Simulation as a Correct Transformation of Rewrite Systems

Wan Fokkink
University of Wales Swansea
Department of Computer Science
Singleton Park, Swansea SA2 8PP, Wales
e-mail: w.j.fokkink@swan.ac.uk

Jaco van de Pol
Eindhoven University of Technology
Department of Computer Science
PO Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: jaco@win.tue.nl

Abstract. Kamperman and Walters proposed the notion of a *simulation* of one rewrite system by another one, whereby each term of the simulating rewrite system is related to a term in the original rewrite system. In this paper it is shown that if such a simulation is *sound* and *complete* and *preserves termination*, then the transformation of the original into the simulating rewrite system constitutes a correct step in the compilation of the original rewrite system. That is, the normal forms of a term in the original rewrite system can then be obtained by computing the normal forms of a related term in the simulating rewrite system.

1 Introduction

Questions on the correctness of compilation of programming languages date back to McCarthy [12]. In this paper we present a technique to deduce the correctness of compilation steps for functional programming languages which stay inside the domain of rewrite systems.

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, 10, 18, 16, 19, 22, 7]. 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 rewrite system can somehow simulate the original rewrite system, so that the reduction graph of an original and a simulating term have the same structure.

Recently, Kamperman and Walters [7, 8, 6] proposed a notion of simulation of one rewrite system by another rewrite system. A simulation basically consists of a surjective mapping ϕ which relates each term in the simulating rewrite system

to a term in the original rewrite system. A simulation should be *sound*, meaning that if a term t can be rewritten to a term t' in one step in the simulating rewrite system, then $\phi(t)$ can be rewritten to $\phi(t')$ in zero or more steps in the original rewrite system. Furthermore, a simulation should be *complete*, meaning that if a term $\phi(t)$ can be rewritten to a term s in one step in the original rewrite system, then t can be rewritten to t' with $\phi(t') = s$ in one or more steps in the simulating rewrite system. Finally, a simulation should *preserve termination*, meaning that if the original rewrite system is terminating for a term $\phi(t)$, then the simulating rewrite system should be terminating for t. (The other way around is guaranteed by completeness.)

Kamperman and Walters apply simulation to transform a left-linear rewrite system into a form which is more suitable for compilation, as a first step in the implementation of their equational programming language EPIC [20, 21]. Kamperman and Walters state, for example in the title of [8], that if a simulation is sound and complete and preserves termination, then it constitutes a correct transformation of rewrite systems. However, they do not yet provide a foundation for this claim. 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 one is not. Hence, the question arises what it means to state that such a transformation of rewrite systems is 'correct'.

Although preservation of reduction graphs underlies simulation, reduction graphs are usually not of interest in applications of rewrite 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 rewrite system is transformed as part of a compilation project, then the main interest is that the transformation preserves normal forms. We propose the notion of a correct transformation of rewrite systems, based on ideas on compiler correctness by Burstall and Landin [2] and Morris [13]. We say that the transformation of one rewrite system into another is correct if no information on normal forms in the original rewrite system is lost. That is, it should be possible to provide mappings *parse* from original to transformed terms and *print* from transformed to original terms such that for each original term t its normal forms can be computed as follows: compute the normal forms of $\mathit{parse}(t)$, and apply the *print* function to them. In order to make sure that the simulating rewrite system returns an answer whenever the original rewrite system does, it is required that a correct transformation also preserves termination properties.

We show that the notion of a simulation as proposed in [7, 8, 6] constitutes a correct transformation, under the conditions that it is sound and complete and preserves termination. Hence, the notion of simulation constitutes a useful tool for proving correctness of compilation of rewrite systems. Namely, such a compilation may involve a chain of transformations of rewrite systems. In order to prove correctness of one such transformation, it is sufficient to find a simulation relation that is sound, complete and termination preserving. The intuitive link

between the two rewrite systems before and after a transformation can often be materialized in an explicit simulation relation. Simulation and its properties soundness, completeness and termination preservation are all conserved under composition, so that it suffices to determine these properties for each consecutive step of a transformation chain.

We will generalize and simplify existing simulation definitions considerably. The proof of the correctness of simulation will use almost in full the criteria for soundness and completeness and preservation of termination. One could therefore argue that these criteria were designed to satisfy the requirements of a correct transformation implicitly.

In practical cases, a simulation is often not immediately sound, complete and termination preserving, due to the fact that the simulating rewrite system contains 'junk'. In such cases, a reachability restriction on the elements in the simulating rewrite system can help to make the simulation sound, complete and termination preserving. We will formalize this reachability notion.

Related Work. In [4, 14], a transformation of an equational specification of abstract data types 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.

Thatte [17, 18] defined a transformation of certain types of rewrite systems, over a signature Σ, into rewrite systems that are constructor based, over an extended signature Σ^*. The relation between the original and the transformed rewrite system, with the latter restricted to the part that is reachable from Σ, is given by a mapping $\phi : \Sigma^* \to \Sigma$, which is the identity on Σ. Since this mapping is surjective, it is a simulation. Thatte shows that this simulation is sound, and satisfies a weaker completeness notion: if a term s can be rewritten to a term s' in one step in the original rewrite system, then s can be rewritten to s' in zero or more steps in the simulating rewrite system. This weaker completeness notion (which was also used by Sekar et al. [16]) does not imply that a transformation is correct.

Luttik [11] proposed a series of stronger simulation notions, and shows that they preserve termination and confluence.

In the technical report version of this paper [5], more information is provided on so-called 'weak correctness' of transformations, which basically means that at least one normal form of each term in the original rewrite system is conserved by the transformation.

Acknowledgements. Jasper Kamperman, Bas Luttik, Karen Stephenson and Pum Walters are thanked for useful comments, and Jan Bergstra for his support. A considerable part of this research was carried out when both authors worked at the Philosophy Department of Utrecht University.

2 Abstract Reduction Systems

This section introduces some preliminaries from rewriting [3, 9].

Definition 1. An *abstract reduction system (ARS)* consists of a collection A of elements, together with a binary reduction relation R between elements in A.

R^+ denotes the transitive closure of a reduction relation R, and R^* the reflexive transitive closure of R. In the following definitions, we assume an ARS (A, R).

Definition 2. $a \in A$ is a *normal form* for R if there does not exist an $a' \in A$ with aRa'. $a \in A$ is a *normal form of* $a' \in A$ if $a'R^*a$ and a is a normal form.

$nf_R : A \to \mathcal{P}(A)$ maps each $a \in A$ to its collection of normal forms for R.

Definition 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.)

3 Correctness of Transformations

We formulate general conditions which ensure that a transformation of rewrite systems is correct. We adopt the point of view that such a transformation is correct if it constitutes a sensible step in a compilation procedure. This is the case if the input/output behaviour of the original rewrite system is maintained, where the input is any term, and the output is (one of) its normal form(s). Hence, for compilation of rewrite systems, the prime interest of a transformation is that it preserves 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. Justifications of this claim abound in the literature:

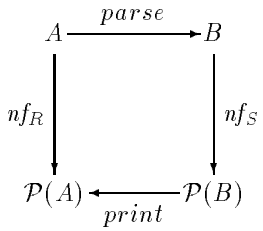
- equational theorem proving is mostly concerned with terminating rewrite systems which yield unique normal forms [15];
- if rewriting is applied to implement abstract data types, then the meaning of a term is fixed by its normal forms [1];
- in [3] it is remarked that "rewrite systems defining at most one normal form for any input term can serve as functional programs".

As explained in the introduction, we propose a notion of correctness of transformations, which requires parse and print functions that allow the normal forms of the original rewrite system to be computed via the simulating rewrite system.

In the following definition, 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 4. 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. $print(nf_S(parse(a))) = nf_R(a)$ for $a \in A$, that is, the diagram below commutes:



For this notion of a correct transformation, the definition of the *print* function outside of $nf_S(parse(A))$, and the S-relation outside the range of *parse*, are irrelevant.

4 Correctness of Simulation

4.1 Simulation

Kamperman and Walters [7, 8, 6] propose a notion of simulation for rewrite systems. We present simplified and more general versions of their definitions in the next sections. A simulation of an ARS (A, R) by an ARS (B, S) is characterized by a surjective mapping $\phi : B \to A$. The intuition for this mapping ϕ is that the reduction graph of $a \in A$ with respect to R is simulated by the reduction graphs of all $b \in \phi^{-1}(a)$ with respect to S.

Definition 5. A *simulation* of an ARS (A, R) by an ARS (B, S) is a surjective mapping $\phi : B \to A$.

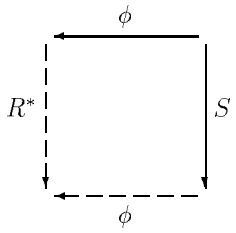
Simulation is a transitive relation, simply because the composition of two surjective mappings is again surjective.

4.2 Soundness

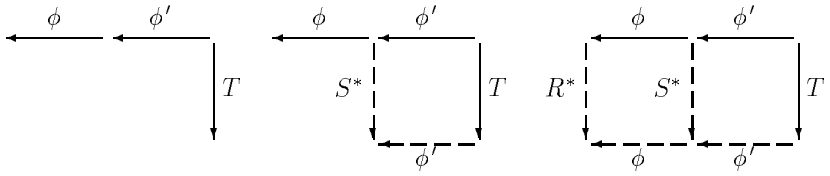
In this and the following sections we assume as general notation that the ARS (A, R) is simulated by the ARS (B, S) by means of the surjective mapping $\phi : B \to A$. Soundness of a simulation ensures that each S-step of $b \in B$ is a simulation of some finite R-reduction of $\phi(b)$.

Definition 6. The simulation ϕ is *sound* if for each $b, b' \in B$ with bSb' we have $\phi(b)R^*\phi(b')$.

Soundness can be depicted as follows:



Let ϕ be a simulation of (A, R) by (B, S), and ϕ' a simulation of (B, S) by (C, T). If both ϕ and ϕ' are sound, then their composition $\phi \circ \phi'$ is also sound, which can be seen from the following graphical outline.



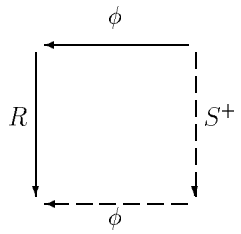
The first step is due to soundness of ϕ', and the second to soundness of ϕ, applied to each step in the S^*-reduction.

4.3 Completeness

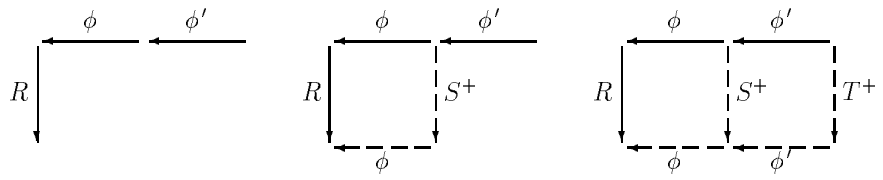
As opposed to soundness, completeness ensures that each R-step from $\phi(b)$ is simulated by a finite S-reduction of b with length greater than zero. Note that, due to the surjectivity of ϕ, each $a \in A$ can be written as $\phi(b)$ for some b.

Definition 7. The simulation ϕ is *complete* if for each $a \in A$ and $b \in B$ with $\phi(b)Ra$ there is a $b' \in B$ with bS^+b' and $\phi(b') = a$.

Completeness can be depicted as follows:



Let ϕ be a simulation of (A, R) by (B, S), and ϕ' a simulation of (B, S) by (C, T). If both ϕ and ϕ' are complete, then their composition $\phi \circ \phi'$ is also complete, which can be seen from the following graphical outline.



The first step is due to completeness of ϕ, and the second to completeness of ϕ', applied to each step in the S^+-reduction.

4.4 Termination Preservation

Termination preservation of a simulation ensures that termination properties for the original rewrite system are preserved by the simulating rewrite system.

Definition 8. A simulation *preserves termination* if for each $a \in A$ for which R is terminating, S is terminating for each $b \in \phi^{-1}(a)$.

Let ϕ be a simulation of (A, R) by (B, S), and ϕ' a simulation of (B, S) by (C, T). If both ϕ and ϕ' preserve termination, then their composition $\phi \circ \phi'$ also preserves termination. Namely, suppose that R is terminating for $a \in A$, and let $\phi \circ \phi'(c) = a$. Then termination preservation of ϕ yields that S is terminating for $\phi'(c)$, and termination preservation of ϕ' yields that T is terminating for c.

4.5 Correctness

Theorem 9. *If a simulation is sound and complete and preserves termination, then it is a correct transformation.*

Proof. Let ϕ be a simulation of (A, R) by (B, S) which is sound and complete and preserves termination. Choose *print* to be ϕ, and let *parse* be any inverse of *print* (i.e., $parse(a) \in \phi^{-1}(a)$ for each $a \in A$). We show that these mappings satisfy the requirements of a correct transformation.

First we prove $print(nf_S(parse(a))) \subseteq nf_R(a)$ for $a \in A$. Let $b \in nf_S(parse(a))$; we have to show that $print(b) \in nf_R(a)$. Since $parse(a)S^*b$, soundness yields $aR^*print(b)$. Since b is a normal form for S, completeness implies that $print(b)$ is a normal form for R. So, since $aR^*print(b)$, it follows that $print(b) \in nf_R(a)$.

Next we prove $nf_R(a) \subseteq print(nf_S(parse(a)))$ for $a \in A$. Let $a' \in nf_R(a)$; we show that $a' \in print(nf_S(parse(a)))$. Since aR^*a', completeness yields that there exists a $b \in B$ such that $parse(a)S^*b$ and $print(b) = a'$. Since a' is a normal form for R, preservation of termination yields that S is terminating for b. Hence, there exists a $b' \in B$ which is a normal form for S such that bS^*b'. Since $parse(a)S^*bS^*b'$, it follows that $b' \in nf_S(parse(a))$. Since bS^*b' and $print(b) = a'$, soundness yields $a'R^*print(b')$. The fact that a' is a normal form for R then implies $print(b') = a'$. Hence, $a' \in print(nf_S(parse(a)))$.

Finally, if R is terminating for $a \in A$, then preservation of termination ensures that S is terminating for $parse(a)$. □

4.6 Reachability

Definition 10. Let (B, S) be an ARS, and $B_0 \subseteq B$. An element $b \in B$ is *S-reachable from B_0* if $b_0 S^* b$ for some $b_0 \in B_0$.

This reachability notion also occurs in [18, 19].

In practical cases, a simulation $\phi : B \to A$ of an ARS (A, R) by an ARS (B, S) is not always immediately sound, complete and termination preserving, due to the fact that B contains 'junk'. In such cases, it may be possible to define a collection $B_0 \subseteq B$ that is closed under S-reductions, such that ϕ restricted to B_0 is still surjective, and a sound, complete, termination preserving simulation. A particular case arises when A itself is a subset of B. In that case, restricting ϕ to the S-reachable terms of A often does the job.

We note that such a restriction of a simulation never loses any desirable properties. That is, let $\phi : B \to A$ be a simulation of (A, R) by (B, S) that is sound or complete or termination preserving. If the restriction of ϕ to a collection $B_0 \subseteq B$, closed under S-reductions, is still surjective, then this restricted simulation is still sound or complete or termination preserving, respectively.

5 An Example

We present a toy example of a transformation of a term rewriting system (TRS) [3, 9], which will be shown to be a sound, complete and termination preserving simulation, if a reachability restriction is imposed on the ARS induced by the transformed TRS.

Assume two constants c and c', and two unary functions f and g, and let A be the set of closed terms over the signature $\{c, c', f, g\}$. Consider the TRS R that consists of the following rewrite rule:

$$f(g(c)) \longrightarrow c'$$

This rewrite rule induces an ARS on A, also denoted by R, in the standard way. That is, a relation aRa' holds if a' can be obtained from a by replacing a subterm $f(g(c))$ of a by c'.

In order to compile (A, R), the TRS R can be transformed into a so-called 'minimal' TRS, using a strategy from [7, 8, 6]. In minimal TRSs, rewrite rules are not allowed to contain more than three function symbols, and no more than two function symbols at each side of the rewrite rule. These restrictions are convenient for compilation of the TRS, because matchings with respect to a minimal rewrite system can easily be expressed in basic instructions of an abstract machine. Note that the rewrite rule in R is not minimal. The minimization strategy transforms it into the following minimal TRS M:

$$\begin{aligned} f(g(x)) &\longrightarrow f_g(x) \\ f(x) &\longrightarrow f^d(x) \\ f_g(c) &\longrightarrow c' \\ f_g(x) &\longrightarrow f^d(g(x)) \end{aligned}$$

These rewrite rules generate an ARS, also denoted by M, which applies to the collection B of closed terms over the signature $\{c, c', f, f^d, f_g, g\}$. A relation bMb' holds if b' can be obtained from b by replacing a subterm b_0 of b, which is an instance of the left-hand side of a rewrite rule r in M, by the corresponding instance of the right-hand side of r. The rewriting strategy is *innermost*, which requires that the proper subterms of b_0 cannot be replaced. Furthermore, rewrite rules are applied with respect to *specificity ordering*, which requires that the left-hand side of r is the most specific of all left-hand sides of rules in M that match with b_0. Note that the left-hand side of the first and third rewrite rule in M are more specific than the left-hand side of the second and fourth rule, respectively.

B contains two new unary function symbols, f^d and f_g. The intuition behind the transformation, and the two new function symbols, is as follows. Basically, the transformation splits possible matchings with respect to the left-hand side of the rule in R into elementary steps. A term $f_g(b)$ is an encoded representation of the term $f(g(b))$; this is captured by the first rewrite rule of M. Furthermore, a term $f^d(b)$ is an encoded representation of the term $f(b)$, and expresses that this term cannot be rewritten by the rule in R. Therefore, the second rule of M reduces $f(b)$ to $f^d(b)$, if b is not of the form $g(b')$ (due to the fact that the first rule has priority over the second rule). The third rule mimics the rule in R, because $f_g(c)$ represents $f(g(c))$. Finally, the fourth rule of M expresses that a term $f(g(b))$ with $b \neq c$ cannot be rewritten by R, so that in M a term $f_g(b)$ is reduced to $f^d(g(t))$ if $b \neq c$ (due to the fact that the third rule has priority over the fourth rule).

In spite of the intuition presented above, it remains a natural question to ask whether the transformation of R into M is correct. In order to give a positive answer to this question, we capture this intuition by a simulation relation as follows. Let B_0 the set of elements in B that are M-reachable from A. Note that $A \subseteq B_0$. We define a surjective mapping $\phi : B_0 \rightarrow A$ inductively as follows:

$$\begin{array}{ll} \phi(c) = c & \phi(f(t)) = f(\phi(t)) \\ \phi(c') = c' & \phi(f^d(t)) = f(\phi(t)) \\ \phi(g(t)) = g(\phi(t)) & \phi(f_g(t)) = f(g(\phi(t))) \end{array}$$

This simulation is sound and complete and preserves termination. Hence, according to Theorem 9, the transformation of (A, R) into (B_0, M) is correct.

Completeness depends on the reachability restriction that was imposed on B_0. For example, $f^d(g(c))$ is a normal form for M, while $\phi(f^d(g(c))) = f(g(c))$ is not a normal form for R. However, $f^d(g(c))$ is not in B_0, because it is not M-reachable from A.

References

1. J.A. Bergstra, J. Heering, and P. Klint, eds. *Algebraic Specification*. ACM Press in cooperation with Addison Wesley, 1989.
2. R.M. Burstall and P.J. Landin. Programs and their proofs: an algebraic approach. In *Machine Intelligence*, Volume 4, pp. 17–43. Edinburgh University Press, 1969.

3. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Volume B*, pp. 243–320. Elsevier, 1990.
4. H. Ehrig, H.-J. Kreowski, and P. Padawitz. Stepwise specification and implementation of abstract data types. In *Proceedings ICALP'78*, LNCS 62, pp. 205–226. Springer, 1978.
5. W.J. Fokkink and J.C. van de Pol. Correct transformation of rewrite systems for implementation purposes. Logic Group Preprint Series 164, Utrecht University, 1996. Available at http://www.phil.ruu.nl.
6. J.F.Th. Kamperman. *Compilation of Term Rewriting Systems*. PhD thesis, University of Amsterdam, 1996.
7. J.F.Th. Kamperman and H.R. Walters. Minimal term rewriting systems. In *Proceedings WADT'95*, LNCS 1130, pp. 274–290. Springer, 1996.
8. J.F.Th. Kamperman and H.R. Walters. Simulating TRSs by minimal TRSs: a simple, efficient, and correct compilation technique. Report CS-R9605, CWI, 1996. Available at http://www.cwi.nl/epic.
9. J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Volume I*, pp. 1–116. Oxford University Press, 1992.
10. A. Laville. Lazy pattern matching in the ML language. In *Proceedings FSTTCS'87*, LNCS 287, pp. 400–419. Springer, 1987.
11. B. Luttik. Transformation of reduction systems: a view on proving correctness. Master's Thesis, University of Amsterdam, 1996.
12. J. McCarthy. Towards a mathematical science of computation. In *Proceedings Information Processing '62*, pp. 21–28. North-Holland, 1963.
13. F.L. Morris. Advice on structuring compilers and proving them correct. In *Proceedings POPL'73*, pp. 144–152. ACM Press, 1973.
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, 1985.
16. R.C. Sekar, S. Pawagi, and I.V. Ramakrishnan. Transforming strongly sequential rewrite systems with constructors for efficient parallel execution. In *Proceedings RTA'89*, LNCS 355, pp. 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. Transformations and confluence for rewrite systems. *Theoretical Computer Science*, 152(2):269–283, 1995.
20. H.R. Walters and J.F.Th. Kamperman. EPIC: an equational language – abstract machine and supporting tools. In *Proceedings RTA'96*, LNCS 1103, pp. 424–427. Springer, 1996.
21. H.R. Walters and J.F.Th. Kamperman. EPIC 1.0 (unconditional), an equational programming language. Report CS-R9604, CWI, 1996. Available at http://www.cwi.nl/epic.
22. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1,2):89–105, 1995.

This article was processed using the LaTeX macro package with LLNCS style