termination of computational restrictions of rewriting in many cases. Some facts and questions arise:

---

[1] We use version 1.0.5 of Maude interpreter (available at http://maude.cs.uiuc.edu/current/system/).

3

1. Generalizations of existing reduction orderings (e.g., recursive path orderings, polynomial orderings, Knuth-Bendix orderings, etc.) have already been developed in some cases. For instance, [BLR02] extends the recursive path ordering to permit its use with *CSR*; on the othe hand, [GL02b] discusses the use of polynomial orderings for proving termination of *CSR*. Are there other suitable generalizations? Is there a generic methodology for obtaining them for a given (class of) computational restrictions?
2. Transformation techniques can also be helpful here as they are able to transform proofs of termination of computational restrictions of rewriting into proofs of termination of rewriting: see [GM03,Luc02c] for *CSR*; [Luc02b] for *LR*; and [Luc01a] for *ODR*. Is there any generic transformation which could be specialized/simplified in some cases?
3. Regarding strategies, [FGK01] describes a direct technique for directly proving termination of *E*-strategies and [Luc01b] shows that proofs of termination of (innermost) *CSR* can also be used for that. In [AEGL02] termination of on-demand strategies is addressed.

The presentation will further discuss and exemplify these challenges and their possible solutions.

References

[AEGL02]   M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas. Improving On-Demand Strategy Annotations. In Matthias Baaz and Andrei Voronkov, editors, *Proc. 9th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'02*, LNAI 2514:1-18, Springer-Verlag, Berlin, 2002.

[AL02]   S. Antoy and S. Lucas. Demandness in rewriting and narrowing. Electronic Notes in Theoretical Computer Science, volume 76. Elsevier Sciences, 2002.

[Ant92]   S. Antoy. Definitional Trees. In H. Kirchner and G. Levi, editors, *Proc. of 3rd International Conference on Algebraic and Logic Programming, ALP'92*, LNCS 632:143-157, Springer-Verlag, Berlin, 1992.

[BLR02]   C. Borralleras, S. Lucas, and A. Rubio. Recursive Path Orderings can be Context-Sensitive. In A. Voronkov, editor *Proc. of 18th International Conference on Automated Deduction, CADE'02*, LNAI 2392:314-331, Springer-Verlag, Berlin, 2002.

[Der87]   N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69-115, 1987.

[Eke98]   S. Eker. Term Rewriting with Operator Evaluation Strategies. In C. Kirchner and H. Kirchner, editors, *Proc. of 2nd International Workshop on Rewriting Logic and its Applications, WRLA'98*, Electronic Notes in Computer Science, 15(1998):1-20, 1998.

[FGK01]   O. Fissore, I. Gnaedig, and H. Kirchner. Induction for termination with local strategies. *Electronic Notes in Theoretical Computer Science*, volume 58(2), 2001.

[FKW00]   W. Fokkink, J. Kamperman, and P. Walters. Lazy Rewriting on Eager Machinery. *ACM Transactions on Programming Languages and Systems*, 22(1):45-86, 2000.

[GL02b]   B. Gramlich and S. Lucas. Simple termination of context-sensitive rewriting. In B. Fischer and E. Visser, editors, *Proc. of 3rd ACM SIGPLAN Workshop on Rule-Based Programming, RULE'02*, pages 29-41, ACM Press, New York, 2002.

[GM03]   J. Giesl and A. Middeldorp. Transformation Techniques for Context-Sensitive Rewrite Systems. *Journal of Functional Programming*, to appear, 2003.

[KdV03]   J. Kennaway and F.J. de Vries. Infinitary rewriting. In TeReSe, *Term Rewriting Systems*, Chapter 11. Cambridge University Press, 2003.

[Luc98]   S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1-61, January 1998.

[Luc01a] S. Lucas. Termination of on-demand rewriting and termination of OBJ programs. In *Proc. of 3rd International Conference on Principles and Practice of Declarative Programming, PPDP'01*, pages 82-93, ACM Press, 2001.

[Luc01b] S. Lucas. Termination of Rewriting With Strategy Annotations. In R. Nieuwenhuis and A. Voronkov, editors, *Proc. of 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'01*, LNAI 2250:669-684, Springer-Verlag, Berlin, 2001.

[Luc02a] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):293-343, 2002.

[Luc02b] S. Lucas. Lazy Rewriting and Context-Sensitive Rewriting. *Electronic Notes in Theoretical Computer Science*, volume 64, Elsevier, 2002.

[Luc02c] S. Lucas. Termination of (Canonical) Context-Sensitive Rewriting. In S. Tison, editor, *Proc. of 13th International Conference on Rewriting Techniques and Applications, RTA'02*, LNCS 2378:296-310, Springer-Verlag, Berlin, 2002.

[Luc03]   S. Lucas. Semantics of programs with strategy annotations. Technical Report DSIC II/08/03, Universidad Politécnica de Valencia, April 2003.

[MR92]   J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic programming with functions and predicates: the language BABEL. *Journal of Logic Programming*, 12:191-223, 1992.

[NO01]   M. Nakamura and K. Ogata. The evaluation strategy for head normal form with and without on-demand flags. In K. Futatsugi, editor, *Proc. of 3rd International Workshop on Rewriting Logic and its Applications, WRLA'00*, *Electronic Notes in Theoretical Computer Science*, volume 36, 17 pages, 2001.

[Pol01]   J. van de Pol. Just-in-time: On Strategy Annotations. *Electronic Notes in Theoretical Computer Science*, volume 57, Elsevier, 2001.

[Zan03]   H. Zantema. Termination. In TeReSe, *Term Rewriting Systems*, Chapter 6. Cambridge University Press, 2003.