Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Simulating TRSs by minimal TRSs: a simple, efficient, and correct compilation technique

J.F.Th. Kamperman and H.R. Walters

Computer Science/Department of Software Technology

CS-R9605 1996

Report CS-R9605
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Simulating TRSs by Minimal TRSs a Simple, Efficient, and Correct Compilation Technique

J.F.Th. Kamperman and H.R. Walters ({pum,jasper}@cwi.nl)

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

A simple, efficient, and correct compilation technique for left-linear Term Rewriting Systems (TRSs) is presented. TRSs are compiled into *Minimal Term Rewriting Systems* (MTRSs), a subclass of TRSs, presented in [WK95]. In MTRSs, the rules have such a simple form that they can be seen as instructions for an easily implementable abstract machine, the *Abstract Rewriting Machine* (ARM). In the correctness proof, it is shown that the MTRS resulting from compilation of a TRS simulates neither too much (*soundness*) nor too little (*completeness*), nor does it introduce unwarranted infinite sequences (*termination conservation*). The compiler and its correctness proof are largely independent of the reduction strategy.

CR Subject Classification (1991): D.3.4 [Programming languages]: Processors – Compilers; D.1.1 [Programming Techniques]: Applicative (Functional) Programming; D.1.6: Logic Programming.

AMS Subject Classification (1991): 68N20: Compilers and generators; 68Q05: Models of Computation; 68Q42: Rewriting Systems; 68Q65: Abstract data types; algebraic specification.

Keywords & Phrases: minimal term rewriting systems, program transformation.

Note: Partial support received from the Foundation for Computer Science Research in the Netherlands (SION) under project 612-17-418, "Generic Tools for Program Analysis and Optimization".

1. Introduction

Term (graph) rewriting systems (TRSs) are becoming increasingly important for the implementation of theorem provers, verification tools, algebraic specifications, compiler generators, program analyzers and functional programming languages. Hence, a clear need arises for techniques enabling fast execution of TRSs.

A standard technique for speeding up the execution of a program in a formal (programming) language is *compilation* into the language of a concrete machine (e.g., a microprocessor). In compiler construction (c.f. [ASU86]), it is customary to use an *abstract machine* as abstraction of the concrete machine. On the one hand, this allows hiding details of the concrete machine in a small part of the compiler, and thus an easy reimplementation on other concrete machines. On the other hand, a good design of the abstract machine enables a simple mapping from source language into abstract machine language.

A compiler consists of zero or more transformations in the semantic domain of its source language, followed by a mapping to a lower-level language. This is repeated until the level

of the concrete machine is reached. Because they take place in one domain, the source-to-source transformations are easier to grasp semantically than the mappings to lower levels. In this paper, we present a compilation technique for TRSs which stays entirely within the well-known source language domain.

In [WK95], we have presented *Minimal Term Rewriting Systems* (MTRSs), a syntactic restriction of TRSs, and shown that by a modest change of perspective, an MTRS can be seen as a program for the Abstract Rewriting Machine (ARM), which is in turn easily implemented on a concrete machine. In [WK95], we concentrate on the concretization of MTRSs into abstract machine programs, we only show the plausibility of simulating arbitrary pattern-matching by discussing an example, and we assume innermost rewriting with syntactic specificity ordering throughout. In this paper, we concentrate on the mapping from TRSs into simulating MTRSs, and the correctness proofs of these mappings, and we have formulated our transformations in such a way as to minimize the assumptions regarding strategy and rule ordering.

The idea to express patternmatching of TRSs in the language of TRSs itself was inspired by [Pet92], where patternmatching of ML is expressed in ML itself. This paper does not contain a correctness proof, and the algorithm is formulated in a less formal way than our algorithm. The resulting pattern match code appears to have the same complexity as the code produced by our algorithm.

The idea to include a corectness proof is taken from [HG94], in which steps towards a provably correct compiler for OBJ3 are taken. Their compiler is less geared towards efficiency than ours.

In the remainder of this paper, we proceed as follows. First, in Sections 2, 3 and 4, we review TRSs, simulation and MTRSs, respectively. In Section 5 we discuss an example of the application of our technique.

Then, in Section 6, we present a transformation that yields a simulating MTRS from a given TRS, provided the latter is left-linear, and *simply complete* (in a simply complete TRS, every defined function f has a most general rule, i.e., a rule with an LHS consisting of f applied to a sufficient number of distinct variables). In Section 6.3, we drop the latter requirement of simple completeness by a second transformation.

The first transformation has the remarkable property that the simulation holds for the *unrestricted* rewrite relation, i.e. no assumptions regarding the rewrite strategy are made.

The second transformation is shown to be correct when we assume innermost rewriting with priorities between rules (similar to the priorities defined in [BBKW89]). This is not as bad as it seems, because given an implementation of innermost rewriting, other strategies can be simulated by further transformations (for an example of this, see [KW95]).

We conclude our paper with a discussion of related work, conclusions and directions for future work.

2. Term Rewriting

In this section, we mainly follow [Klo92], except for the notation of paths and contexts, which is taken from [DJ90].

A *signature* Σ consists of:

- A countably infinite set $\mathcal{V}$ of *variables*: $x, y, \ldots$
- A non-empty set $\mathcal{F}$ of *function symbols*: $f, g, \ldots$, each with an *arity* (≥ 0), which is the number of arguments the function requires. We denote the arity of f by $|f|$.

The set $T(\Sigma)$ of terms over Σ is the smallest set satisfying

- $\mathcal{V} \subset T(\Sigma)$,
- for all $f \in \mathcal{F}$ with arity n, and $t_1, \ldots t_n \in T(\Sigma)$, we have $f(t_1, \ldots, t_n) \in T(\Sigma)$.

We will write $var(t)$ for the set of variables occuring in t. Occasionally, we will abbreviate a sequence $t_1, \ldots, t_n$ to $\vec{t}$, and write $\mid \vec{t} \mid$ for n. We generalize this to empty sequences, which have $\mid \vec{t} \mid = 0$.

A path in a term is represented as a sequence of positive integers. By $t|_p$, we denote the *sub-term* of t at path p. For example, if $t = f(g, h(f(y, z)))$, then $t|_{2.1}$ is the first sub-term of t's second sub-term, which is $f(y, z)$. We write $p \in s$ if p is a valid path in s (i.e., indicates a sub-term of s), and $p_1 \leq p_2$ if p_1 is a prefix of p_2 (i.e., $\exists p_3 : p_2 = p_1.p_3$). We write $p|q$ iff neither $p \leq q$ nor $q \leq p$. The empty path (referring to root) is written as ε. We write $t[s]_p$ for the term resulting from the replacement at p of $t|_p$ in t by s. Following [HL91], we write $\mathcal{O}(s)$ for the *occurences* of s, that is $\{p | p \in s\}$.

We write $ofs(f(\vec{t})) = f$ for the *outermost function symbol* f of a term $f(\vec{t})$, $lhs(l \to r) = l$ for the *left hand side* l of a rule $l \to r$, and $rhs(l \to r) = r$ for the *right hand side* r of a rule $l \to r$.

A *context* is a 'term' containing one occurrence of a special symbol $\Box$, denoting an empty place. A context is generally denoted by $C[]$. If $t \in T(\Sigma)$ and t is substituted for $\Box$, the result is $C[t] \in T(\Sigma)$ and t is said to be a subterm of $C[t]$, notated as $C[t] \subseteq t$.

A *substitution* is a (total) map $\sigma : T(\Sigma) \mapsto T(\Sigma)$ satisfying

$$\forall f \in \mathcal{F} \ : \ \sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n)).$$

By convention, we often write t^σ for $\sigma(t)$.

A *rewrite rule* is a pair of terms written as $s \to t$ with $s, t \in T(\Sigma)$. It is assumed that the left-hand side s of a rule $s \to t$ is not a single variable, and that $var(t) \subseteq var(s)$.

A *term rewriting system* $\mathcal{R}$ consists of a signature Σ and a set of rewrite rules R over Σ.

A term rewriting system defines a *rewrite relation* $\to_{\mathcal{R}}$. Since the subscript $\mathcal{R}$ is usually clear from the context, it is omitted. The overloading of $\to$ is by convention.

$s \to t \stackrel{\text{def}}{\iff} \exists \sigma, p, u \to v \in R : s|_p = u^\sigma \wedge t = s[v^\sigma]_p$.

The sub-term u^σ is referred to as *redex* (for reducible expression); the sub-term v^σ, as *reduct*.

If we want to be specific about the rule and the redex position p, we write $s \xrightarrow{p}_{(l \to r)} t$.

We write $\xrightarrow{*}$ for the transitive reflexive closure of $\to$.

The rewrite relation is closed under contexts, i.e., if $s \to t$, then for all $C[]$, $C[s] \to C[t]$.

A series of terms $s = s_1, s_2, \ldots$ such that $s_1 \to s_2 \to \ldots$ is called a *rewrite sequence*. A term s is said to be in *normal form* if there is no t such that $s \to t$. A function-symbol f is called a *defined* function symbol if there is a rule $f(t_1, \ldots, t_n) \to r$. A function-symbol c is called a *constructor* symbol if there is a normal form in which it occurs, and a *free constructor* if it is not a defined symbol.

A TRS is called *left-linear* if all left-hand sides are linear. A TRS is called *confluent* if, for all terms t_1, t_2, t_3, we have that $t_1 \xrightarrow{*} t_2$ and $t_1 \xrightarrow{*} t_3$ implies that there exists a term t_4 such that $t_2 \xrightarrow{*} t_4$ and $t_3 \xrightarrow{*} t_4$. A TRS is called *terminating* if there are no infinite rewrite sequences. Note that confluence and termination are generally undecidable.

Let $r_1 : l \to r$ and $r_2 : g \to d$ be rewrite rules. If there exists a context $C[]$, a non-variable term s, and a substitution σ such that $l = C[s]$ and $s^\sigma = g^\sigma$, then g *overlaps with* l. We say there is overlap *between* a rule r and a TRS T iff either r overlaps with a rule of T, or there is a rule of T that overlaps with r.

A TRS is called *orthogonal* if it is left-linear, and there is no overlap between the rules.

Following [HL91], we write $\mathcal{R}(s)$ for the set of paths to redexes in s.

Given a rewrite step $A : s \xrightarrow{p_A}_{(l \to r)} t$ and $p \in \mathcal{R}(s)$, where there is no overlap between l and the rule of p, we define the set $p\backslash A$ of *residuals* or *descendants* of p by A as a subset of $\mathcal{O}(s)$:

$$p\backslash A = \begin{cases} \emptyset & \text{if } p = p_A; \\ \{p\} & \text{if } p | p_A \text{ or } p \leq p_A; \\ \{p_i p_n p_r |\ r|_{p_n} = x\} & \text{if } p = p_i p_m p_r \text{ and } l|_{p_m} = x \in \mathcal{V}. \end{cases}$$

For rewrite sequences, we define $p\backslash A$ by

$$\begin{cases} p\backslash \epsilon = \{p\} \\ p\backslash AB = \{p_a\backslash B |\ p_a \in p\backslash A\} \end{cases}$$

For orthogonal systems (where there is no overlap at all) these definitions generalize to the ones given in [HL91].

In general, a term may contain many redexes. A rewriting *strategy* determines which of these is chosen. Confluence guarantees unique normal forms, regardless of the strategy. A well-known strategy is *rightmost innermost*, which chooses the rightmost redex that does not contain another redex.

In *priority* rewrite systems (PRSs) [BBKW89], the rules are (partially) ordered, and a rule may be applied only if there are no applicable rules (i.e., even after reduction of subterms) with higher priority. We will also consider *syntactic priority*, in which the decision whether a rule is applicable is made without considering reductions of sub-terms.

The ordering we will use is *syntactic specificity ordering*, where a rule $l \to r > s \to t$, when there exists a substitition σ such that $s^\sigma = l$ (in [BBKW89], *specificity ordering* implies that

all ambiguities are between terms that are ordered according to specificity, which we do not demand for *syntactic specificity ordering*).

Under syntactic specificity ordering, any set of terms with the same outermost function symbol has a greatest lower bound (glb). We will call such a glb, a term of the form $f(\vec{x})$, a *most general* LHS.

A TRS is called *sufficiently complete* if defined functions do not appear in normal forms. In general, sufficient completeness is undecidable. We will call a TRS *simply complete* if every defined function has a most general rule. It is clear that simple completeness implies sufficient completeness.

3. Term Rewriting Simulations

In this section, we define the notion of *simulation* of a TRS by another TRS.

In principle, a TRS $T = (\Sigma, R)$ is simulated by a TRS $T' = (\Sigma', R')$ if every rewrite sequences w.r.t. R can be related to a rewrite sequence w.r.t. R'. To this end, there must be a map from $T(\Sigma')$ to $T(\Sigma)$, which is called the simulation map.

This notion of simulation can be developed for arbitrary relations, but we will only use it in the more limited context of (minimal) term rewriting systems. In that context, as we will see, it is preferable to regard a simulating TRS of which the signature is an extension of that of the simulated TRS (i.e., $\Sigma' \supseteq \Sigma$), and for which the simulation map is identity on the common set of terms $T(\Sigma)$.

3.1 Simulation maps between terms

Definition 1 *Let $\Sigma = (\mathcal{F}, \mathcal{V})$ and $\Sigma' = (\mathcal{F}', \mathcal{V}')$ be signatures, such that $\Sigma' \supseteq \Sigma$. A* simulation map *is a partial map $\mathcal{S} : \mathcal{F}' \rightarrow \mathcal{F}$ for which $\forall f \in \mathcal{F} : \mathcal{S}(f) = f$. Let $\mathcal{D}_\mathcal{S}$ be a predicate that holds precisely for all symbols in $\mathcal{F}'$ for which $\mathcal{S}$ is defined.*

Note that the composition of two simulation maps is again a simulation map.

Under this definition, symbols in the original signature simulate themselves, and a simulating TRS may use intermediate symbols (terms) which are not a simulation of any symbol (term) in $\mathcal{F}$.

We extend $\mathcal{S}$ and $\mathcal{D}_\mathcal{S}$ to $T(\Sigma')$ by (partial) homomorphic extension.

As an example, consider $\mathcal{F} = \{f, a\}$ and $\mathcal{F}' = \{f, a, f_c, h\}$. In this example, f_c is a variant (a so-called constructor variant, discussed further in the sequel) of f with $\mathcal{S}(f_c) = f$, and h is an auxiliary function that has no counterpart in $\mathcal{F}$. Supposing that the arity of f is 1, and the arity of a is 0, we have (by partial homomorphic extension) that $\mathcal{S}(f(f_c(a))) = f(f(a))$, and $\neg\mathcal{D}_\mathcal{S}(f(h(a)))$, so $\mathcal{S}(f(h(a)))$ is undefined.

3.2 Simulating Relations

Let $\Re = \langle \Sigma, R \rangle$ and $\Re' = \langle \Sigma', R' \rangle$ be TRSs, with the understanding that by R and R' we sometimes mean the rewrite *relation*, rather than the rewrite rules[1], and let $\mathcal{S} : \Sigma' \rightarrow \Sigma$ be a simulation map. We will define simulation of $\Re$ by $\Re'$ under $\mathcal{S}$. First, we define three auxiliary

[1] This makes it easier to discuss *restrictions* of the rewrite relation, e.g. the relation with only innermost rewrites.

notions: soundness, completenes and termination conservation. In the figures illustrating the definitions below, dashed arrows are implied by solid arrows, closed points are universally quantified, and open points are existentially quantified.

Soundness of the triple $(R, \mathcal{S}, R')$ means that sufficiently many sequences in R' are mapped (by $\mathcal{S}$) to sequences in R. If we have a sequence sR'^*t in the simulating system with $\mathcal{S}$ defined on s and t, it is only reasonable to call $(R, \mathcal{S}, R')$ *sound* when $\mathcal{S}(s)R^*\mathcal{S}(t)$, so the image of R'^* under $\mathcal{S}$ is contained in R^* (depicted in Fig. 1a). In case $\mathcal{S}$ is *not* defined on t, we do not want the sequence to 'escape into undefinedness', so we demand that there is some u with tR'^*u and $\mathcal{S}$ defined on u (depicted in Fig. 1b). Formally, soundness is defined in Definition 2.

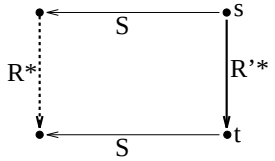


Fig. 1a.

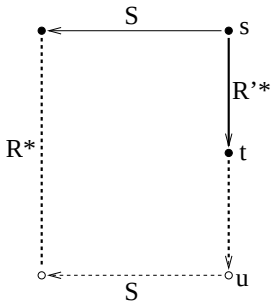
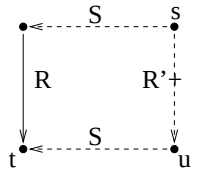


Fig. 1b.

Definition 2 *A simulation* $(\mathcal{S}, R')$ *of* R *is* sound *whenever*

$$\forall st\ (\mathcal{D}_{\mathcal{S}}(s) \wedge sR'^*t) \Longrightarrow \mathcal{S}(s)R^*\mathcal{S}(t) \vee (\neg\mathcal{D}_{\mathcal{S}}(t) \wedge \exists u \mathcal{D}_{\mathcal{S}}(u) \wedge tR'^*u)$$

The triple $(R, \mathcal{S}, R')$ is *complete*, when every step $\mathcal{S}(s)Rt$ in the simulated relation has as counterpart a simulating sequence sR'^+t, with $\mathcal{S}(u) = t$, provided s is reachable, i.e. $s_oR'^*s$, for some $s_o \in Ter(\Sigma)$, written $reachable\Sigma R'(s)$. This is defined formally in Definition 3, and depicted in Fig. 2.

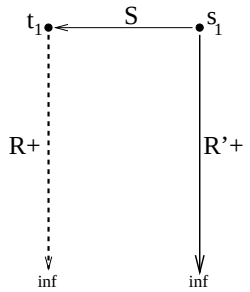


Definition 3 *A simulation* $(\mathcal{S}, R')$ *of a relation* R *is* complete *whenever*

$$\forall st\ reachable\Sigma R'(s) \wedge \mathcal{D}_{\mathcal{S}}(s) \wedge \mathcal{S}(s)Rt \Longrightarrow \exists u\ sR'^+u \wedge \mathcal{S}(u) = t$$

Fig. 2. Completeness

Termination conservation of the triple $(R, \mathcal{S}, R')$ means that only terms taking part in infinite sequences in R, have origins (under $\mathcal{S}$) occurring in infinite sequences in R'.



Definition 4 *A simulation* $(\mathcal{S}, R')$ *is* termination preserving *whenever*

$$\forall s \in inf(R') : \mathcal{D}_{\mathcal{S}}(s_1) \Longrightarrow \exists t \in inf(R)\ \mathcal{S}(s_1) = t_1$$

where $inf(R)$ *is the set of* infinite *sequences in* R*, and we denote the* i*th term in a rewrite sequence* s *by* s_i*.*

Fig. 3. Conservation of termination

We can now define *simulation*:

Definition 5 *(Simulation) Let* $\Re = \langle \Sigma, R \rangle$ *and* $\Re' = \langle \Sigma', R' \rangle$ *be TRSs with* $\Sigma \subseteq \Sigma'$ *and let* $\mathcal{S} : \Sigma' \to \Sigma$ *be a simulation map. We say that* $\Re$ is simulated by $\Re'$ under $\mathcal{S}$, *written as* $\Re \,|\wr_{\mathcal{S}}\, \Re'$, *iff the triple* $(R, \mathcal{S}, R')$ *is sound, complete and termination conserving.*

When $\mathcal{S}$ undefined on $\Sigma' \backslash \Sigma$, or if $\mathcal{S}$ is clear from the context, we will write $\Re \,|\wr\, \Re'$.

Normal forms, confluence, and strong and weak normalization are preserved under simulation. It is easy to verify that normal forms are preserved under simulation, that is, if we have $\mathcal{S}(m) = n$ with n a normal form then for all mR'^*m', we have that $\mathcal{S}(m') = n$, and from termination preservation it follows that there are no infinite sequences starting with m. Confluence follows directly from completeness. Conservation of strong normalization follows directly from termination preservation. With regard to weak normalization we remark that from completeness it follows that the sequence leading to a normal form n can be simulated.

Note that our notion of simulation is *transitive*: given that $\Re \,|\wr_{\mathcal{S}}\, \Re'$ and $\Re' \,|\wr_{\mathcal{S}'}\, \Re''$, we have that $\Re \,|\wr_{\mathcal{S} \odot \mathcal{S}'}\, \Re''$.

In a *simple simulation*, the effect of a single rule is simulated by a pair of complementary rules.

Lemma 1 *(Simple Simulation) Let* $\Re = \langle \Sigma, R \rangle$ *and* $\Re' = \langle \Sigma', R' \rangle$ *such that:*

1. $\Sigma' = \Sigma \cup \{f\}$ $(f \notin \Sigma)$;
2. $R = R_0 \cup \{r_0 : l \to r\}$ $(r_0 \notin R_0)$;
3. $R' = R_0 \cup \{r_1 : l \to f(\vec{t}), r_2 : f(\vec{x}) \to r'\}$;
4. $s \to_{r_0} t \wedge s \to_{r_1} s' \Leftrightarrow s \to_{r_1} s' \to_{r_2} t$;
5. *All (sub)terms occurring in* $\vec{t}$ *also occur in* l *or in* r.

Then $\Re \,|\wr_{\mathrm{I}_\Sigma}\, \Re'$.

Proof We have to prove completeness, soundness, and termination conservation of the triple $(R, \mathrm{I}_\Sigma, R')$.

Completeness is trivial, it follows directly from requirement 4.

For soundness, we first observe that given a sequence sR'^*t with $\mathcal{D}_{\mathcal{S}}(s) \wedge \mathcal{D}_{\mathcal{S}}(t)$ (i.e., both $s, t \in Ter(\Sigma)$) we have sR^*t. This follows from the fact that applications of rule r_2 are only possible on terms created by applications of rule r_1. Because r_2 has no overlap with other rules, and no redexes of r_2 remain in t (this follows from $\mathcal{D}_{\mathcal{S}}(t)$), we can replace the applications of r_1 in sR'^*t by applications of r_0, delete the applications of r_2, and thus obtain sR^*t. Second, we observe that when we have sR'^*t with $\neg\mathcal{D}_{\mathcal{S}}(t)$, this must be because there are some r_2-redexes left in t. We can rewrite these and obtain $sR'^*tR'^*t'$, with $\mathcal{D}_{\mathcal{S}}(t')$. Now, by the first observation, sR^*t, which completes the proof of soundness.

We prove termination conservation by considering the number of r_1-contractions. If there are no r_1-contractions in an infinite R'-sequence starting in a term t with $\mathcal{D}_{\mathcal{S}}(t)$, there are no r_2-contractions either, so the infinite sequence is itself an R-sequence. If there is only a finite number of r_1-contractions in an infinite sequence, there can only be an infinite number of r_2-contractions if there is some context $C[]$ in which (descendants of) an r_2 redex can be duplicated infinitely many times. But because r_2 has no overlap with other rules, this means that (descendants of) the r_1-redex can already be duplicated infinitely many times in $C[]$,

which is a contradiction, so all r_1 and r_2-contractions occur in a finite prefix of the infinite sequence, and the infinite suffix corresponds to an infinite R-sequence. Finally, if there is an infinite number of r_1 contractions, then there is also an infinite number of r_0-contractions possible, because all subterms in an instantiated RHS of r_1 are also in an instantiated RHS of r_0, and an instance of the RHS of r_1 itself can only be contracted by r_2, with the same result as a direct contraction by r_0 ■

4. Minimal Term Rewriting Systems

Here, we repeat the definition of *minimal term rewriting systems* (MTRSs), a syntactic restriction of TRSs that can be interpreted as the language of an abstract machine (see [WK95]).

In MTRSs, all rules have an extremely simple form. The most conspicuous aspect is that any rule has at most three function symbols, of which at most two are found on either side. Even the SKI calculus ([Klo92]), which is minimal in the number of rules (3), and in the total number of function symbols (4: S, K, I, and $\cdot$), needs 7 function symbols in its most complicated rule ($S{\cdot}x{\cdot}y{\cdot}z \to (x{\cdot}y){\cdot}(y{\cdot}z)$). Somewhat less conspicuous, but equally important for the interpretation as a machine language, is the fact that the 'action' (adding, changing or deleting function symbols or variables) performed by application of a rule is 'local', i.e. restricted to a number of consecutive arguments and the outermost function symbol.

Definition 6 *(MTRS) Let $\Re = \langle \Sigma, R\rangle$ be a TRS, and $r : s \to t$ a rule in R. The rule r is called* minimal *if it is left-linear and it is in one of the following six forms:*

$$
\begin{array}{rrcll}
\mathbf{C}: & f(\vec{x},\vec{y},\vec{z}) & \to & h(\vec{x},g(\vec{y}),\vec{z}) & \\
\mathbf{R}: & f(y) & \to & y & \\
\mathbf{M}: & f(\vec{x},g(\vec{y}),\vec{z}) & \to & h(\vec{x},\vec{y},\vec{z}) & \\
\mathbf{A}: & f(\vec{x},\vec{z}) & \to & h(\vec{x},y,\vec{z}) & (y \text{ is } x_i \text{ or } z_i)\\
\mathbf{D}: & f(\vec{x},\vec{y},\vec{z}) & \to & h(\vec{x},\vec{z}) & (\mid \vec{y} \mid \neq 0)\\
\mathbf{I}: & f(\vec{x}) & \to & h(\vec{x}) &
\end{array}
$$

A TRS $\Re$ is called a Minimal Term Rewriting System (MTRS) if all its rules are minimal.

We have labeled the forms with mnemonics reminding of their basic purpose (in the context of innermost rewriting). The mnemonic **C** stands for *continuation*, in the sense that h is the continuation after the evaluation of g. Conversely, **R** stands for *return*, in the sense that control is passed to a continuation if that was issued earlier, or rewriting is finished if there is no such continuation. Rules of the form **M** take apart a term, when there is a *match* of the symbol g. The forms **A**, **D** and **I** are for *addition, deletion* and *identity* on the set of variables.

5. An illustrative example compilation

Before we present our compilation technique in its general form, we would like to give an intuitive impression by showing how a concrete TRS is transformed into a simulating MTRS. Consider the following example of a simply complete TRS:

$$f(g(X),h(Y)) \to a(g(Y),X) \tag{5.1}$$

$$f(X,f(Y,Z)) \to b \tag{5.2}$$

$$f(X,Y) \to c \tag{5.3}$$

In Section 6.1, we will show how pattern-matching is simulated by a transformation. On the TRS above, this transformation would yield:

$$\begin{aligned}
f(g(X),Y) &\to f_g(X,Y) && (5.4)\\
f(X,Y) &\to f^S(X,Y) && (5.5)\\
f_g(X,h(Y)) &\to f_{gh}(X,Y) && (5.6)\\
f_g(X,Y) &\to f^S(g(X),Y) && (5.7)\\
f^S(X,f(Y,Z)) &\to f^S_f(X,Y,Z) && (5.8)\\
f^S(X,Y) &\to f^{SS}(X,Y) && (5.9)\\
f_{gh}(X,Y) &\to a(g(Y),X) && (5.10)\\
f^S_f(X,Y) &\to b && (5.11)\\
f^{SS}(X,Y) &\to c && (5.12)
\end{aligned}$$

The newly introduced functions f_g, f^S, f^S_g and f^{SS} can be understood as representants of the states of a matching automaton for LHS patterns (inspired by [HO82, Wal91, Pet92]). In this TRS, all rules are minimal, except for (5.10), in which a non-trivial RHS appears (the variables are in the wrong order). In Section 6.2, we will show how the construction of RHSs is simulated by a transformation. This transformation replaces (5.10) by the rules

$$\begin{aligned}
f_{gh}(X,Y) &\to a^R(X,Y,X) && (5.13)\\
a^R(X,Y,X') &\to a^{RR}(Y,X') && (5.14)\\
a^{RR}(Y',X') &\to a(g(Y'),X') && (5.15)
\end{aligned}$$

The result is an MTRS. The reader is invited to verify that this MTRS simulates the original TRS, using as simulation map I_Σ, the identity on Σ.

Now suppose that rule (5.3) were not present in the original system, then the system would not be simply complete. But, as is shown in general in Section 6.3, the TRS

$$\begin{aligned}
f(g(X),h(Y)) &\to a(g(Y),X) && (5.16)\\
f(X,f_c(Y,Z)) &\to b && (5.17)\\
f(X,Y) &\to f_c(X,Y) && (5.18)
\end{aligned}$$

simulates this system, when we assume innermost rewriting with specificity, and $\mathcal{S}(f_c) = f$. The idea is that rule (5.18) only applies when none of the other rules apply, so that in normal forms, f_c occurs exactly where in the original system f would have occurred. Rule (5.17) is adapted to match such normal forms (in innermost rewriting, the proper subterms of a redex are in normal form by definition).

6. Every simply complete TRS can be simulated by an MTRS

We will now show that every simply complete left-linear TRS can be simulated by an MTRS. We will give a constructive proof by providing a terminating transformation that transforms any simply complete left-linear TRS into a simulating MTRS. Because simulation is transitive, it suffices to prove simulation for every individual step of the transformation.

6.1 Transforming complicated LHSs

We will now present a specification of the function **sim**, and prove that applying **sim** to a simply complete TRS $\langle \Sigma, R\rangle$ yields a TRS $\langle \Sigma', R'\rangle$ such that $R \,|\!|_{\mathrm{I}_\Sigma} R'$). The specification of **sim** is itself nonambiguous and terminating, so it can be used as a pattern matching compiler.

In the specification, we will extensively use *union* of TRSs:

$$\langle \Sigma, R\rangle \cup \langle \Sigma', R'\rangle \quad = \quad \langle \Sigma \cup \Sigma', R \cup R'\rangle$$

Given some index set $I = \{i_1, \ldots, i_n\}$, we will use the notation $\bigcup_{i\in I} T_i$ for the (finite) union $T_{i_1} + \ldots + T_{i_n}$.

If all rules in R are *most general*, then:

$$[\text{sim-base}]\ \mathbf{sim}\langle \Sigma, R\rangle \quad = \quad \langle \Sigma, R\rangle.$$

Otherwise, let i be the least index such that for some rule $f(\vec{t}) \to s \in R$, we have $t_i \notin V$, and let $G = \{g | f(\vec{t}) \to r \in R \wedge t_i = g(\vec{u})\}$, the set of function symbols found at position i of LHSs defining f in R. Then, taking $|\ \vec{x}\ | = |\ \vec{t}\ | = i-1$, and $f^S, f_g \notin \Sigma$ fresh symbols, we have:

$$[\text{sim-rec}]\ \mathbf{sim}\langle \Sigma, R\rangle \quad = \quad (\textstyle\bigcup_{g\in G} \mathit{Match}_g \cup \mathit{Matched}_g) \cup \mathit{Skip} \cup \mathit{Other},$$

where

$$\begin{array}{rcl}
\mathit{Match}_g & = & \langle \{f, g, f_g\}, \{(m1:) f(\vec{x}, g(\vec{y}), \vec{z}) \to f_g(\vec{x}, \vec{y}, \vec{z})\}\rangle; \\
\mathit{Matched}_g & = & \mathbf{sim}\langle \Sigma \cup \{f_g, f^S\},\ \{(m2:) f_g(\vec{t}, \vec{u}, \vec{v}) \to r |\ (m3:) f(\vec{t}, g(\vec{u}), \vec{v}) \to r \in R\} \\
 & & \qquad \cup \{(m4:) f_g(\vec{x}, \vec{y}, \vec{z}) \to f^S(\vec{x}, g(\vec{y}), \vec{z})\}\rangle; \\
\mathit{Skip} & = & \langle \Sigma \cup \{f^S\}, \{(s1:) f(\vec{x}) \to f^S(\vec{x})\}\rangle; \\
\mathit{Other} & = & \mathbf{sim}\langle \Sigma \cup f^S,\ \{(o1:) f^S(\vec{t}) \to s |\ (o2:) f(\vec{t}) \to s \in R \wedge\ \vec{t}_i \in V\} \\
 & & \qquad \cup \{(o3:) r |\ r \in R \wedge \mathit{ofs}(\mathit{lhs}(r)) \neq f\}\rangle.
\end{array}$$

An intuitive explanation of [sim-rec] is, that Match_g has a rule $m1$ that matches a symbol g at position i, $\mathit{Matched}_g$ deals with a succesful match of g at position i, by either completing a match of $m3$ by applying $m2$, or restoring the LHS of $m1$ (up to f^S) by applying $m4$, *Skip* just replaces f by f^S (with the effect of sharing an 'automaton' state between reconstructed terms and terms for which matching fails right away), and *Other* simulates the rules $o2$ that have a variable at position i with rules $o1$, and rules $o3$ for other function symbols than f.

Let $NVP(R)$ be the number of paths to nonvariable proper subterms of LHSs of the rewrite rules R. It is clear that $NVP(R)$ is a well-founded measure on TRSs. It is easily established that, when read from left to right, the recursive rule [sim-rec] is decreasing in this measure. Furthermore, the conditions are decidable, so the specification of **sim** is an executable specification that can be used as a pattern-match compiler.

Theorem 1 *Let* $\langle \Sigma', R'\rangle = \mathbf{sim}(\langle \Sigma, R\rangle)$. *Then the triple* $(R, \mathrm{I}_\Sigma, R')$ *is* complete.

Proof By induction on $NVP(R)$. If $NVP(R) = 0$, we have case [sim-base], which is trivially complete. Otherwise, [sim-rec] must be applied. Because the simulation map is I_Σ, we only have to consider terms in $\mathit{Ter}(\Sigma)$. There are three cases: either a rule of type $m3$ is applied,

or another rule defining f, or a rule defining another function symbol than f. A rule of type $m3$ is simulated by applying $m1$ and then $m2$; other rules defining f are simulated by applying $s1$ and then $m2$; rules defining other symbols than f are available as rules of type $o3$ ■

Theorem 2 *Let* $\langle \Sigma', R' \rangle = \mathbf{sim}(\langle \Sigma, R \rangle)$. *Then the triple* $(R, \mathrm{I}_\Sigma, R')$ *is* sound.
Proof With induction on $NVP(R)$. When $NVP(R) = 0$, it is clear we have case [sim-base], and soundness is trivial. Otherwise, let i be the least index such that there is a rule $r \in R$ with $lhs(r)|_i \notin V$. Without loss of generality, we can assume r to be $f(\vec{t}, g(\vec{u}), \vec{v}) \to s$, with $|\ \vec{t}\ | = i - 1$. We have to prove that

$$\forall st\ (\mathcal{D}_\mathcal{S}(s) \wedge sR'^*t) \Longrightarrow \mathcal{S}(s)R^*\mathcal{S}(t) \vee (\neg\mathcal{D}_\mathcal{S}(t) \wedge \exists u \mathcal{D}_\mathcal{S}(u) \wedge tR'^*u)$$

By the induction hypothesis, we may assume $Matched_g$ to simulate

$$\begin{array}{l} \langle \Sigma \cup \{f_g, f^S\}, \\ \{(m2:) f_g(\vec{t}, \vec{u}, \vec{v}) \to r |\ (m3:) f(\vec{t}, g(\vec{u}), \vec{v}) \to r \in R\} \\ \cup \{(m4:) f_g(\vec{x}, \vec{y}, \vec{z}) \to f^S(\vec{x}, g(\vec{y}), \vec{z})\}\rangle \end{array},$$

and $Other$ to simulate

$$\begin{array}{l} \langle \Sigma \cup f^S, \\ \{(o1:) r^S |\ r \in R \wedge ofs(lhs(r)) = f \wedge lhs(r)|_i \in V\} \\ \cup \{(o3:) r |\ r \in R \wedge ofs(lhs(r)) \neq f\}\rangle \end{array}.$$

It is clear that when the sequence sR'^*t does not contain a (sub)term with a function symbol f_g or f^S, sR^*t holds trivially. A (sub)term with function symbol f_g is necessarily a descendant of a term introduced by rule $m1$. Such a term is a redex for $m4$, and potentially also for $m2$. Five things can happen to the descendants: they may be contracted according to $m2$ or $m4$, they may persist in t, they may be duplicated and they may be deleted by contraction of a higher redex. Duplication and deletion are trivially copied in R. If the redex persists in t, we can construct t' by applying $m4$, which brings us to the case where we have a (sub)term with function symbol f^S (see below). The result of a contraction according to $m2$ can be obtained in R by contracting the original term according to $m3$. The result of a contraction according to $m4$ brings us again to the case where we have a (sub)term with function symbol f^S. Apart from the two cases above, a (sub)term with function symbol f^S can be introduced by $s1$. Because of simple completeness, such a (sub)term is a redex of at least one of the rules in $Other$, but there can only be root-overlap with other rules, because f^S does not occur in Σ. Therefore, the descendants of this redex may be duplicated, deleted, or contracted according to one of the rules in $Other$. Duplication and deletion are again trivially copied in R, contraction according to $o1$ or $o3$ corresponds to a contraction according to $o2$ or $o3$, respectively. Finally, if a (sub)term with f^S is left in t, we have $\neg\mathcal{D}_\mathcal{S}(t)$, but we have tR'^*t' by a most general rule in $Other$. By the argument above, we now have that sRt' ■

Theorem 3 *Let* $\langle \Sigma', R' \rangle = \mathbf{sim}(\langle \Sigma, R \rangle)$. *Then the simulation* $(\mathrm{I}_\Sigma, \langle \Sigma', R' \rangle)$ *of* $\langle \Sigma, R \rangle$ *is* termination preserving *under innermost rewriting*
Proof Obvious by induction on $NVP(R)$ ■

6.2 Transforming Complicated RHSs

Here we present a transformation that will transform a TRS N, which may have RHSs that do not conform to the restrictions imposed on MTRSs, into a simulating TRS M, whose RHSs are minimal. Any rule with a minimal LHS and a non-minimal RHS has the form $l(\vec{x}, \vec{y}, \vec{z}) \to h(\vec{x}, \vec{t}, u, \vec{z})$, where u is either a variable (not equal to the last variable of $\vec{y}$) or a term $g(\vec{u})$, and $\vec{x}$ and $\vec{z}$ contain only variables, and are taken of maximal length. The goal is to reduce the non-compliant segment $\vec{t}, u$.

In case u is a variable, we replace the rule by the following rules:

$$l(\vec{x}, \vec{y}, \vec{z}) \to h^R(\vec{x}, \vec{y}, u, \vec{z}) \tag{6.19}$$

$$h^R(\vec{x}, \vec{y}, u', \vec{z}) \to h(\vec{x}, \vec{t}, u', \vec{z}), \tag{6.20}$$

Where u' is a fresh variable. Rule (6.19) is an instance of **A**, and rule (6.20) has a shorter non-compliant segment $\vec{t}$.

In case $u = g(\vec{u}$, we replace the rule by the following rules:

$$l(\vec{x}, \vec{y}, \vec{z}) \to h^R(\vec{x}, \vec{t}, \vec{u}, \vec{z}) \tag{6.21}$$

$$h^R(\vec{x'}, \vec{y'}, \vec{z}) \to h(\vec{x'}, g(\vec{y'}), \vec{z}) \tag{6.22}$$

where $|\vec{x'}| = |\vec{x}| + |\vec{t}|$, $|\vec{y'}| = |\vec{u}|$; h^R is a fresh function symbol which did not already occur in the TRS; and $\vec{x'}$ and $\vec{y'}$ consist entirely of fresh variables. Rule (6.21) contains one function symbol less than the original rule, and rule (6.22) is an instance of **C**.

We take the simulation map $\mathcal{S}$ to be undefined for h^R. Repeated application of the transformation above to a TRS with minimal LHSs leads to an MTRS.

We note that both the variable case and the nonvariable case fulfill the requirements for a simple simulation (see Lemma 1), so the RHS transformation yields a simulating TRS.

6.3 Simulating General Left-Linear TRSs by Simply Complete left-linear TRSs

Until now, we have only dealt with simply complete TRSs. Unfortunately, simple completeness is a rare property. Here we will show that, under the restriction to innermost rewriting with syntactic specificity ordering, every TRS can be simulated by a simply complete TRS.

Let the TRS $\langle \Sigma, R \rangle$ be given, and let $\Sigma_p \subseteq \Sigma$ be the set of function symbols for which R has no most general rule. Let Σ_c contain a so-called *constructor variant* f_c for every $f \in \Sigma_u$, and let $\mathcal{S}(f_c) = f$. Given a term t or a sequence $\vec{t}$, define t_c or $\vec{t}_c$ to be the term or sequence obtained by replacing all $f \in \Sigma_u$ by their constructor variants f_c. Taking $R' = \{o1 : f(\vec{t}_c) \to s | f(\vec{t}) \in R\} \cup \{c1 : f(\vec{x}) \to f_c(\vec{x}) | f \in \Sigma_u\}$, we have obtained a simply complete TRS $\langle \Sigma \cup \Sigma_c, R' \rangle$.

It is easy to see that the triple $(\langle \Sigma, R \rangle, \mathcal{S}, \langle \Sigma', R' \rangle)$ is sound, complete and termination conserving, so $R \,|\!|_{\mathcal{S}}\, R'$. For soundness, we observe that, given a rewrite sequence $t_1 \to t_2 \to \ldots t_n$ in R', it follows that either $\mathcal{S}(t_i) = \mathcal{S}(t_{i+1})$ (in case $c1$ is applied), or $\mathcal{S}(t_i) R \mathcal{S}(t_{i+1})$ (in case $o1$ is applied), so always $t_1 R * t_n$.

For completeness, we observe that a step $\mathcal{S}(t_i) R \mathcal{S}(t_{i+1})$ may not be possible in R' because some subterms of t_i have an original function symbol where a constructor variant is needed. We may first rewrite exactly these subterms with rules of type $c1$, however, $t_i R' * t_i'$, with

$\mathcal{S}(t_i') = \mathcal{S}(t_i)$, and then we have $t_i' R' t_{i+1}$. Because a step $t_i R t_{i+1}$ is only taken when all subterms of t_i are already in normal form, rewriting t_i to t_i' does not invalidate future R-rewrites and because of specificity ordering, rules of type $c1$ can only rewrite terms t for which $\mathcal{S}(t)$ is a normal form.

Finally, conservation of termination follows from the fact that only a finite number of applications of rules of type $c1$ is possible on any term, so if there is an infinite reduction on t according to R', there is necessarily an infinite reduction on $\mathcal{S}(t)$ according to R.

Without proof, we mention that non-linear TRSs can be simulated in a simular vein, under the same restrictions and for given Σ.

6.4 Efficiency Considerations

The efficiency of compilers can be expressed by several measures:

- The size (in number of rules) of the target program;
- The time and space taken for compilation from source to target language;
- The time and space taken by an execution of the target program, compared to the time and space taken by execution of the source program.

It is clear that the size of the target program depends in a linear fashion on the total number of occurrences of function symbols in the source program, and rules in the target program are at least as simple as the rules in the source program.

With regard to the space taken by the compilation, we observe that the number of new rules constructed depends in a linear fashion on the total number of occurrences of function symbols in the source program. Thus, a naive implementation needs at most an amount of space linear in the size of the source program.

With respect to the time taken, even a naive implementation of **sim** that scans all rules to find a rule with nonvariable arguments, will only be quadratic in the number of rules and linear in the number of symbol occurrences in LHSs.

Considering the time taken by the execution of the target program, we remark that indeed the number of rewriting steps is linearly increased by the compilation. The complexity of executing a single step, however, is decreased. In practice, this leads to comparable performance. The big gain, however, is in the fact that the MTRS can easily be seen as a program to be executed by a concrete machine (see [WK95]). The resulting machine code and its performance is similar to that of existing compilers for functional languages [HF+96].

7. Conclusions and Future Work

We have presented transformations from arbitrary left-linear TRSs into simulating MTRSs. The transformations can be expressed in a concise way, and their correctness proofs are short and easy to grasp. Furthermore, the transformations are described as executable specifications, which can be used as an efficient compiler. The resulting code is similar to the code generated by an earlier version of our TRS compiler, with which favorable results have been reached [HF+96].

In [KW95], we presented a transformation to simulate lazy rewriting by eager (innermost) rewriting. It appears that this transformation can be simplified greatly by first applying the

transformations in this paper, and then the laziness transformation, which is much simpler when only MTRSs have to be considered.

Similarly, we expect that the transformations given in this paper could simplify other research on TRSs (e.g., Hans Zantema suggested that termination proofs might be simpler after our transformations, but we have not yet investigated this issue in any depth).

In the future, we hope to find a bigger class of TRSs for which a strategy-independent simulation by MTRSs can be given. For our current implementation requirements, however, the current class is sufficient.

An interesting class of TRSs (which unfortunately has no inclusion relation with simply complete TRSs) is the class that admits *specificity ordering* as defined in [BBKW89]. It appears that applying the transformation in this paper to a member of this class yields a simulating MTRS, if we consider the priority rewrite relation, without any further assumption about the strategy. We would like to establish this rigorously.

Finally, we thank Bas Luttik for commenting on various drafts of this paper.

REFERENCES

[ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers. Principles, Techniques and Tools*. Addison-Wesley, 1986.

[BBKW89] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, and W.P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, 67(1):283–301, 1989.

[DJ90] N. Dershowitz and J.-P Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol B.*, pages 243–320. Elsevier Science Publishers, 1990.

[HF+96] Pieter H. Hartel, Marc Feeley, et al. Benchmarking implementations of functional languages with "pseudoknot", a float-intensive benchmark. *Journal of Functional Programming*, 1996. Accepted for publication.

[HG94] Lutz H. Hamel and Joseph A. Goguen. Towards a provably correct compiler for OBJ3. In *Proceedings of the International Conference on Programming Language Implementation and Logic Programming, PLILP '94*, 1994.

[HL91] Gérard Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, I. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in honor of Alan Robinson*, chapter 11. MIT Press, Cambridge, Massachusetts, 1991.

[HO82] C.M. Hoffmann and M.J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 2.*, pages 1–116. Oxford University Press, 1992.

[KW95] J.F.Th. Kamperman and H.R. Walters. Lazy rewriting and eager machinery. In Jieh Hsiang, editor, *Rewriting Techniques and Applications*, number 914 in Lecture Notes in Computer Science, pages 147–162. Springer-Verlag, 1995.

[Pet92] Mikael Pettersson. A term pattern-match compiler inspired by finite automata theory. In U. Kastens and P. Pfahler, editors, *Proceedings of the Fourth Inter-*

national Conference on Compiler Construction, number 641 in Lecture Notes in Computer Science, pages 258–270. Springer-Verlag, 1992.

[Wal91] H.R. Walters. *On Equal Terms, Implementing Algebraic Specifications.* PhD thesis, University of Amsterdam, 1991. Available by *ftp* from ftp.cwi.nl:/pub/gipe/reports as Wal91.ps.Z.

[WK95] H.R. Walters and J.F.Th. Kamperman. Minimal term rewriting systems. Technical Report CS-R9573, CWI, december 1995. Submitted for publication. Available as http://www.cwi.nl/ gipe/epic/articles/CS-R9573.ps.Z.