Correct Transformation of Rewrite Systems for Implementation Purposes

Wan Fokkink & Jaco van de Pol
Utrecht University, Department of Philosophy
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
fokkink@phil.ruu.nl   jaco@phil.ruu.nl

Abstract

We propose the notion of a *correct transformation* of one rewrite system into another. If such a transformation is correct, then the normal forms of a term in the original rewrite system can be obtained by computing the normal forms of the interpretation of this term in the transformed rewrite system. We show for several transformations from the literature that they are correct, most notably for the notion of simulation from Kamperman and Walters.

1 Introduction

Quite a number of papers deal with particular examples of transformations of rewrite systems, usually with the aim to obtain a rewrite system which satisfies some desirable property, e.g. [17, 12, 2, 18, 16, 20, 19, 23, 9, 10, 8]. In most of these papers, correctness of the transformation is stated, meaning that the original and the transformed rewrite system are in some sense 'equivalent'. This claim is based on the observation either that desirable properties such as confluence and termination are preserved by the transformation, or that the transformed system can somehow simulate the original system, so that the reduction tree of an original and a simulating term have the same structure. In this paper we formulate general conditions which ensure that a transformation of rewrite systems constitutes a correct implementation step.

Recently, Kamperman and Walters [9, 10] proposed a notion of simulation of one rewrite system by another. They apply simulation to transform a rewrite system into a so-called 'minimal' rewrite system, which has a form more suitable for compilation into an abstract machine. This transformation constitutes a step in the implementation of their functional programming language EPIC [21, 22]. Luttik [13] proposes a series of stronger simulation notions, and derives desirable properties for them, such as termination and confluence.

Kamperman and Walters state, for example in the title of [10], that simulation constitutes a correct transformation of rewrite systems. However, they do not provide any further foundation for this claim. Unfortunately, the definition of simulation is quite complex, so that it is not so easy to grasp its intuition. Also, at first sight the link between the original and the simulating rewrite system is unclear. For example, in general the syntax of the original and of the simulating rewrite system differ. Furthermore, the original rewrite system may be confluent, while the simulating rewrite system is not. Hence, the question arises what it means to state that such a transformation of rewrite systems is 'correct'.

Although preservation of reduction trees underlies simulation, this property is usually not of interest for applications of rewriting systems. Especially if a rewrite system is used to implement a functional language, then one is solely interested in the input/output behaviour of the system, where the input is any term, and the output is (one of) its normal form(s). So if a transformation of rewrite systems is part of an implementation project, then the main interest is that the transformation preserves normal forms.

In this paper, we propose the notion of a *correct transformation* of rewrite systems. Basically, a transformation of one rewrite system into another is correct if no information on normal forms in the original rewrite system is lost. That is, there should exist mappings *parse* from original to transformed terms and *print* from transformed to original terms such that for each original term t one can compute its normal forms as follows: compute the normal forms of the term $parse(t)$, and apply the *print* function to them. Furthermore, it is required that a correct transformation preserves termination properties.

We will confirm the claim of Kamperman and Walters, that is, we will show that the notion of simulation as proposed in [9, 10] constitutes a correct transformation. In the presentation, we will generalize and simplify some of the original simulation definitions. The proof of the correctness result will use the criteria for a simulation almost in full. One could therefore argue that the simulation definition has been designed to satisfy the requirements of a correct transformation implicitly. We will also briefly study several other examples of transformations of rewrite systems, to decide whether or not they satisfy our correctness criteria.

Acknowledgements. Jasper Kamperman and Bas Luttik are thanked for useful discussions.

2 Abstract Reduction Systems

This section introduces some preliminaries from rewriting; for more background see [5, 11]. We will focus on abstract reduction systems instead of on term rewriting systems, in order to emphasize the generality of our approach.

Definition 2.1 *An* abstract reduction system (ARS) *consists of a collection A of elements, together with a binary reduction relation R between elements in A.*

We will write R^+ for the transitive closure of a reduction relation R, and R^* for the reflexive transitive closure of R.

In the following definitions, we assume an ARS (A, R).

Definition 2.2 $a \in A$ *is a* normal form (for R) *if there does not exist an $a' \in A$ such that aRa'.*

$a \in A$ *is a* normal form of $a' \in A$ *if* $a'R^*a$ *and* a *is a normal form.*

$\mathit{nf}_R : A \to \mathcal{P}(A)$ denotes the mapping that assigns to each $a \in A$ the collection of its normal forms.

Definition 2.3 *R is* terminating *for $a \in A$ if there does not exist an infinite reduction $aRa_1Ra_2R\cdots$. (This is also known as strong normalization.)*
R is weakly terminating *for $a \in A$ if $\mathit{nf}_R(a) \neq \emptyset$.*
(A, R) is (weakly) terminating *if R is (weakly) terminating for each $a \in A$.*

Proposition 2.4 *termination $\Rightarrow$ weak termination.*

Definition 2.5 *R is* confluent *for $a \in A$ if for each pair of reductions aR^*a_1 and aR^*a_2 there exists an $a_3 \in A$ such that $a_1R^*a_3$ and $a_2R^*a_3$.*
(A, R) is confluent *if R is confluent for each $a \in A$.*

3 Correct Transformations

The aim of this paper is to formulate general conditions which ensure that a transformation of rewrite systems is correct. We adopt the point of view that a transformation is correct if it constitutes a sensible step in an implementation procedure. This is the case if the input/output behaviour of the system is maintained, where the input is any term, and the output is (one of) its normal form(s). Hence, for us the prime interest of a transformation is that it preserves normal forms.

In [3], the distinction is made of 'control' versus 'computation', which in rewriting would be the internal structure of a reduction tree versus its eventual normal forms. We note that rewriting is mostly concerned with the computational aspect, that is, a rewrite system is characterized by the normal forms that it attaches to terms, together with its termination properties. For example:

- in equational theorem proving one is mostly concerned with terminating rewrite systems which yield unique normal forms [15];
- if rewriting is applied to implement abstract data types, then the meaning of a term is fixed by its normal forms [4];
- in [5] it is remarked that "rewrite systems defining at most one normal form for any input term can serve as functional programs".

We propose the notion of a *correct transformation* of rewrite systems. Basically, a transformation of one rewrite system into another is correct if no information on normal forms in the original rewrite system is lost. That is, there should exist mappings *parse* from original to transformed terms and *print* from transformed to original terms such that for each original term t one can compute its normal forms as follows: compute the normal forms of the term $\mathit{parse}(t)$, and apply the *print* function to them.

Furthermore, if the original rewrite system is terminating for a certain term, then in the implementation one can be sure that the normal form of such a term can be computed, simply by applying the rewrite rules sufficiently many times. Hence, we require that the transformed rewrite system is terminating for the parsed version of such a term.

In the following definitions, we assume that a mapping $f : V \to W$ extends to a mapping $f : \mathcal{P}(V) \to \mathcal{P}(W)$ as expected: $f(V_0) = \{f(v) \mid v \in V_0\}$.

Definition 3.1 *An ARS (B, S) is a* correct transformation *of an ARS (A, R) if there exist mappings parse : $A \to B$ and print : $B \to A$ such that:*

1. *if R is terminating for $a \in A$, then S is terminating for parse(a);*
2. *print(nf$_S$(parse(a))) $=$ nf$_R(a)$, that is, the diagram below commutes:*

$$\begin{tikzcd} A \arrow[r, "parse"] \arrow[d, "nf_R"'] & B \arrow[d, "nf_S"] \\ \mathcal{P}(A) & \mathcal{P}(B) \arrow[l, "print"] \end{tikzcd}$$

For the implementation of a rewrite system often a specific reduction strategy is selected, so that for each term only one of its possible reductions is implemented. Then for each term no more than one of its normal forms is preserved, so if the original rewrite system is not confluent then in general such an implementation does not constitute a correct transformation. Therefore, we propose the notion of a *weakly correct transformation*, which requires that for each original term t which has one or more normal forms, at least one of these normal forms is obtained by computing the normal forms of *parse*(t) in the transformed rewrite system and applying the *print* function to them. As before, we also require that if the original rewrite system is terminating for some term, then the transformed rewrite system is terminating for the parsed version of this term.

Definition 3.2 *An ARS (B, S) is a* weakly correct transformation *of an ARS (A, R) if there exist mappings parse : $A \to B$ and print : $B \to A$ such that for each $a \in A$:*

1. *if R is terminating for $a \in A$, then S is terminating for parse(a);*
2. *print(nf$_S$(parse(a))) $\subseteq$ nf$_R(a)$;*
3. *if R is weakly terminating for $a \in A$, then S is weakly terminating for parse(a).*

Proposition 3.3 *correctness $\Rightarrow$ weak correctness.*

Proof. Suppose that (B, S) is a correct transformation of (A, R).

If R is terminating for some $a \in A$, then by definition of correctness S is terminating for *parse*(a).

Furthermore, since *print*(*nf*$_S$(*parse*(a))) equals *nf*$_R(a)$, in particular it is a subset of *nf*$_R(a)$.

Finally, let $a \in A$ with $nf_R(a) \neq \emptyset$. Then $print(nf_S(parse(a))) = nf_R(a) \neq \emptyset$, so also $nf_S(parse(a)) \neq \emptyset$. $\Box$

We note that for confluent rewrite systems, weak correctness agrees with correctness.

Proposition 3.4 *A transformation of a confluent ARS* (A, R) *into an ARS* (B, S) *is correct if and only if it is weakly correct.*

Proof. According to Proposition 3.3 correctness implies weak correctness, so we only need to prove the reverse. Assume that (B, S) is a weakly correct transformation of (A, R), and that (A, R) is confluent. We show that this transformation is correct.

If R is terminating for some $a \in A$, then by definition of weak correctness S is terminating for $parse(a)$.

Fix an $a \in A$; we show that $print(nf_S(parse(a))) = nf_R(a)$. Since (A, R) is confluent, clearly $nf_R(a)$ can contain no more than one element. We distinguish two cases.

- $nf_R(a) = \emptyset$. Weak correctness implies that $print(nf_S(parse(a)))$ is contained in $nf_R(a)$, so then it is also empty.

- $nf_R(a)$ contains one element. Then R is weakly terminating for a, so weak correctness implies that S is weakly terminating for $parse(a)$. Then $nf_S(parse(a))$ is not empty, so the same holds for $print(nf_S(parse(a)))$. Since this last collection is contained in $nf_R(a)$, which contains only one element, it follows that $print(nf_S(parse(a))) = nf_R(a)$. $\Box$

3.1 An Example

We present an example of a transformation, which will be shown to be correct later on. In the next section, it will be used as a running example.

Example 3.5 Assume the constant 0, the unary successor function *succ*, and the binary addition +. Let R be the following standard implementation of addition on the natural numbers over $T(\{0, succ, +\})$, which consists of the closed terms over $\{0, succ, +\}$.

$$\begin{array}{rcl} x + 0 & \longrightarrow & x \\ succ(x) + y & \longrightarrow & succ(x + y) \end{array}$$

In so-called 'minimal' rewrite systems [9, 10], rewrite rules are not allowed to contain more than three function symbols. Note that the second rule of R does not satisfy this requirement. In order to obtain a minimal rewrite system, the second rule in R can be replaced by two new rules, which contain an auxiliary binary function symbol f. Furthermore, for the sake of this toy example, the + is replaced by its reverse, denoted by $\oplus$. Thus, R is transformed into the following minimal rewrite system S over $T(\{0, succ, \oplus, f\})$:

$$\begin{array}{rcl} 0 \oplus x & \longrightarrow & x \\ x \oplus succ(y) & \longrightarrow & f(x, y) \\ f(x, y) & \longrightarrow & succ(x \oplus y) \end{array}$$

Define

$$\begin{array}{rcl@{\qquad}rcl} \mathit{parse}(0) & = & 0 & \mathit{print}(0) & = & 0 \\ \mathit{parse}(\mathit{succ}(x)) & = & \mathit{succ}(\mathit{parse}(x)) & \mathit{print}(\mathit{succ}(x)) & = & \mathit{succ}(\mathit{print}(x)) \\ \mathit{parse}(x+y) & = & \mathit{parse}(y) \oplus \mathit{parse}(x) & \mathit{print}(x \oplus y) & = & \mathit{print}(y) + \mathit{print}(x) \end{array}$$

Note that the *print* function is only partially defined, for terms in $T(\{0, \mathit{succ}, \oplus\})$. It is not hard to verify the following properties:

1. the mappings *parse* and *print*, restricted to $T(\{0, \mathit{succ}, \oplus\})$, are each other's inverses;

2. if $t \longrightarrow t'$ in R, then $\mathit{parse}(t) \longrightarrow^{+} \mathit{parse}(t')$ in S;

3. if $\mathit{parse}(t) \longrightarrow^{*} u$ in S, then there exists a t' in $T(\{0, \mathit{succ}, +\})$ such that $u \longrightarrow^{*} \mathit{parse}(t')$ in S and $t \longrightarrow^{*} t'$ in R;

4. S is terminating.

We will see in the next section that these properties together ensure that the transformation is correct.

4 Application to Simulation

4.1 Simulation

Kamperman and Walters [9, 10] propose a notion of simulation for rewrite systems, which they apply to transform rewrite systems into so-called 'minimal' rewrite systems. Their definitions are presented in the next sections. In several cases we propose simplifications and/or generalizations of the original definitions.

A simulation of an ARS (A, R) by an ARS (B, S) is characterized by two mappings $\phi : B \to A$ and $\psi : A \to B$. The intuition for the mapping ϕ, which in general is only partially defined, is that the reduction tree of $a \in A$ with respect to R is mimicked by the reduction tree of each $b \in \phi^{-1}(a)$ with respect to S. The mapping ψ selects for each $a \in A$ an interpretation in $\phi^{-1}(a)$, so in particular $\phi(\psi(a)) = a$.

Definition 4.1 *A* simulation *of an ARS (A, R) by an ARS (B, S) consists of two mappings:*

1. *a partially defined mapping $\phi : B \to A$;*

2. *a mapping $\psi : A \to B$ such that $\phi(\psi(a)) = a$ for each $a \in A$.*

Note that the transformation described in Example 3.5 is a simulation, if we put $\phi = \mathit{print}$ and $\psi = \mathit{parse}$.

The second condition in Definition 4.1 implies that ϕ is surjective and that ψ is injective. However, in examples of simulation that occur in the literature, typically

ϕ is *not* injective and ψ is *not* surjective. In most of these examples, A is a proper subset of B and ψ is simply the identity mapping.

At several points, our notions for simulation are more general than as formulated in [9, 10]. There, only term rewriting systems are considered, where A is a proper subset of B, and ψ is the identity mapping. Furthermore, ϕ is required to be a homomorphism with respect to terms, inspired by the fact that this is usually the case with practical examples of simulations. However, this requirement does not serve any further purpose, and it cannot be formulated in the setting of ARSs.

4.2 Soundness and Completeness

In this and the following sections we assume as general notation that the ARS (A, R) is simulated by the ARS (B, S) by means of the mappings $\phi : B \to A$ and $\psi : A \to B$.

Suppose that $\phi(b)$ is defined for some $b \in B$. Soundness of the simulation means that each finite S-reduction of b is a mimicking of some finite R-reduction of $\phi(b)$.

Definition 4.2 *(Soundness) A simulation is* sound *if for each $b, b' \in B$ with $\phi(b)$ defined and bS^*b', there is a $b'' \in B$ with $b'S^*b''$ and $\phi(b'')$ defined and $\phi(b)R^*\phi(b'')$.*

It follows easily from property 3 (together with property 1) in Example 3.5 that the simulation described there is sound.

As opposed to soundness, completeness means that each R-step from $\phi(b)$ can be mimicked by a finite S-reduction of b with length greater than zero.

Definition 4.3 *(Completeness) A simulation is* complete *if for each $a \in A$ and $b \in B$ with $\phi(b)$ defined and $\phi(b)Ra$, there is a $b' \in B$ with bS^+b' and $\phi(b')$ is defined and $\phi(b') = a$.*

It follows easily from property 2 (together with property 1) in Example 3.5 that the simulation described there is complete.

We also define a weaker completeness notion, which helps to ensure that if there exist R-steps from $\phi(b)$, then at least one of these R-steps can be mimicked by a finite S-reduction of b with length greater than zero.

Definition 4.4 *(Weak completeness) A simulation is* weakly complete *if for each $b \in B$ with $\phi(b)$ defined and b a normal form for S, $\phi(b)$ is a normal form for R.*

It is not hard to see that the composition of two simulations is again a simulation. Moreover, soundness and completeness and weak completeness are preserved under composition.

Proposition 4.5 *completeness $\Rightarrow$ weak completeness.*

Proof. Suppose that the ARS (B, S) simulates the ARS (A, R) by means of the pair (ϕ, ψ), and that this simulation is complete.

Let $b \in B$ with $\phi(b)$ defined and $\phi(b)$ not a normal form for R. Then $\phi(b)Ra$ for some $a \in A$, so completeness yields that bS^+b' for some $b' \in B$. Hence, b is not a normal form for S. So the simulation is weakly complete. $\square$

4.3 Termination Conservation

The properties for simulations that are formulated in the next two definitions ensure that termination qualities for the original rewrite system are preserved by the simulating rewrite system. Total conservation (Definition 4.6) ensures that termination properties are preserved with respect to the mapping ϕ, while conservation (Definition 4.7) only ensures that termination properties are preserved with respect to the mapping ψ.

For textual convenience we adopt the convention that formulations which contain occurrences of the expression '(weak)' or '(weakly)' can be read both with and without the word 'weak' or 'weakly' at those places, respectively.

Definition 4.6 *(Total conservation of (weak) termination) A simulation* totally conserves (weak) termination *if for each $a \in A$ for which R is (weakly) terminating, also S is (weakly) terminating for each $b \in \phi^{-1}(a)$.*

Definition 4.7 *(Conservation of (weak) termination) A simulation* conserves (weak) termination *if for each $a \in A$ for which R is (weakly) terminating, also S is (weakly) terminating for $\psi(a)$.*

Proposition 4.8

1. *total conservation of (weak) termination $\Rightarrow$ conservation of (weak) termination;*
2. *total conservation of termination + completeness $\Rightarrow$ total conservation of weak termination*

Proof. We assume that the ARS (B, S) simulates the ARS (A, R) by means of the pair (ϕ, ψ).

1. If the simulation totally conserves (weak) termination, then it also conserves (weak) termination, simply because $\psi(a) \in \phi^{-1}(a)$ for each $a \in A$.

2. Suppose that the simulation (ϕ, ψ) totally conserves termination and is complete. Let R be weakly terminating for $a \in A$, and let $b \in \phi^{-1}(a)$. We show that S is weakly terminating for b, by induction on the length of the shortest normalization reduction for a.

   If a is a normal form for R, then total conservation of termination yields that S is terminating for $b \in \phi^{-1}(a)$, so according to Proposition 2.4 S is also weakly terminating for b.

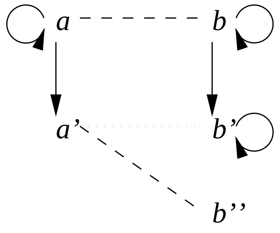
   Next, suppose that we have proved the case for normalization reductions of length n, and let the shortest normalization reduction for a have length $n+1$. Then there exists a reduction aRa' where the shortest normalization reduction for a' has length n. Since $\phi(b) = a$, completeness yields that bS^+b' for some $b' \in B$ with $\phi(b') = a'$. Since R is weakly terminating for a' with a shortest normalization reduction of length n, induction yields that S is weakly terminating for b'. Since bS^+b', it follows that S is also weakly terminating for b.

We note that the second part of Proposition 4.8 would not hold if the adjective 'total' were omitted from it, that is, a complete simulation which conserves termination does not necessarily conserve weak termination. This is shown in the following example.

Example 4.9 Let $A = \{a, a'\}$ and aRa and aRa'. Furthermore, let $B = \{b, b', b''\}$ and bSb and bSb' and $b'Sb'$. Define a simulation (ϕ, ψ) as follows:

$$\begin{array}{ll} \phi(b) = a & \psi(a) = b \\ \phi(b') = a' & \\ \phi(b'') = a' & \psi(a') = b'' \end{array}$$

This simulation can be depicted as follows:



This simulation is sound and complete, and it conserves termination: R is only terminating for a', and S is terminating for $\psi(a') = b''$. However, this simulation does not conserve weak termination: (A, R) is weakly terminating, but S is not weakly terminating for $\psi(a) = b$.

4.4 Reachability

Thatte [18] and Verma [20] studied a transformation of rewrite systems into so-called constructor-based rewrite systems, and they concluded that their transformation preserves normal forms for what they called the *reachable* part of the transformed system. In our terminology, their reachability notion can be formulated as follows, where as before we assume that (A, R) is simulated by (B, S) through (ϕ, ψ).

Definition 4.10 *(Reachability) $b \in B$ is* reachable *if $\psi(a)S^*b$ for some $a \in A$.*

Lemma 4.11 *Let (ϕ, ψ) be a simulation of (A, R) by (B, S), and let $\bar{\phi}$ denote the restriction of ϕ to the reachable part of B. Then $(\bar{\phi}, \psi)$ is also a simulation of (A, R) by (B, S). Furthermore, if (ϕ, ψ) satisfies soundness or (weak) completeness or (total) conservation of (weak) termination, then $(\bar{\phi}, \psi)$ also satisfies this property.*

Proof. Clearly $\psi(a)$ is reachable for each $a \in A$. Hence, $\bar{\phi}$ is defined for each $\psi(a)$, and $\bar{\phi}(\psi(a)) = a$. So $(\bar{\phi}, \psi)$ is a simulation.

Assume that (ϕ, ψ) is sound; we show that $(\bar{\phi}, \psi)$ is also sound. Let $\bar{\phi}(b)$ be defined, and bS^*b'. Then soundness of (ϕ, ψ) yields that $b'S^*b''$ where $\phi(b'')$ is defined and $\phi(b)R^*\phi(b'')$. Since b is reachable and $bS^*b'S^*b''$, it follows that b'' is also reachable. Hence, $\bar{\phi}$ is defined for b'', and $\bar{\phi}(b)R^*\bar{\phi}(b'')$. So $(\bar{\phi}, \psi)$ is sound.

Assume that (ϕ, ψ) is complete; we show that $(\bar{\phi}, \psi)$ is also complete. Let $\bar{\phi}(b)$ be defined, and $\bar{\phi}(b)Ra$. Then completeness of (ϕ, ψ) yields that bS^+b' where $\phi(b') = a$. Since b is reachable and bS^+b', it follows that b' is also reachable. Hence, $\bar{\phi}$ is defined for b', and $\bar{\phi}(b') = a$. So $(\bar{\phi}, \psi)$ is complete.

Finally, if (ϕ, ψ) totally conserves (weak) termination, then the same holds for $(\bar{\phi}, \psi)$, because $\bar{\phi}^{-1}(a) \subseteq \phi^{-1}(a)$ for all $a \in A$. And if (ϕ, ψ) conserves (weak) termination, then the same holds for $(\bar{\phi}, \psi)$, simply because this property does not depend on ϕ, but on ψ. □

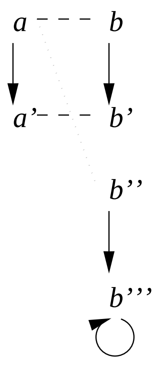
In [10], the reachability restriction is added to the definition of completeness for simulations. However, the rationale of Lemma 4.11 is that the notion of reachability needs no elaboration in the theory of simulations.

The converse of Lemma 4.11 does not hold. Namely, there exist simulations (ϕ, ψ) which are not sound or not complete or which do not (totally) conserve (weak) termination, but which do have this property if ϕ is restricted to the reachable part of B. We give an example.

Example 4.12 Let $A = \{a, a'\}$ and aRa'. Furthermore, let $B = \{b, b', b'', b'''\}$ and bSb' and $b''Sb'''$ and $b'''Sb'''$. Define a simulation (ϕ, ψ) as follows:

$$\begin{array}{ll} \phi(b) = a & \psi(a) = b \\ \phi(b') = a' & \psi(a') = b' \\ \phi(b'') = a & \end{array}$$

This simulation can be depicted as follows:



This simulation is not sound nor complete nor does it totally conserve termination. However, if ϕ is restricted to the reachable part of B, which consists of $\{b, b'\}$, then the simulation becomes sound and complete and totally conserves termination.

4.5 Simulation Violates Confluence

The following example shows that there exist simulations which are sound and complete and which totally conserve termination, but which do not conserve confluence.

Example 4.13 Let $A = \{a\}$ and $B = \{b, b', b''\}$ and bSb' and bSb''. Define a simulation (ϕ, ψ) as follows:

$$\begin{array}{ll} \phi(b) = a & \psi(a) = b \\ \phi(b') = a & \\ \phi(b'') = a & \end{array}$$

This simulation can be depicted as follows:

$$\begin{tikzcd}
a \arrow[rrr, dash, dashed] \arrow[drr, dash, dotted] \arrow[drrrr, dash, dotted, bend right] & & & b \arrow[dl] \arrow[dr] & \\
& & b' & & b''
\end{tikzcd}$$

This simulation sound and complete and totally conserves termination. However, (A, R) is confluent, while B is not confluent for $\psi(a) = b$.

4.6 Correctness Criteria for Simulation

In this section we study under which conditions a simulation is (weakly) correct. First, we present two lemmas which indicate when $\phi(nf_S(\psi(a)))$ is a subset of $nf_R(a)$, and vice versa.

Lemma 4.14 *If a simulation is sound and weakly complete, then* $\phi(nf_S(\psi(a))) \subseteq nf_R(a)$.

Proof. Let $b \in nf_S(\psi(a))$; we show that $\phi(b)$ is defined and $\phi(b) \in nf_R(a)$.

Since $\psi(a)S^*b$, soundness implies that there exists a $b' \in B$ with bS^*b' and $\phi(b')$ defined and $aR^*\phi(b')$. Since b is a normal form for S, and bS^*b', it follows that $b = b'$. Hence, $\phi(b)$ is defined and $aR^*\phi(b)$.

Furthermore, since b is a normal form for S, weak completeness says that $\phi(b)$ is a normal form for R.

Since $aR^*\phi(b)$ and $\phi(b)$ is a normal form for R, it follows that $\phi(b) \in nf_R(a)$. $\Box$

Lemma 4.15 *If a simulation is sound and complete and totally conserves weak termination, then* $nf_R(a) \subseteq \phi(nf_S(\psi(a)))$.

Proof. Let $a' \in nf_R(a)$; we show that $a' \in \phi(nf_S(\psi(a)))$.

Since aR^*a', completeness yields that there exists a $b \in B$ such that $\psi(a)S^*b$ and $\phi(b)$ is defined and $\phi(b) = a'$. Since a' is a normal form for R, total conservation of weak termination yields that S is weakly terminating for a'. Hence, there exists a $b' \in B$ which is a normal form for S such that bS^*b'. Since $\psi(a)S^*bS^*b'$, it follows that $b' \in nf_S(\psi(a))$.

Since bS^*b' and $\phi(b) = a'$, soundness says that there exists a $b'' \in B$ such that $b'S^*b''$ and $\phi(b'')$ is defined and $a'R^*\phi(b'')$. Since b' is a normal form for S and $b'S^*b''$, it follows that $b' = b''$, so $a'R^*\phi(b')$. Since a' is a normal form for R, it follows that $\phi(b') = a'$. Hence, $a' \in \phi(nf_S(\psi(a)))$. $\Box$

Now we are ready to prove under which conditions simulation is a (weakly) correct transformation.

Theorem 4.16 *If a simulation is sound and complete and conserves termination and totally conserves weak termination, then it is a correct transformation.*

Proof. Choose *parse* to be ψ, and *print* to be any total extension of ϕ. We show that these mappings satisfy the requirements of a correct transformation.

If R is terminating for $a \in A$, then conservation of termination ensures that S is terminating for $\psi(a)$.

According to Proposition 4.5 completeness induces weak completeness, so Lemma 4.14 implies that $\phi(nf_S(\psi(a))) \subseteq nf_R(a)$.

Finally, according to Lemma 4.15, soundness and completeness and total conservation of weak termination yield $nf_R(a) \subseteq \phi(nf_S(\psi(a)))$. □

Theorem 4.17 *If a simulation is sound and weakly complete and conserves both termination and weak termination, then it is a weakly correct transformation.*

Proof. Choose *parse* to be ψ, and *print* to be any total extension of ϕ. We show that these mappings satisfy the requirements of a weakly correct transformation.

If R is terminating for some $a \in A$, then conservation of termination ensures that S is terminating for $\psi(a)$.

Furthermore, according to Lemma 4.14, soundness together with weak completeness implies that $\phi(nf_S(\psi(a))) \subseteq nf_R(a)$.

Finally, if R is weakly terminating for some $a \in A$, then conservation of weak termination ensures that S is weakly terminating for $\psi(a)$. □

In Example 3.5, the transformed rewrite system S is terminating, so clearly the simulation in that example totally conserves termination. Earlier, we noted that this simulation is sound and complete. Then, by Proposition 4.8, it conserves termination and totally conserves weak termination. So according to Theorem 4.16 it is a correct transformation.

Kamperman and Walters [9, 10] consider several practical examples of simulation, and show that they are sound and complete and totally conserve termination. According to Theorem 4.16, in order for a simulation to be a correct transformation, the last requirement can be weakened to conservation of termination together with total conservation of weak termination. According to Proposition 4.8, these two requirements indeed follow from total conservation of termination.

We note that the reverse does not hold, namely, there exist sound and complete simulations which conserve termination and which totally conserve weak termination, but which do not totally conserve termination. This is shown in the following example.

Example 4.18 Let $A = \{a, a'\}$ and aRa and aRa'. Furthermore, let $B = \{b, b', b'', b'''\}$ and bSb and bSb' and bSb'' and $b''Sb''$ and $b''Sb'''$. Define a simulation (ϕ, ψ) as follows:

$$\begin{array}{ll} \phi(b) = a & \psi(a) = b \\ \phi(b') = a' & \psi(a') = b' \\ \phi(b'') = a' & \\ \phi(b''') = a' & \end{array}$$

This simulation can be depicted as follows:

$$\begin{tikzcd}
a \arrow[loop left] \arrow[r, dash, dashed] \arrow[d] & b \arrow[loop right] \arrow[d] \arrow[dr] & \\
a' \arrow[r, dash, dashed] \arrow[rr, dash, dotted, bend right] \arrow[drr, dash, dotted] & b' & b'' \arrow[loop right] \arrow[d] \\
& & b'''
\end{tikzcd}$$

This simulation is sound and complete. Furthermore, it conserves termination: R is only terminating for a', and S is terminating for $\psi(a') = b'$. Also, it totally conserves weak termination, because (B, S) is weakly terminating. However, this simulation does not totally conserve termination: R is terminating for a', but S is not terminating for $b'' \in \phi^{-1}(a')$.

5 Further Examples of Transformations

In this section we consider other transformations of rewrite systems that have been proposed in the literature. In order to avoid extensive technical expositions, the foundations for our correctness claims will not be described in detail.

Kamperman and Van de Pol [8] show how a weakly terminating rewrite system together with a normalization strategy can be turned into a terminating rewrite system. They prove that their transformation is a simulation that is sound and complete and totally conserves termination. So according to results obtained in this paper it is a correct transformation.

Zantema [23] invented the technique of semantic labelling, where semantics is provided to the function symbols, and for each choice of semantic labels for the operators in the left-hand side of a rewrite rule, a new rewrite rule is introduced. Zantema proves that the original rewrite system is terminating if and only if its labelled transformation is so. Sometimes, proving termination of the labelled rewrite system is much easier than proving termination for the original rewrite system, see for example [7]. Semantic labelling preserves the structure of reduction trees, so it constitutes a correct transformation.

Graph rewriting is an implementation strategy for term rewriting systems where variables are shared. It is well-known that the transformation of term rewriting into graph rewriting can lead to undesirable complications. For example, the term rewriting system

$$\begin{array}{rcl} f(a,b) & \longrightarrow & a \\ f(x,x) & \longrightarrow & f(x,x) \\ a & \longrightarrow & b \end{array}$$

is not adequately simulated by its corresponding graph rewriting system. In the term rewriting system the normalizing reduction $f(a,a) \longrightarrow f(a,b) \longrightarrow a$ is possible. However, in graph rewriting due to the sharing of variables, only the reductions $f(a,a) \longrightarrow f(a,a)$ and $f(a,a) \longrightarrow f(b,b)$ are possible, so that $f(a,a)$ does not have a normal form. In [2], soundness and completeness of the transformation of term rewriting into graph rewriting is studied. The conclusion is that for left-linear weakly non-overlapping term rewriting systems, the transformation into graph rewriting systems preserves normal forms. In order to obtain correctness, one additionally has to verify that termination is preserved, which is indeed the case.

Laville [12] considered rewrite systems with priorities, which were first studied in [1]. If two rewrite rules can be applied to the same term, then only the rule with the highest priority is applied. Priorities are a powerful means to capture intricate rewriting in a simple rewrite system, but they are troublesome when it comes to implementation. Laville shows how a rewrite system with priorities can be transformed into a rewrite system where the priorities are captured in the syntax; this method has been applied in the implementation of the CAML system. Laville's transformation leaves the structure of reduction trees in tact, so clearly it is correct.

Thatte [17] showed how a left-linear non-overlapping rewrite system R can be transformed into a left-linear non-overlapping rewrite system S that is constructor-based, where R ranges over a signature Σ and S ranges over an extended signature $\overline{\Sigma}$. Thatte showed that the original and the transformed rewrite system are equivalent, in the sense that there is a mapping $\phi : \overline{\Sigma} \to \Sigma$, which is the identity on Σ, such that:

- if bSb' then $\phi(b)R^*\phi(b')$;
- if aRa' then aS^*a'.

The first property ensures that as a simulation this transformation is sound. However, completeness and conservation of (weak) termination cannot be concluded from these properties. Therefore, this equivalence notion does not imply that the transformation is correct.

In [18], Thatte claimed for several more general notions of rewrite systems that his transformation preserves confluence and normal forms in the reachable part of the transformed system. However, Verma [20, 19] showed that two of these claims are erroneous. Namely, Thatte's transformation does not preserve confluence nor normal forms for confluent non-overlapping rewrite systems, but only for confluent weakly persistent rewrite systems, see [20]. Moreover, Thatte's transformation does not preserve confluence. Therefore, Verma [19] introduced a new transformation for confluent terminating rewrite systems, and showed that his transformation does preserve both confluence and normal forms. We remark that transformations of terminating rewrite systems that preserve normal forms are correct.

Sekar et al. [16] showed how a strongly sequential constructor-based rewrite system can be transformed into an 'equivalent' path sequential rewrite system. Their notion of equivalence is similar to the one of Thatte in [17], so it cannot be concluded from their equivalence notion that the transformation is correct.

In [6, 14], transformations of equational specifications of abstract data types are studied. Such a transformation is called a 'correct implementation' if the initial algebras of the original and the transformed specification are isomorphic. This notion is considerably stronger than our notion of a correct transformation. For example, the ARS $(\{a\}, \emptyset)$ transforms correctly into $(\{b, b'\}, \emptyset)$ by means of $\mathit{parse}(a) = b$ and $\mathit{print}(b) = a$ and $\mathit{print}(b') = a$. However, when considered as specifications of algebraic data types, $(\{b, b'\}, \emptyset)$ is not a correct implementation of $(\{a\}, \emptyset)$.

References

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

[2] H.P. Barendregt, M.C.J.D van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, and M.R. Sleep. Term graph rewriting. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings 1st Conference on Parallel Architectures and Languages Europe (PARLE'87),* Eindhoven, *Vol. II: Parallel Languages*, LNCS 259, pages 141–158. Springer, 1987.

[3] H.P. Barendregt, H. Wupper, and H. Mulder. Computable processes. Report CS-R9428, CWI, Amsterdam, 1994.

[4] J.A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification.* ACM Press in cooperation with Addison Wesley, 1989.

[5] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B, Formal Methods and Semantics*, pages 243–320. Elsevier, 1990.

[6] H. Ehrig, H.-J. Kreowski, and P. Padawitz. Stepwise specification and implementation of abstract data types. In G. Ausiello and C. Böhm, editors, *Proceedings 5th Colloquium on Automata, Languages and Programming (ICALP'78),* Udine, LNCS 62, pages 205–226. Springer, 1978.

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

[8] J.F.Th. Kamperman and J.C. van de Pol. Simulation of strategies in term rewriting. In preparation, 1996.

[9] J.F.Th. Kamperman and H.R. Walters. Minimal term rewriting systems. In *Proceedings 11th Workshop on Abstract Data Types*, LNCS, to appear. Available at http://www.cwi.nl/epic.

[10] J.F.Th. Kamperman and H.R. Walters. Simulating TRSs by minimal TRSs: a simple, efficient, and correct compilation technique. Technical Report CS-R9605, CWI, 1996. Available at http://www.cwi.nl/epic.

[11] 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.

[12] A. Laville. Lazy pattern matching in the ML language. In K.V. Nori, editor, *Proceedings 7th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'87),* Pune, India, LNCS 287, pages 400–419. Springer, 1987.

[13] B. Luttik. Simulation for rewrite systems. In preparation.

[14] C.F. Nourani. Abstract implementations and their correctness proofs. *Journal of the ACM*, 30:343–359, 1983.

[15] M.J. O'Donnell. *Equational Logic as a Programming Language.* MIT Press, Cambridge, 1985.

[16] R.C. Sekar, S. Pawagi, and I.V. Ramakrishnan. Transforming strongly sequential rewrite systems with constructors for efficient parallel execution. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications (RTA'89),* Chapel Hill, LNCS 355, pages 404–418. Springer, 1989.

[17] S.R. Thatte. On the correspondence between two classes of reduction systems. *Information Processing Letters*, 20(2):83–85, 1985.

[18] S.R. Thatte. Implementing first-order rewriting with constructor systems. *Theoretical Computer Science*, 61(1):83–92, 1988.

[19] R.M. Verma. A theory of using history for equational systems with applications. *Journal of the ACM*, 42(5):984–1020, 1995.

[20] R.M. Verma. Transformations and confluence for rewrite systems. *Theoretical Computer Science*, 152(2):269–283, 1995.

[21] H.R. Walters and J.F.Th. Kamperman. EPIC: an equational language – abstract machine and supporting tools. In *Proceedings 7th Conference on Rewriting Techniques and and Applications (RTA'96)*, LNCS, to appear.

[22] H.R. Walters and J.F.Th. Kamperman. Epic 1.0 (unconditional), an equational programming language. Report CS-R9604, CWI, Amsterdam, 1996. Available at http://www.cwi.nl/epic.

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