When are Two Rewrite Systems More than None?*

Nachum Dershowitz

Department of Computer Science, University of Illinois
Urbana, IL 61801, USA
nachum@uiuc.edu

Abstract. It is important for programs to have modular correctness properties. We look at non-deterministic programs expressed as term-rewriting systems (which compute normal forms of input terms) and consider the case where individual systems share constructors, but not defined symbols. We present some old and new sufficient conditions under which termination (existence of normal forms, regardless of computation strategy) and confluence (uniqueness) are preserved by such combinations.

1 Introduction

Rewriting is an important model of computation, with its clean syntax and simple semantics. Rewriting is also an important tool for equational reasoning in automated theorem proving and symbolic computation systems. Recent surveys of rewriting include [Avenhaus and Madlener, 1990; Dershowitz and Jouannaud, 1990; Klop, 1992; Plaisted, 1993].

A *rewrite system* is a set of oriented equations, called *(rewrite) rules*. We use an arrow instead of an equal sign, as in $append(nil, x) \to x$, to distinguish the left side, $append(nil, x)$, from the right side, x. A rule $l \to r$ is applied to a term t by finding a subterm s of t that matches the left side l (meaning that there exists a substitution σ of terms for variables in l such that $s = l\sigma$) and replacing s with the corresponding instance ($r\sigma$) of the rule's right side. We write $t \to t'$ to indicate that the result of the replacement is t'. One computes with rewrite systems by repeatedly, and nondeterministically, applying rules to rewrite an input term until a *normal form* (unrewritable term) is obtained. When the normal form is unique, it can be taken as the value of the initial term.

Two of the most central properties of relevance for rewrite systems are confluence (the Church-Rosser property; see Section 2)—which implies that there can be at most one normal form for any term, and termination (strong normalization in lambda calculus parlance; see Section 3)—which implies the existence of at least one normal form. A confluent and terminating system is called *convergent*

* This research was supported in part by the National Science Foundation under grants INT-95-07248 and CCR-97-00070 and was performed while on leave at the Hebrew University, Jerusalem, Israel and at École Normale Supérieure de Cachan, France.

(or *complete* or *canonical*) and defines exactly one normal form for each input term (see Section 4).

If rewriting is to be recommended as a practical programming paradigm, then it is important that one at least be able to combine two independent rewrite systems into one, and still maintain the desired properties for the combined system. Unfortunately, this is not always the case, but—as we will see—in certain more or less reasonable situations one can obtain such modularity.

For example, suppose one has a red system (over a red alphabet consisting of the defined symbol $+$)

$$\begin{array}{rcl} x+0 & \to & x \\ x+s(y) & \to & s(x+y) \end{array}$$

for adding two numbers (in successor notation, with constructors s and 0) and a blue system (with blue defined symbol *append*)

$$\begin{array}{rcl} \mathit{append}(\mathit{nil},x) & \to & x \\ \mathit{append}(\mathit{cons}(x,y),z) & \to & \mathit{cons}(x,\mathit{append}(y,z)) \end{array}$$

for appending two lists (using the list constructors *cons* and *nil*). We would like to be certain that the union of these two unrelated programs is terminating and confluent, just as its constituent systems are. That way, we could be certain that terms containing a mixture of red and blue symbols, such as

$$\mathit{append}(\mathit{cons}(s(0)+s(0),\mathit{nil}),\mathit{cons}(s(s(0))+s(0),\mathit{nil})$$

have unique normal forms. (For the purposes of this exposition, a *defined symbol* is any function symbol or constant that appears at the head of a left side and a *constructor* is any other non-variable symbol appearing in the rules.) We would like modularity to hold even in the presence of additional rules, like

$$\begin{array}{rcl} 0+x & \to & x \\ \mathit{append}(\mathit{append}(x,y),z) & \to & \mathit{append}(x,\mathit{append}(y,z)) \end{array}$$

The above red and blue systems have no symbols at all in common. In most practical situations, one would want to be able to combine the blue system with a system like:

$$\begin{array}{rcl} \mathit{interleave}(\mathit{nil},x) & \to & x \\ \mathit{interleave}(\mathit{cons}(x,y),z) & \to & \mathit{cons}(y,\mathit{interleave}(x,z)) \end{array}$$

that interleaves, rather than concatenates, two lists. Here the two list constructors appear in both programs.

In our definition of a rewrite rule we imposed no restrictions on the appearance of variables: Both $x\times 0\to 0$ and $0\to x\times 0$ are legitimate rewrite rules. Applying the latter to a term containing the constant 0 results in the replacement of that occurrence of 0 with any term of the form $u\times 0$ (u can be any term at all). A system having a rule with a variable on the right that is not also on the left, is nonterminating and likely nonconfluent. Similarly, a priori a rule could have just a variable on the left (for example, $x \to x \times 1$), in which case it is nonterminating. Since we are interested here in combinations of conceptually independent programs, we must rule out such cases from our discussions (as is indeed the convention of some authors, including [Huet, 1980]): a rule with a new variable on the right could introduce arbitrary nesting of variegated symbols; a rule with a variable for left side would apply at all positions of all terms and interfere with any other intended computation step. Accordingly, we define *constructor-sharing* pairs of rewrite systems as including only rules with nonvariable left sides and no new right side variables and for which all function symbols that appear at the top of the left side of a rule of one system are prohibited from also appearing at the top left of a rule in the conjoined system.

In the following sections, we summarize some of what is known about constructor-sharing combinations, and sketch some new results. Properties other than confluence and termination, as well as (hierarchical) combinations that share more than constructors, lie beyond the scope of this paper.

2 Confluence

The *rewrite relation* on terms, for a given system, is denoted by $\to$, its reflexive-transitive closure, called *derivability*, is $\to^*$, and $\leftrightarrow^*$ is its reflexive-symmetric-transitive closure, called *convertibility*. A system (or indeed any binary relation) is *confluent* if $s, t \to^* v$ for some v, whenever if $u \to^* s, t$. Confluence is equivalent to the *Church-Rosser* property: $s, t \to^* v$ whenever $s \leftrightarrow^* t$.

The confluence of unions of confluent relations was considered early on in [Hindley, 1964; Rosen, 1973; Staples, 1975].

In the following circumstances, it is known that the union of two confluent systems is confluent:

(a) The systems are both *left-linear* (that is, no variable appears more than once on the left side) [Raoult and Vuillemin, 1980].
(b) There are no shared constructors [Toyama, 1987b].
(c) Both systems are *bright* (meaning that the right-hand side of each rule is a defined symbol, not a variable or constructor) [Ohlebusch, 1994a].
(d) Each system is *normalizing* (in the sense that every term has at least one normal form) [Ohlebusch, 1994a].
(e) One system is terminating and left-linear and the other is bright [Dershowitz, 1997].

(This list and those in the sequel omit some known conditions that involve undecidable properties of the union.)

The necessity of these conditions may be seen from the following standard example [Huet, 1980]:

$$\begin{array}{rcl} g(x,x) & \to & 0 \\ g(x,c(x)) & \to & 1 \\ \hline a & \to & c(a) \end{array} \tag{A}$$

The upper part is not left-linear; the lower is not normalizing; c is a shared constructor; neither is bright.

A careful analysis of why modularity fails [Dershowitz *et al.*, 1997] shows that at the crux of the problem lie certain instances $s\sigma$ and $t\tau$ of terms s and t appearing in left sides of one system such that $t\tau$ contains $s\sigma$ as a subterm, but no other defined symbols. If $s\sigma \leftrightarrow^* t\tau$ holds in the union, but not in the one system alone, then confluence is not guaranteed. The above results follow from this observation.

3 Termination

A rewrite system (or any binary relation) is *terminating* if there are no infinite derivations $t_1 \rightarrow t_2 \rightarrow \cdots$.

Modularity of termination was considered in [Dershowitz, 1981].

In the following circumstances, it is known that the union of two constructor-sharing terminating systems is terminating:

(a) One system is left-linear; the other is right linear (no variable appears more than once on the right side) and bright [Bachmair and Dershowitz, 1986].

(b) The systems are each *finitely-branching* (no term rewrites in one step to infinitely many terms) and remain terminating when combined with the (non-confluent, nonbright) system $\{h(x,y) \rightarrow x, h(x,y) \rightarrow y\}$ (for new function symbol h) [Gramlich, 1994].

(c) The systems do not share constructors and each remains terminating when combined with $\{h(x,y) \rightarrow x, h(x,y) \rightarrow y\}$ (for new function symbol h) [Ohlebusch, 1994b].

(d) Both systems bright [Gramlich, 1994; Ohlebusch, 1994b].

(e) The systems are both *non-duplicating* (that is, each rule's right side contains no more occurrences of any variable than does the left) [Dershowitz, 1995; Ohlebusch, 1994b].

(f) One of the systems is both bright and non-duplicating [Dershowitz, 1995; Ohlebusch, 1994b].

The necessity of most of these conditions can be seen from the following nonterminating union [Toyama, 1987a]:

$$\begin{array}{rcl} g(x,y) & \rightarrow & x \\ g(x,y) & \rightarrow & y \\ \hline f(0,1,x) & \rightarrow & f(x,x,x) \end{array} \tag{B}$$

Its upper half is not bright; its lower half duplicates x, is not right linear, and is nonterminating when conjoined with the rules for h.

4 Convergence

A convergent system is one that is both terminating and confluent. Confluence of the union follows from termination of the union by Knuth's Critical Pair Lemma [Knuth and Bendix, 1970], so one needs to find conditions under which termination is preserved for confluent systems. Modularity of convergence was investigated in [Bidoit, 1981].

In the following circumstances, it is known that the union of two constructor-sharing convergent systems is convergent:

(a) For each system no left side unifies with a proper subterm of any left side (with variables of the two sides considered disjoint) [Gramlich, 1992; Dershowitz, 1995].

(b) They have no shared constructors and both are left-linear [Toyama *et al.*, 1995].

(c) One is *constructor-based* (proper subterms of left sides do not contain defined symbols) and left-linear [Dershowitz, 1997].

The case when both are constructor-based [Middeldorp and Toyama, 1993] follows from (a).

Even without shared constructors, modularity fails in general (as seen, for example, from the following nonterminating combination due to [Drosten, 1989]):

$$\begin{array}{rcl} g(x,x,y) & \to & y \\ g(x,y,y) & \to & x \\ \hline f(a,b,x) & \to & f(x,x,x) \\ f(x,y,z) & \to & 0 \\ a & \to & 0 \\ b & \to & 0 \end{array} \tag{C}$$

The upper part is not left-linear; the lower part is not constructor-based and a and b appear as proper subterms on its left.

If the union is nonterminating, then there is an infinite derivation with minimal *rank* (alternation of colors of symbols along a path from root to leaf) with infinitely many rewrites in the *cap* (topmost maximal monochrome context). Thus, subterms of lesser rank are terminating. To show termination of the union, we need to find a transformation of the *alien* terms (subterms below the cap) such that a rewrite in the cap can be mirrored by a rewrite of transformed terms and such that a rewrite below the cap does not affect the transformation. Variations on this approach lead to the above results. Using the idea of [Marchiori, 1995] for proving (b), one can extend the modularity of confluence to some constructor-sharing unions of left-linear systems.

References

[Avenhaus and Madlener, 1990] Jürgen Avenhaus and Klaus Madlener. Term rewriting and equational reasoning. In R. B. Banerji, editor, *Formal Techniques in Artificial Intelligence: A Sourcebook*, pages 1–41. Elsevier, Amsterdam, 1990.

[Bachmair and Dershowitz, 1986] Leo Bachmair and Nachum Dershowitz. Commutation, transformation, and termination. In J. H. Siekmann, editor, *Proceedings of the Eighth International Conference on Automated Deduction (Oxford, England)*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20, Berlin, July 1986. Springer-Verlag.

[Bidoit, 1981] Michel Bidoit. *Une méthode de présentation de types abstraits: Applications*. PhD thesis, Université de Paris-Sud, Orsay, France, June 1981. Rapport 3045.

[Dershowitz and Jouannaud, 1990] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter 6, pages 243–320. North-Holland, Amsterdam, 1990.

[Dershowitz *et al.*, 1997] Nachum Dershowitz, Maribel Fernández, and Jean-Pierre Jouannaud. Modular confluence revisited: The constructor-sharing case, 1997. In preparation.

[Dershowitz, 1981] Nachum Dershowitz. Termination of linear rewriting systems (preliminary version). In *Proceedings of the Eighth International Colloquium on Automata, Languages and Programming (Acre, Israel)*, volume 115 of *Lecture Notes in Computer Science*, pages 448–458, Berlin, July 1981. European Association of Theoretical Computer Science, Springer-Verlag.

[Dershowitz, 1995] Nachum Dershowitz. Hierarchical termination. In N. Dershowitz and N. Lindenstrauss, editors, *Proceedings of the Fourth International Workshop on Conditional and Typed Rewriting Systems (Jerusalem, Israel, July 1994)*, volume 968 of *Lecture Notes in Computer Science*, pages 89–105, Berlin, 1995. Springer-Verlag.

[Dershowitz, 1997] Nachum Dershowitz. Innocuous constructor-sharing combinations. In H. Comon, editor, *Proceedings of the Eighth International Conference on Rewriting Techniques and Applications (Sitges, Spain)*, number 1232 in Lecture Notes in Computer Science, pages 203–216, Berlin, June 1997. Springer-Verlag.

[Drosten, 1989] K. Drosten. *Termersetzungssysteme*. PhD thesis, Universitat Passau, Passau, Germany, 1989. Informatik Fachberichte 210, Springer-Verlag (Berlin).

[Gramlich, 1992] Bernhard Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. In A. Voronkov, editor, *Proceedings of the Conference on Logic Programming and Automated Reasoning (St. Petersburg, Russia)*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 285–296, Berlin, July 1992. Springer-Verlag.

[Gramlich, 1994] Bernhard Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.

[Hindley, 1964] J. Roger Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, 1964.

[Huet, 1980] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. of the Association for Computing Machinery*, 27(4):797–821, October 1980.

[Klop, 1992] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–117. Oxford University Press, Oxford, 1992.

[Knuth and Bendix, 1970] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, U. K., 1970. Reprinted in *Automation of Reasoning 2*, Springer-Verlag, Berlin, pp. 342–376 (1983).

[Marchiori, 1995] Massimo Marchiori. Modularity of completeness revisited. In *Proceedings of the Sixth International Conference on Rewriting Techniques and Applications (Kaiserslautern, Germany)*, volume 914 of *Lecture Notes in Computer Science*, pages 2–10, Berlin, April 1995. Springer-Verlag.

[Middeldorp and Toyama, 1993] Aart Middeldorp and Yoshihito Toyama. Completeness of combinations of constructor systems. *J. Symbolic Computation*, 15:331–348, 1993.

[Ohlebusch, 1994a] Enno Ohlebusch. On the modularity of confluence of constructor-sharing term rewriting systems. In S. Tison, editor, *Proceedings of the Nineteenth International Colloquium on Trees in Algebra and Programming (Edinburgh, UK)*, volume 787 of *Lecture Notes in Computer Science*, pages 262–275, Berlin, April 1994. Springer-Verlag.

[Ohlebusch, 1994b] Enno Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136(2):333–360, December 1994.

[Plaisted, 1993] David A. Plaisted. Term rewriting systems. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, chapter 2. Oxford University Press, Oxford, 1993. To appear.

[Raoult and Vuillemin, 1980] Jean-Claude Raoult and Jean Vuillemin. Operational and semantic equivalence between recursive programs. *J. of the Association for Computing Machinery*, 27(4):772–796, October 1980.

[Rosen, 1973] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *J. of the Association for Computing Machinery*, 20(1):160–187, January 1973.

[Staples, 1975] John Staples. Church-Rosser theorem for replacement systems. In J. N. Crossley, editor, *Algebra and Logic: 1974 Summer Research Institute of the Australian Mathematical Society*, volume 450 of *Lecture Notes in Mathematics*, pages 291–307, Berlin, West Germany, 1975. Springer-Verlag.

[Toyama *et al.*, 1995] Yoshihito Toyama, Jan Willem Klop, and Hendrik Pieter Barendregt. Termination for direct sums of left-linear complete term rewriting systems. *J. of the Association for Computing Machinery*, 42(6):1275–1304, November 1995.

[Toyama, 1987a] Yoshihito Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[Toyama, 1987b] Yoshihito Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *J. of the Association for Computing Machinery*, 34(1):128–143, January 1987.