Termination Modulo Equations by Abstract Commutation with an Application to Iteration*

Wan Fokkink & Hans Zantema
Utrecht University
Departments of Philosophy and of Computer Science
Utrecht, The Netherlands
`fokkink@phil.ruu.nl` `hansz@cs.ruu.nl`

Abstract

We generalize a termination theorem in term rewriting, based on an abstract commutation technique, to rewriting modulo equations. This result is applied in the setting of process algebra with iteration.

1 Introduction

Term rewriting is often applied in the setting of process algebra, see [11, 12, 7, 20] for examples in the process algebra ACP. In general, a complicating factor of rewriting in process algebra is that it means rewriting modulo equations, namely, modulo commutativity and associativity of the binary operator $+$, which represents alternative composition. For example, proving termination modulo equations is still a relatively unexplored area. It has been studied mostly for path orderings [13, 23, 6, 5, 22, 25, 31, 3, 14]; the majority of these papers only deal with rewriting modulo AC. Furthermore, Ferreira [15, 16] extended the technique of dummy elimination for proving termination from Ferreira and Zantema [17] to rewriting modulo equations, inspired by an application in process algebra [21].

In this paper, we show how another recent termination technique from Zantema and Geser [34], based on abstract commutation, can be extended to the setting of rewriting modulo equations. It can be considered as a general applicable technique for proving termination of rewriting modulo equations. This is of interest itself, independent from the field of process algebra. Basically, termination of a rewrite system R is proved by means of termination of a simplified rewrite system S and an auxiliary rewrite system U connecting R and S. Surprisingly, for extending this framework to the setting of rewriting modulo a set of equations E, no cooperation between R and E is required, only between U and E. The commutation technique from [34] is related to earlier techniques from [4, 8, 3]. In [4] rewriting modulo equations is considered, and in [3] the commutation technique is presented in the setting of rewriting modulo AC.

*Revised version of 'Prefix iteration in basic process algebra: applying termination techniques', which appeared in the proceedings of the ACP'95 workshop in Eindhoven, April 1995.

We present an extensive example of an application of abstract commutation modulo equations, which concerns the binary operator x^*y from Kleene [26], called Kleene star or iteration. The process term p^*q can choose to execute either p, after which it evolves into p^*q again, or q, after which it terminates. Milner [28] was the first to study the Kleene star in process algebra, modulo bisimulation equivalence from Park [29]. Recently, a paper by Bergstra, Bethke, and Ponse [10] has caused a resurgence in this line of research, mostly dealing with complete axiomatizations for iteration [20, 19] and with variants of iteration [18, 1, 2].

In process algebra, rewriting is usually applied in order to obtain normal forms for which the syntax and their semantics are closely related. In the case of iteration, such rewriting strategies have the tendency to produce self-embedding rewrite rules, where the right-hand side can be obtained from the left-hand side by the elimination of function symbols. For such a rewrite system, standard techniques to prove termination such as path orderings and weight functions in the natural numbers cannot be applied, see for example [32]. Hence, the only sensible strategy to prove termination of such a rewrite system is to transform it into a rewrite system without self-embedding rules, for which termination can be derived. Transformation techniques which can be applied for this purpose are based either on commutations [8, 34] or on semantic properties [24, 33].

In [20], a rewrite system for iteration in process algebra was applied, which contains self-embedding rules. It was proved to be terminating by the technique of semantic labelling from [33]. Here, we consider another rewrite system for iteration in process algebra, motivated by the aim to find normal forms for which the syntax and their semantics are closely related. Again this rewrite system contains self-embedding rules. We present an elegant proof that it is terminating, based on the technique of abstract commutation from [34] extended to rewriting modulo equations.

Acknowledgements. Alfons Geser and Rob van Glabbeek provided helpful comments.

2 Abstract Commutation Modulo Equations

We present a generalization of a termination technique from Zantema and Geser [34], based on abstract commutation, to the setting of rewriting modulo equations.

2.1 The basic theorem

We start with some general observations concerning relations.

Let R, S, T, E denote binary relations on a fixed set $\mathcal{V}$. We write a dot symbol for relational composition, i.e., one has $t(R.S)t'$ if and only if there exists a t'' such that tRt'' and $t''St'$. We write R^+ for the transitive closure of R and R^* for the reflexive transitive closure of R. Further we write $R \subseteq S$ if tRt' implies tSt'. Clearly, if $R \subseteq S$ then $R.T \subseteq S.T$ and $T.R \subseteq T.S$.

We write $\infty(t, R)$ if there exists an infinite sequence $tRt_1Rt_2Rt_3R\cdots$. A relation R is called *terminating* if there does not exist any term t satisfying $\infty(t, R)$.

In the following lemma we collect some standard properties for relations, which are easy to check.

Lemma 2.1

1. *If* $R.S \subseteq S^*.R$*, then* $R.S^* \subseteq S^*.R$*.*
2. *If* $R.S \subseteq S.R^*$*, then* $R^*.S \subseteq S.R^*$*.*
3. *If* $R.S \subseteq T^+.R$ *and* $t'Rt$ *and* $\infty(t,S)$*, then* $\infty(t',T)$*.*

For relations R, E we write R/E for $E^*.R.E^*$. The intuition here is that the reduction relation R is taken modulo equations E. However, we do not need that E is symmetric, hence our theorem is even on relative termination which is more general than modulo an equivalence. Now we state our main termination theorem.

Theorem 2.2 *Let* R, S, T, E *be binary relations satisfying*

1. S/E *is terminating,*
2. $R \subseteq (S/E)^+.T^*$*,*
3. $T.R \subseteq (R/E)^+.T^*$*,*
4. $T.E \subseteq E^*.T$*.*

Then R/E *is terminating.*

Binary relations on a fixed set describing some notion of reduction are often called abstract reduction systems. Conditions 3 and 4 describe commutation between the various relations, which is why the technique of using such a theorem is called abstract commutation.

First, we provide some intuition for Theorem 2.2. Suppose that R/E is not terminating, so that there is an infinite reduction $tE^*.R.E^*.R.E^* \cdots$. Using condition 2, the leftmost R-step in this reduction can be replaced by $(S/E)^+.T^*$. The created T-steps jump over the consecutive E-steps by applying condition 4, and they jump over the next R-step by applying condition 3. Then the infinite reduction starts with $(S/E)^+.R$, and the same procedure can be applied with respect to the leftmost R-step in this reduction. Repeating this construction forever yields an infinite (S/E)-reduction, which contradicts condition 1.

This vague reasoning is not yet a formal proof; the "$^+$" and "*" signs in the conditions are rather subtle, as we shall see in Example 2.3. We continue to present the exact proof of Theorem 2.2.

Proof. From condition 4 and the first item of Lemma 2.1 we conclude $T.E^* \subseteq E^*.T$. From this and condition 3 we conclude:

$$\begin{array}{rcl}(T/E).(R/E) & = & E^*.T.E^*.R.E^* \\ & \subseteq & E^*.E^*.T.R.E^* \\ & \subseteq & E^*.E^*.(R/E)^+.T^*.E^* \\ & \subseteq & (R/E).((R \cup T)/E)^*.\end{array}$$

Since also $(R/E).(R/E) \subseteq (R/E).((R\cup T)/E)^*$, we obtain

$$((R\cup T)/E).(R/E) \;=\; (R/E).(R/E)\cup (T/E).(R/E) \;\subseteq\; (R/E).((R\cup T)/E)^*.$$

From the second item of Lemma 2.1 and condition 2 we conclude

$$\begin{array}{rcl} ((R\cup T)/E)^*.(R/E) & \subseteq & (R/E).((R\cup T)/E)^* \\ & = & E^*.R.E^*.((R\cup T)/E)^* \\ & \subseteq & E^*.(S/E)^+.T^*.E^*.((R\cup T)/E)^* \\ & = & (S/E)^+.((R\cup T)/E)^*. \end{array}$$

Assume that R/E does not terminate. Then there exists an element t with $\infty(t, R/E)$. Clearly $t((R\cup T)/E)^*t$, hence the third item of Lemma 2.1 yields $\infty(t, S/E)$. This contradicts condition 1. □

To stress the subtlety of this theorem, we show that condition 4 may not be weakened to $T.E \subseteq E^*.T^+$.

Example 2.3 *Let* $\mathcal{V} = \{1, 2, 3, 4\}$ *and*

$$\begin{array}{lllll} 1R3, & & & & \\ 1S4, & & & & \\ 4T3 & 3T2 & 1T1 & 2T2 & 3T3, \\ 1E2 & 2E1 & 2E3 & 3E2. & \end{array}$$

Now S/E is terminating, because S consists only of $1S4$, and 4 cannot be reduced by S nor by E. The relation inclusions in conditions 2 and 3 in Theorem 2.2 are easily checked. Condition 4 does not hold, but the slightly weakened version $T.E \subseteq E^*.T^+$ holds and is easily checked. However, R/E is not terminating: $1R3E2E1R3\cdots$.

2.2 Application to rewrite systems

Before applying Theorem 2.2 to rewrite systems, first we recall some standard terminology from term rewriting. See e.g. Klop [27] for an overview of the field of term rewriting.

Definition 2.4

- *A rewrite rule $l \to r$ is called* left-linear *if each variable occurs at most once in l.*
- *A rewrite rule $l \to r$ is called* non-erasing *if each variable in l also occurs in r.*

A rewrite system is called left-linear or non-erasing respectively if so are all its rules.

Assume a rewrite system R and a set of equations E, and let $\leftrightarrow_E$ denote the rewrite relation obtained from taking the equations of E in both directions as rewrite rules. Hence the congruence $=_E$ is equal to the transitive reflexive closure $\leftrightarrow_E^*$ of $\leftrightarrow_E$.

Definition 2.5 *A rewrite system R is called terminating modulo a set E of equations if no infinite reduction exists of the shape*

$$t_1 \to_R t_2 =_E t_3 \to_R t_4 =_E t_5 \to_R t_6 =_E \cdots.$$

Termination of R modulo E is indeed equivalent to termination of the binary relation R/E as defined in Section 2.1, as was already suggested by the notation.

We want to apply Theorem 2.2 by choosing S to be an adaptation of R for which termination modulo E is easy to prove, and by choosing T to be the inverse of the rewrite relation $\rightarrow_U$ for some auxiliary rewrite system U. The following theorem is an immediate consequence of Theorem 2.2.

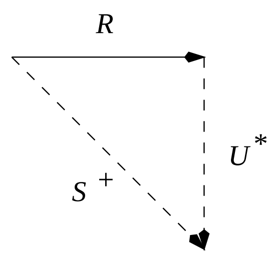
Corollary 2.6 *Let R, S and U be rewrite systems and E a set of equations satisfying*

1. *S/E is terminating,*
2. *for each rule $l \rightarrow r$ in R there exists a t such that $l \rightarrow_S^+ t$ and $r \rightarrow_U^* t$,*
3. *if $t \rightarrow_U t'$ and $t \rightarrow_R t''$, then there exists a u such that $t' \rightarrow_R^+ u$ and $t'' \rightarrow_U^* u$,*
4. *if $t \rightarrow_U t'$ and $t \leftrightarrow_E t''$, then there exists a u such that $t' =_E u$ and $t'' \rightarrow_U u$.*

Then R/E is terminating.

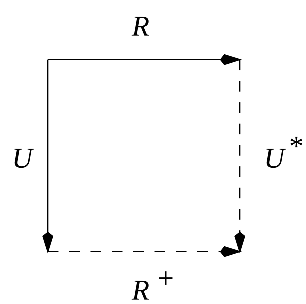
If E is empty, then condition 4 is trivially fulfilled and the theorem coincides with Theorem 12 on abstract commutation from [34].

Condition 2 of Corollary 2.6 can be represented graphically as follows:



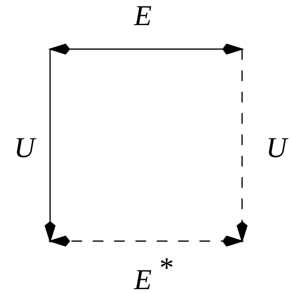
In the typical case S is a modification of R for which termination modulo E is easier to prove than for R, and U is chosen to contain rules justifying condition 2.

Condition 3 can be represented graphically as follows:



If U is left-linear and non-erasing, and if R is left-linear, then this requirement is always fulfilled for non-overlapping redexes, so that it can be verified by a finite analysis of overlapping redexes. In the typical case, in the first attempt for U, being the set of rules required by condition 2, condition 3 does not hold. Then new rules have to be added to U a number of times to obtain condition 3. This is a kind of *completion*, similar to what is done in Bellegarde and Lescanne [8]. In the application of Corollary 2.6 in this paper, the rewrite systems R and S will also be extended during this completion, and the final auxiliary rewrite system U will have infinitely many rules.

Finally, condition 4 of Corollary 2.6 can be represented graphically as follows:



If U is left-linear, and if the equations in E taken as rewrite rules are left-linear and non-erasing in both directions, then this condition can be verified by a finite analysis of overlapping redexes, similar as for condition 3. In particular, both associativity and commutativity satisfy these requirements for E.

3 Application to Process Algebra with Iteration

We consider a rewrite system in process algebra with the iteration operator, which we prove terminating by means of abstract commutation modulo equations.

3.1 Preliminaries

Our application is situated in Basic Process Algebra, which assumes a non-empty alphabet A of atomic actions, and two binary operators $x + y$ and $x \cdot y$, which represent alternative and sequential composition, respectively. Intuitively, an atom a executes action a after which it terminates, a process $p + q$ executes either p or q, and a process $p \cdot q$ executes first p and then q. Furthermore, we assume a special atomic action δ, called deadlock, which blocks all behaviour.

We extend this process algebra with a binary operator x^*y, called Kleene star or iteration, from Kleene [26]. Intuitively, the expression p^*q yields a solution for the recursive equation $X = p \cdot X + q$, that is, p^*q can choose to execute either p, after which it evolves into p^*q again, or q, after which it terminates. Summarizing, process terms are defined inductively as follows, where $a \in A$:

$$p \ ::= \ a \mid \delta \mid p+p \mid p \cdot p \mid p^*p.$$

In the sequel, the $\cdot$ binds stronger than the $+$, so $p \cdot q + r$ represents $(p \cdot q) + r$. Often, $p \cdot q$ will be abbreviated to pq.

In Basic Process Algebra, process terms are considered modulo *bisimulation equivalence* from Park [29]. Intuitively, two process terms p and q are bisimilar (i.e., bisimulation equivalent) if they have the same branching structure, that is, if p can execute action a and evolve into p' (or terminate), then q can execute action a and evolve into a q' which is bisimilar to p' (or terminate).

In the sequel, process terms are considered modulo AC of the $+$, and $p =_{\mathrm{AC}} q$ denotes that p and q are equal modulo AC of the $+$. Note that $p_0 + p_1$ and $p_1 + p_0$ are bisimilar indeed.

3.2 A rewrite system for iteration

In process algebra, rewriting is usually applied in order to obtain normal forms for which the syntax and their semantics are closely related. For example, in a term $(\ldots(p \cdot q_1) \cdot q_2) \ldots)q_n$, the initial behaviour of the term is determined by the subterm p, but on the syntactic level this subterm is 'hidden' at depth n. This inconsistency can be resolved by applying the rewrite rule

$$(xy)z \;\rightarrow\; x(yz)$$

sufficiently many times. Likewise, the Kleene star causes that initial behaviour may be hidden in the syntax; in $p_0^*(p_1^*(\ldots(p_n^* q)\ldots)$, part of the initial behaviour of the term is determined by the subterm q, but on the syntactic level this subterm is hidden at depth n. This inconsistency can be resolved by applying the rewrite rules

$$\begin{array}{rcl} z^*(x^*y) & \rightarrow & z^*(x(x^*y) + y) \\ x^*y + z & \rightarrow & x(x^*y) + y + z \end{array}$$

sufficiently many times. For example, this type of reduction is applied in the completeness proof for *prefix* iteration a^*x with abstraction in [1].

1.	$(x+y)z$	$\rightarrow$	$xz + yz$
2.	$(xy)z$	$\rightarrow$	$x(yz)$
3.	$x + \delta$	$\rightarrow$	x
4.	δx	$\rightarrow$	δ
5.	$(x^*y)z$	$\rightarrow$	$x^*(yz)$
6.	δ^*x	$\rightarrow$	x
7.	$x^*y + z$	$\rightarrow$	$x(x^*y) + y + z$
8.	$(x^*y)^*z$	$\rightarrow$	$(x(x^*y) + y)^*z$
9.	$z^*(x^*y)$	$\rightarrow$	$z^*(x(x^*y) + y)$

Table 1: The rewrite system R_0

Table 1 contains a rewrite system R_0, which aims solely at reducing terms to a syntactic form which is closely related to their semantic behaviour. Rules 1,2,4,5 reduce sequential composition to its prefix counterpart, rules 3,6 remove redundant deadlocks, and rules 7-9 expand iteration in the context with alternative composition and with iteration. The rewrite rules are to be interpreted modulo AC of the $+$. It is not hard to check that the rules in R_0 are 'sound' in the sense that if R_0 reduces p to q, then p and q are bisimilar, see e.g. [10].

Note that the rules 7-9 are self-embedding: their left-hand sides can be embedded in the corresponding right-hand sides. Hence, it is not possible to prove termination of R_0 neither by means of a recursive path ordering nor by a compositional weight function in the naturals.

The following sections are devoted to proving the following theorem, which states that R_0 is terminating modulo AC of the $+$. The proof is based on the technique of abstract commutation modulo equations, which we developed previously.

Theorem 3.1 *The rewrite system R_0 is terminating modulo AC of the $+$.*

3.3 Definition of rewrite systems R, S, U, and E

The intuition behind the termination proof of R_0 is that the expansion from a pattern x^*y to $x(x^*y) + y$, as is done by rules 7-9, can occur at most only once for every occurrence of an iteration symbol. We formalize this as follows. Extend the signature with the binary function symbol $x^{\#}y$. Intuitively, this new function symbol will be used to register that the expansion from x^*y to $x(x^*y)+y$ has been done. In a first attempt to apply Corollary 2.6, we choose R, S, U, and E as follows. The choices for R, S, and U will be adapted later on.

- R equals R_0.
- S, presented in Table 4, consists of rules 1-6 from R_0, together with one extra rule

  $$14. \quad x^*y \;\rightarrow\; x(x^{\#}y) + y,$$

  which enables to mimic the rules 7-9 in R_0, with the modification that the expressions $x(x^*y)$ in the right-hand sides of these rules are replaced by $x(x^{\#}y)$.
- In order to satisfy condition 2 of Corollary 2.6, for each rule $l \rightarrow r$ of R there should exist a t such that $l \rightarrow_S^+ t$ and $r \rightarrow_U^* t$. Since rules 1-9 in R are also in S, these rules do not cause any problem. In order to deal with the rules 10-12, it is sufficient to have the following rule in U:

  $$r_0 \;\; x(y^*z) \;\;\rightarrow\;\; x(y^{\#}z).$$

  So as a first obvious try, we choose U to consist of r_0 only.
- Finally, since rewrite rules are applied modulo AC of the $+$, we take E to consist of the two AC equations, which are presented in Table 2.

$$\begin{array}{rcl} x+y & = & y+x \\ (x+y)+z & = & x+(y+z) \end{array}$$

Table 2: The equational system E

It is easy to check conditions 1,2,4 of Corollary 2.6, that S/E is terminating, and that $\rightarrow_R \;\subseteq\; \rightarrow_S^+ \cdot {}_U^*\!\leftarrow$, and that ${}_U\!\leftarrow \cdot \leftrightarrow_E \;\subseteq\; =_E \cdot {}_U\!\leftarrow$. (We will provide rigorous arguments soon.) However, condition 3 of Corollary 2.6, ${}_U\!\leftarrow \cdot \rightarrow_R \;\subseteq\; \rightarrow_R^+ \cdot {}_U^*\!\leftarrow$, does not yet hold. Therefore we extend the systems R, S, and U with some new rules, triggered by the desired validity of condition 3. This process of completion ends in the following choices for the systems R, S, and U.

- In order to satisfy condition 3, the rewrite system R extends the rewrite system R_0 with four new rules, which are labelled 10-13. R is presented in Table 3.

1.	$(x+y)z$	$\rightarrow$	$xz+yz$
2.	$(xy)z$	$\rightarrow$	$x(yz)$
3.	$x+\delta$	$\rightarrow$	x
4.	δx	$\rightarrow$	δ
5.	$(x^*y)z$	$\rightarrow$	$x^*(yz)$
6.	$\delta^* x$	$\rightarrow$	x
7.	x^*y+z	$\rightarrow$	$x(x^*y)+y+z$
8.	$(x^*y)^*z$	$\rightarrow$	$(x(x^*y)+y)^*z$
9.	$z^*(x^*y)$	$\rightarrow$	$z^*(x(x^*y)+y)$
10.	$(x^\# y)z$	$\rightarrow$	$x^\#(yz)$
11.	$\delta^\# x$	$\rightarrow$	x
12.	$(x^*y)^\# z$	$\rightarrow$	$(x(x^*y)+y)^\# z$
13.	$x^\#(y^*z)$	$\rightarrow$	$x^\#(y(y^*z)+z)$

Table 3: The rewrite system R

- Since we have extended R, and since we want that condition 2 remains valid, the rewrite system S is extended with the rules 10 and 11. S is presented in Table 4. We will see that S/E is terminating, by defining an appropriate weight function in the natural numbers.

1.	$(x+y)z$	$\rightarrow$	$xz+yz$
2.	$(xy)z$	$\rightarrow$	$x(yz)$
3.	$x+\delta$	$\rightarrow$	x
4.	δx	$\rightarrow$	δ
5.	$(x^*y)z$	$\rightarrow$	$x^*(yz)$
6.	$\delta^* x$	$\rightarrow$	x
10.	$(x^\# y)z$	$\rightarrow$	$x^\#(yz)$
11.	$\delta^\# x$	$\rightarrow$	x
14.	x^*y	$\rightarrow$	$x(x^\# y)+y$

Table 4: The rewrite system S

- Finally, in order to satisfy condition 3, we define the rewrite system U to be the infinite collection of rewrite rules in Table 5.

r_0	$x(y^*z)$	$\to$	$x(y^{\#}z)$
r_1	$x((y^*z)w_0)$	$\to$	$x((y^{\#}z)w_0)$
r_2	$x(((y^*z)w_0)w_1)$	$\to$	$x(((y^{\#}z)w_0)w_1)$
		$\vdots$	

Table 5: The rewrite system U

More precisely, U consists of rewrite rules r_i of the form $x \cdot C_i[y^*z] \to x \cdot C_i[y^{\#}z]$ for $i \geq 0$, where the contexts $C_i[]$ are defined inductively by

$$C_0[] = [], \qquad C_{i+1}[] = C_i[] \cdot w_i,$$

with w_i a fresh variable. Equivalently, one can say that r_i is of the form $x \cdot D_i[y^*z] \to x \cdot D_i[y^{\#}z]$, where the contexts $D_i[]$ are defined inductively by

$$D_0[] = [], \qquad D_{i+1}[] = D_i[[] \cdot v_i].$$

with v_i a fresh variable. We will need both representations of r_i later on.

3.4 Termination of rewrite system R

We will now prove that the rewrite system R is terminating modulo AC of the $+$, by verifying the four conditions of Corollary 2.6. Since R incorporates R_0, this result implies Theorem 3.1, which says that R_0 is terminating modulo AC of the $+$.

1. S/E is terminating.

   Define the following weight function on terms.

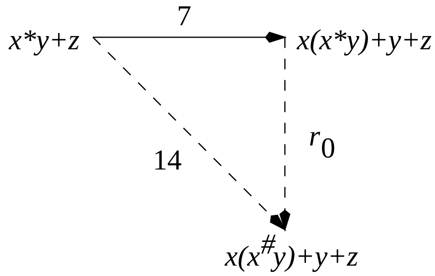
$$\begin{aligned} w(a) &= 2 \\ w(\delta) &= 2 \\ w(x) &= 2 \\ w(p+q) &= w(p) + w(q) \\ w(pq) &= w(p)^2 \cdot w(q) \\ w(p^{\#}q) &= w(p) + w(q) \\ w(p^*q) &= w(p)^2 \cdot (w(p) + w(q)) + w(q) + 1 \end{aligned}$$

   Note that terms which are equal modulo AC of the $+$ have the same weight. It is easy to see that the weight of terms strictly decreases under application of rules in S. Hence, S/E is terminating.

2. For each rule $l \to r$ of R we have some term t for which $l \to_S^+ t$ and $r \to_U^* t$.

   Rules 1-6 and 10,11 in R are also present in S, so for those rules we can choose t equal to r. Rule 14 in S and rule r_0 in U make that we can deal with rules

7-9,12,13 in R. For example, in the case of rule 7:



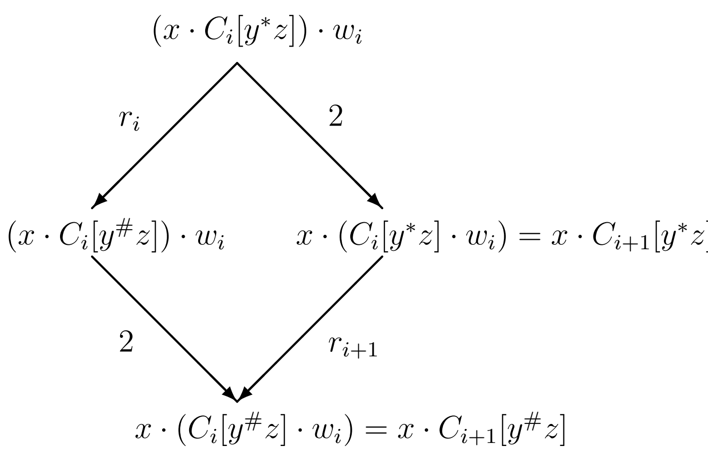
3. If $t \to_U t'$ and $t \to_R t''$, then there exists a u for which $t' \to_R^+ u$ and $t'' \to_U^* u$.

   Note that U is left-linear and non-erasing, and that R is left-linear, so we can limit ourselves to a finite analysis of overlapping redexes. It is not hard to see that there are ten types of overlaps between a left-hand side of U and a left-hand side of R, which involve rules 1, 2 (twice), 4, 5 (twice), 6, 8, 9, 10 in R, respectively. We treat these ten cases separately.

   (a) $(x+y) \cdot C_i[z^*w]$ can be reduced by rule r_i in U to $(x+y) \cdot C_i[z^\# w]$, and by rule 1 in R to $x \cdot C_i[z^*w] + y \cdot C_i[z^*w]$. For this overlap condition 3 is satisfied, because applying rule 1 to $(x+y) \cdot C_i[z^\# w]$ and applying rule r_i twice to $x \cdot C_i[z^*w] + y \cdot C_i[z^*w]$ both yield $x \cdot C_i[z^\# w] + y \cdot C_i[z^\# w]$.

   (b) $(xy) \cdot C_i[z^*w]$ can be reduced by rule r_i in U to $(xy) \cdot C_i[z^\# w]$, and by rule 2 in R to $x(y \cdot C_i[z^*w])$. For this overlap condition 3 is satisfied, because applying rule 2 to $(xy) \cdot C_i[z^\# w]$ and applying rule r_i to $x(y \cdot C_i[z^*w])$ both yield $x(y \cdot C_i[z^\# w])$.

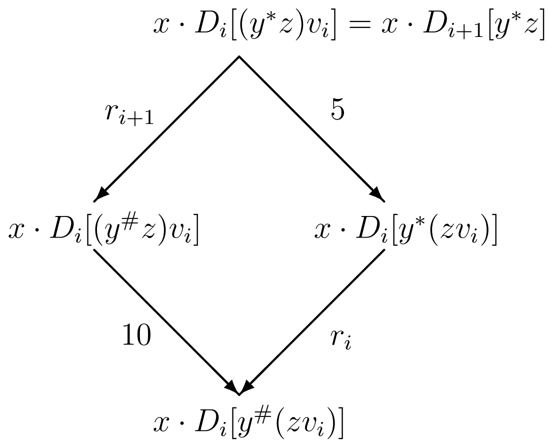
   (c) $(x \cdot C_i[y^*z]) \cdot w_i$ can be reduced by rule r_i in U and by rule 2 in R. For this overlap condition 3 is satisfied, owing to rule r_{i+1} in U.



   (d) $\delta \cdot C_i[x^*y]$ can be reduced by rule r_i in U to $\delta \cdot C_i[x^\# y]$, and by rule 4 in R to δ. For this overlap condition 3 is satisfied, because rule 4 reduces $\delta \cdot C_i[x^\# y]$ to δ.

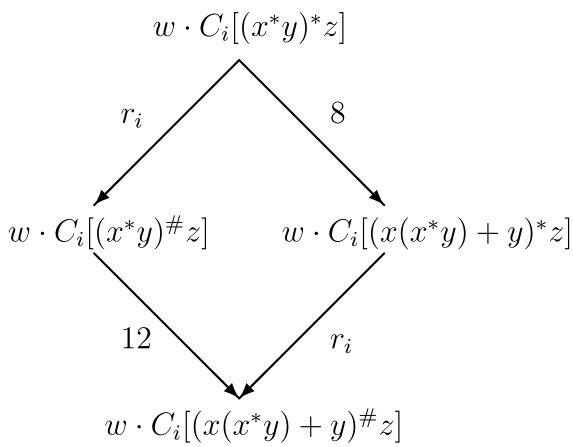
   (e) $(x^*y) \cdot C_i[z^*w]$ can be reduced by rule r_i in U to $(x^*y) \cdot C_i[z^\# w]$, and by rule 5 in R to $x^*(y \cdot C_i[z^*w])$. For this overlap condition 3 is satisfied, because applying rule 5 to $(x^*y) \cdot C_i[z^{\#}w]$ and applying rule r_i to $x^*(y \cdot C_i[z^*w])$ both yield $x^*(y \cdot C_i[z^{\#}w])$.

(f) $x \cdot D_i[(y^*z)v_i]$ can be reduced by rule r_{i+1} in U and by rule 5 in R. For this overlap condition 3 is satisfied, owing to rule r_i in U and rule 10 in R.



(g) $x \cdot C_i[\delta^*y]$ can be reduced by rule r_i in U to $x \cdot C_i[\delta^{\#}y]$, and by rule 6 in R to $x \cdot C_i[\delta]$. For this overlap condition 3 is satisfied, owing to rule 11 in R, which reduces $x \cdot C_i[\delta^{\#}y]$ to $x \cdot C_i[\delta]$.

(h) $w \cdot C_i[(x^*y)^*z]$ can be reduced by rule r_i in U and by rule 8 in R. For this overlap condition 3 is satisfied, owing to rule 12 in R.



(i) $w \cdot C_i[z^*(x^*y)]$ can be reduced by rule r_i in U to $w \cdot C_i[z^{\#}(x^*y)]$, and by rule 9 in R to $w \cdot C_i[z^*(x(x^*y) + y)]$). For this overlap condition 3 is satisfied, because applying rule 13 in R to $w \cdot C_i[z^{\#}(x^*y)]$ and applying rule r_i to $w \cdot C_i[z^*(x(x^*y) + y)]$) both yield $w \cdot C_i[z^{\#}(x(x^*y) + y)]$).

(j) $(x^{\#}y) \cdot C_i[z^*w]$ can be reduced by rule r_i in U to $(x^{\#}y) \cdot C_i[z^{\#}w]$, and by rule 10 in R to $x^{\#}(y \cdot C_i[z^*w])$. For this overlap condition 3 is satisfied, because applying rule 10 to $(x^{\#}y) \cdot C_i[z^{\#}w]$ and applying rule r_i to $x^{\#}(y \cdot C_i[z^*w])$ both yield $x^{\#}(y \cdot C_i[z^{\#}w])$.

4. If $t \rightarrow_U t'$ and $t \leftrightarrow_E t''$, then there exists a u for which $t' =_E u$ and $t'' \rightarrow_U u$.

   Since U is left-linear, and the equations in E taken as rewrite rules are left-linear and non-erasing in both directions, again we can limit ourselves to a finite analysis of overlapping redexes. Since all left-hand and right-hand sides of E contain no other symbols than $+$, and since the left-hand sides of U contain no $+$ symbols, no overlapping redexes are possible.

So according to Corollary 2.6, we may conclude that R/E is terminating. Since the rewrite system R_0 is contained in R, we may conclude that Theorem 3.1 holds, that is, R_0 is terminating modulo AC of the $+$. □

3.5 Variants of iteration

Recently, several variants of iteration have been introduced. We discuss briefly how the rewrite system R_0 for iteration in Table 1 can be adapted for these variants.

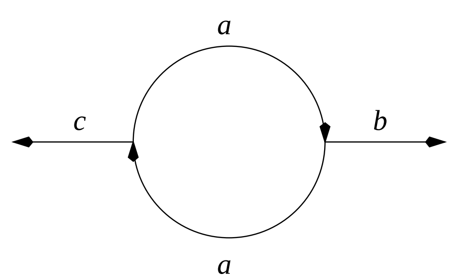
Bergstra, Bethke, and Ponse introduced in [9] a generalized iteration construct $(x_1, x_2)^*(y_1, y_2)$, called double-exit iteration, with the defining equation:

$$(x_1, x_2)^*(y_1, y_2) \;=\; x_1((x_2, x_1)^*(y_2, y_1)) + y_1.$$

The reason for this generalization is the desire to capture regular (i.e., finite-state) processes which cannot be described by iteration. An example of a process that can be described by double-exit iteration, but not by the Kleene star, is given by the following recursive specification:

$$\begin{cases} X &= a \cdot Y + b \\ Y &= a \cdot X + c \end{cases}$$

which can be represented graphically as follows:



Similarly as we did for iteration, in the form of the rewrite system R_0 in Table 1, it is possible to define a rewrite system for double-exit iteration which reduces process terms to a form which is more adapted to their semantics. For example, rule 7 of R_0 formulated for double-exit iteration takes the form

$$(x_1, x_2)^*(y_1, y_2) + z \;\rightarrow\; x_1((x_2, x_1)^*(y_2, y_1)) + y_1 + z.$$

Again, abstract commutation modulo equations suffices to prove termination for this rewrite system for double-exit iteration, following the lines of the termination proof for R_0.

In [18], prefix iteration a^*y was introduced, where the left-hand side of iteration is restricted to atomic actions. There are two motivations to do so. Firstly, the process algebra CCS restricts sequential composition xy to its prefix counterpart ay, which is troublesome if one wants to axiomatize the Kleene star. This complication resolves if iteration is restricted to prefix iteration. Secondly, Sewell [30] showed that the Kleene star in the presence of deadlock does not allow a complete finite equational axiomatization, while in [18] it was shown that such an axiomatization does exist for prefix iteration with deadlock.

In order to obtain a rewrite system for prefix iteration which reduce process terms to a form which is more adapted to their semantics, it is sufficient to replace rules 5,7,8,9 by their respective instantiations for prefix iteration. For example, the adaptation of rule 7 for prefix iteration is

$$a^*y + z \;\rightarrow\; a(a^*y) + y + z.$$

Termination of the resulting rewrite system for prefix iteration follows immediately from termination of R_0.

In [21], termination of this rewrite system for prefix iteration extended with two rules for the empty process ϵ, was used to deduce a completeness result for prefix iteration. One of the extra rules for the empty process reads

$$\epsilon x \rightarrow x.$$

We note that adding this rule to the rewrite system R_0 in Table 1 for iteration would yield a rewrite system that is *not* terminating:

$$\begin{array}{rcl} \epsilon(\epsilon^*x) + x & \rightarrow & \epsilon^*x + x \\ & \rightarrow & \epsilon(\epsilon^*x) + x + x \\ & \rightarrow & \cdots. \end{array}$$

Recently, a simpler completeness proof for prefix iteration was discovered [1].

4 Concluding Remarks

Instead of studying arbitrary process terms, it is often more convenient to look only at process terms which have a 'nice' syntactic form, which reflects their semantics. Term rewriting can be a useful tool to reduce process terms to such a nice form: rewrite rules are applied to process terms giving equivalent terms, until no rewrite rule is applicable any more. The result is called a normal form, and the goal is to design the rewrite system in such a way that the normal forms have the desired syntactic shape. In order to be able to extend a semantics for normal forms to all process terms it is necessary that every process term has a unique normal form. To achieve this, the rewrite system has to be (ground) confluent and (weakly) normalizing.

In this paper, we focused on the iteration construct p^*q. This process term either executes p and evolves into p^*q, or q and terminates, so in order to obtain terms of a syntactic form which relates closely to their semantics, the form $p(p^*q) + q$ is preferred over p^*q. However, if we allow any unfolding from x^*y to $x(x^*y) + y$

as a rewrite step, we will not obtain normalization. Therefore, we allowed such unfoldings only to iteration operators which occur as an argument in a "+" or a "*". From a semantic point of view this is already satisfactory.

The main problem in this case is normalization: such an unfolding is in conflict with the basic intuition of normalization as making terms smaller. The idea of our termination proof is that every "*" symbol which occurs as the root symbol of an argument at the right-hand side of a "·", is changed into a fresh symbol "#". In this way, rewrite rules of the form

$$C[x^*y] \ \ \rightarrow \ \ C[x(x^*y)+y]$$

for certain non-empty contexts $C[]$, are replaced by the single rule

$$x^*y \ \ \rightarrow \ \ x(x^\# y)+y.$$

The old rules conflict with the basic idea of normalization as decreasing terms, since the left-hand side can be embedded in the right-hand side. In the new rule we do not have this problem, since we are free to interpret "*" as being big and "#" as being small.

To justify normalization of this transformation, we used the technique of abstract commutation, which can be described in an abstract setting. A basic ingredient is commutation with an auxiliary system, which describes the difference between the old and the new system, in our case an infinite extension of the rule $x(y^*z) \rightarrow x(y^\# z)$, which was obtained by some kind of completion. We extended the technique of abstract commutation to rewriting modulo equations, because in the setting of process algebra the symbol "+" is taken modulo AC.

References

[1] L. Aceto, W.J. Fokkink, R.J. van Glabbeek, and A. Ingólfsdóttir. Axiomatizing prefix iteration with silent steps. *Information and Computation*, 127(1):26–40, 1996.

[2] L. Aceto and W.J. Fokkink. An equational axiomatization for multi-exit iteration. Report RS-96-22, BRICS, University of Aalborg, 1996.

[3] L. Bachmair. Associative-commutative reduction orderings. *Information Processing Letters*, 43(1):21–27, 1992.

[4] L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In J.H. Siekmann editor, *Proceedings 8th Conference on Automated Deduction (CADE-8),* Oxford, LNCS 230, pages 5–20. Springer-Verlag, 1986.

[5] L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proceedings 1st Symposium on Logic in Computer Science (LICS'86),* Cambridge, Massachusetts, pages 346–357. IEEE Computer Society Press, 1986.

[6] L. Bachmair and D.A. Plaisted. Termination orderings for associative-commutative rewrite systems. *Journal of Symbolic Computation*, 1(4):329–349, 1985.

[7] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, 9(2):127–167, 1986.

[8] F. Bellegarde and P. Lescanne. Termination by completion. *Applicable Algebra in Engineering, Communication and Computation*, 1:79–96, 1990.

[9] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration. Report CS-R9314, University of Amsterdam, 1993. Available via ftp at `fwi.uva.nl` as `pub/programming-research/reports/1993/P9314.ps.Z`.

[10] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[11] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.

[12] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[13] N. Dershowitz, J. Hsiang, N.A. Josephson, and D.A. Plaisted. Associative-commutative rewriting. In *Proceedings 8th Conference on Artificial Intelligence,* Karlsruhe, pages 940–944, 1983.

[14] N. Dershowitz and S. Mitra. Path orderings for termination of AC rewriting. In M. Rusinowitch and J.L. Rémy, editors, *Proceedings 3rd Workshop on Conditional Term Rewriting Systems (CTRS'92),* Pont-à-Mousson, LNCS 656, pages 168–174. Springer-Verlag, 1992.

[15] M.C.F. Ferreira. *Termination of Term Rewriting – well-foundedness, totality and transformations*. PhD thesis, Utrecht University, 1995.

[16] M.C.F. Ferreira. Dummy elimination in equational rewriting. In H. Ganzinger, editor, *Proceedings 7th Conference on Rewriting Techniques and Applications (RTA'96),* New Jersey, LNCS 1103, pages 63–77. Springer-Verlag, 1996.

[17] M.C.F. Ferreira and H. Zantema. Dummy elimination: making termination easier. In H. Reichel, editor, *Proceedings 10th Conference on Fundamentals of Computation Theory (FCT'95),* Dresden, LNCS 965, pages 243–252. Springer-Verlag, 1995.

[18] W.J. Fokkink. A complete equational axiomatization for prefix iteration. *Information Processing Letters*, 52(6):333–337, 1994.

[19] W.J. Fokkink. On the completeness of the equations for the Kleene star in bisimulation. In M. Wirsing and M. Nivat, editors, *Proceedings 5th Conference on Algebraic Methodology and Software Technology (AMAST'96)*, Munich, LNCS 1101, pages 180–194. Springer-Verlag, 1996.

[20] W.J. Fokkink and H. Zantema. Basic process algebra with iteration: completeness of its equational axioms. *The Computer Journal*, 37(4):259–267, 1994.

[21] W.J. Fokkink and H. Zantema. Prefix iteration in basic process algebra: applying termination techniques. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Proceedings 2nd Workshop on the Algebra of Communicating Processes (ACP'95),* Eindhoven, Report CS-95-14, pages 139–156. Eindhoven University of Technology, 1995.

[22] I. Gnaedig and P. Lescanne. Proving termination of associative-commutative rewriting systems by rewriting. In J.H. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction (CADE-8),* Oxford, LNCS 230, pages 52–61. Springer-Verlag, 1986.

[23] J.P. Jouannaud and M. Muñoz. Termination of a set of rules modulo a set of equations. In R.E. Shostak, editor, *Proceedings 7th Conference on Automated Deduction (CADE-7),* Napa Valley, California, LNCS 170, pages 175–193. Springer-Verlag, 1984.

[24] S. Kamin and J.J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.

[25] D. Kapur, G. Sivakumar, and H. Zhang. A new method for proving termination of AC-rewrite systems. In K.V. Nori and C.E. Veni Madharan, editors, *Proceedings 10th Conference on Foundations of Software Technology and Theoretical Computer Science,* Bangalore, LNCS 472, pages 133–148. Springer-Verlag, 1990.

[26] S.C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[27] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume I, Background: Computational Structures*, pages 1–116. Oxford University Press, 1992.

[28] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[29] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference,* Karlsruhe, LNCS 104, pages 167–183. Springer-Verlag, 1981.

[30] P.M. Sewell. Bisimulation is not finitely (first order) equationally axiomatisable. In *Proceedings 9th Symposium on Logic in Computer Science (LICS'94),* Paris, pages 62–70. IEEE Computer Society Press, 1994.

[31] J. Steinbach. Improving associative path orderings. In M.E. Stickel, editor, *Proceedings 10th Conference on Automated Deduction (CADE-10),* Kaiserslautern, LNCS 449, pages 411–425. Springer-Verlag, 1990.

[32] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, 1994.

[33] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1,2):89–105, 1995.

[34] H. Zantema and A. Geser. A complete characterization of termination of $0^p1^q \to 1^r0^s$. In J. Hsiang, editor, *Proceedings 6th Conference on Rewriting Techniques and Applications (RTA'95),* Kaiserslautern, LNCS 914, pages 41–55. Springer-Verlag, 1995.