Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 204 (2008) 35–51
www.elsevier.com/locate/entcs

Termination of Lazy Rewriting Revisited

Felix Schernhammer and Bernhard Gramlich[1]

Theory and Logic Group
Institute of Computer Languages
TU Wien, Austria

Abstract

Lazy rewriting is a proper restriction of term rewriting that dynamically restricts the reduction of certain arguments of functions in order to obtain termination. In contrast to context-sensitive rewriting, reductions at such argument positions are not completely forbidden but delayed. Based on the observation that the only existing (non-trivial) approach to prove termination of such lazy rewrite systems is flawed, we develop a modified approach for transforming lazy rewrite systems into context-sensitive ones that is sound and complete with respect to termination. First experimental results with this transformation based technique are encouraging.

Keywords: term rewriting, lazy rewriting, termination, context-sensitive system

1 Introduction

In functional programming languages, evaluations are often carried out in a lazy fashion. This means that in the evaluation of an expression, the result of certain subexpressions is not computed until it is known that the particular result is actually needed. A very similar idea is used in lazy rewriting ([3]) where the reduction of certain subterms is delayed as long as possible (cf. also [1], [8], and [10]).

Termination analysis of functional programs has recently become a major subject of research in the rewriting community. Due to the similarity of lazy rewriting and the lazy evaluation strategy of functional programs, the use of lazy rewriting seems promising to find new methods for proving termination of functional programs. In particular lazy rewriting and lazy evaluation in functional programming share the idea of postponing certain evaluation steps. More precisely, arguments of functions are only evaluated if their final result is needed to compute the function.

Example 1.1 *Consider the following functional program given in term rewriting system (TRS) syntax.*

[1] Email: {felixs,gramlich}@logic.at

doi:10.1016/j.entcs.2008.03.052

$$from(x) \to x : from(s(x)) \qquad take(0, xs) \to []$$
$$take(x, []) \to [] \qquad take(s(x), y : ys) \to y : take(x, ys)$$

Without an evaluation strategy the input term $take(s(0), from(0))$ *is non-terminating. Yet, when using a lazy evaluation strategy it is terminating and the result is* 0. *The crucial difference is that in a term of the shape* $take(\_, \_ : from(\_))$ *the from subterm may not be evaluated under lazy evaluation, because its result is not needed to evaluate any more outer function.*

However, a reduction of the problem of proving termination of functional programs to the problem of proving termination of lazy TRSs is non-trivial. The reason is that functional programming languages typically allow for features such as higher-order functions, typing and strategies other than lazy evaluation; e.g., in *Haskell* always the "first" defining equation is applied to an object term when several equations are applicable (at the same position) (this has already been pointed out in [6]).

Therefore, lazy rewriting can only approximate lazy (Haskell) evaluations but clearly not simulate them in a one-to-one fashion. Yet, the described features of functional programs can be encoded or approximated through standard transformations such that finally lazy rewriting can be used for a termination analysis (cf. e.g. [7]).

In this work we present a transformation from lazy TRSs into context-sensitive ones that is sound and complete w.r.t. termination and thus allows us to reduce the problem of deciding whether a lazy TRS is terminating or not to the same problem for context-sensitive TRSs, which has already been studied to some extent (cf. eg. [12]). As this is the first sound and complete transformation for lazy TRSs, it enables us for the first time to investigate the use of lazy rewriting in the area of termination analysis of functional programs.

Lazy rewriting was initially introduced by [3] in a graph rewriting setting (although the basic underlying idea is much older, cf. e.g. [4], [15], [14]). However, for the termination analysis of functional programs it makes sense to consider term rewriting instead of graph rewriting, for the following reasons. First of all, using term rewriting instead of graph rewriting is more general and does not prohibit certain evaluations a priori. Moreover, the theory of ordinary term rewriting is much further developed as compared to graph rewriting. In particular, existing methods for the termination analysis of ordinary as well as context-sensitive rewrite systems can be applied where possible. Hence, in this work we will use the notion of lazy term rewriting introduced in [11].

In [11] a transformation from lazy rewrite systems into context-sensitive ones was proposed, which was supposed to preserve non-termination and conjectured to be complete w.r.t. termination. Unfortunately, a counterexample (see Example 3.4 below) proves that this transformation is unsound w.r.t. termination. In this paper we repair the transformation and prove both soundness and completeness of the new transformation w.r.t. termination.

In Section 2 of this paper we will present basic definitions and notations of lazy rewriting. In Section 3 we introduce the transformation of [11] and give a counterexample to its soundness w.r.t. termination. We then propose a modified version of the transformation which is proved to be sound and complete w.r.t. termination. Section 4 contains a discussion of the presented approach and of some experimental results. [2]

2 Preliminaries

We assume familiarity with the basic concepts and notations in term rewriting as well as context-sensitive term rewriting as provided for instance in [2,9].

As in [3] and [11] we are concerned with left-linear lazy rewrite systems in this work.

General assumption: Throughout the paper we assume that all lazy rewrite systems are *left-linear* [3] and *finite*.

Lazy rewriting operates on labelled terms. Each function and variable symbol of a term has either an *eager* label e or a *lazy* label l which we will write as superscripts. So, given a signature $\Sigma = \{f_1, ..., f_n\}$, we consider a new signature $\Sigma' = \{f_1^e, f_1^l, ..., f_n^e, f_n^l\}$. We denote by V' the set of labelled variables, so $\mathcal{T}(\Sigma', V')$ is the set of labelled terms of a labelled signature Σ'. The notation t^e (resp. t^l) for a labelled term t indicates that t has an eager (resp. lazy) root label. Following [11] we use a replacement map μ to specify for each function $f \in \Sigma$ which arguments should be evaluated eagerly. Given a replacement map μ we define the *canonical labelling* of terms as a mapping $label_\mu \colon \mathcal{T}(\Sigma, V) \to \mathcal{T}(\Sigma', V')$, where Σ' is the labelled signature and V' are the labelled variables [11]:

$$\begin{aligned} label_\mu(t) &= label_\mu^e(t) \\ label_\mu^\alpha(x) &= x^\alpha \ (\alpha \in \{e, l\}) \\ label_\mu^\alpha(f(t_1, ..., t_n)) &= f^\alpha(label_\mu^{\alpha_1}(t_1), ..., label_\mu^{\alpha_n}(t_n)) \\ &\quad \text{where } \alpha_i = e \text{ if } i \in \mu(f),\ l \text{ otherwise, and } \alpha \in \{e, l\} \end{aligned}$$

Given a labelled term t, the unlabeled term $erase(t)$ is constructed from t by omitting all labels. A position p of a term t is said to be *eager* (resp. *lazy*), if the symbol at the root of the subterm starting at position p of t has an *eager* (resp. *lazy*) label. Note that the *lazy* positions of a term are not the same as the non-replacing positions in context-sensitive rewriting. The reason is that in lazy rewriting eager positions may occur below lazy ones whereas in context-sensitive rewriting all positions which are below a non-replacing position are non-replacing.

However, rewrite steps may only be performed at so-called *active* positions. A position p is called *active* if all positions on the path from the root to p are eager. Note that, given an unlabeled term t and a replacement map μ, the active positions of $label_\mu(t)$ are exactly the replacing positions w.r.t. context-sensitive rewriting.

---

[2] Due to lack of space, the proofs of some results (e.g., of Lemmata 3.13, 3.18 and 3.19) have been omitted here. They can be found in the technical report version of this paper [17].

[3] Nevertheless, for clarity we will mention this assumption in the main results.

Definition 2.1 ([11], [3]) *The* active *positions of a labelled term t (denoted $Act(t)$) are recursively defined as follows.*

- *The root position ϵ of t is* active.
- *If p is an* active *position and position $p.i$ is* eager, *then position $p.i$ is* active.

Example 2.2 *Consider a labelled term $f^e(a^l, g^e(h^l(a^e)))$. Positions $\epsilon, 2$ and $2.1.1$ are eager. Positions 1 and 2.1 are lazy and positions ϵ and 2 are active.*

Definition 2.3 ([11], [3]) *Let $l \in \mathcal{T}(\Sigma, V)$ be linear, $t \in \mathcal{T}(\Sigma', V')$ be a labelled term and let p be an* active *position of t. Then l matches $t|_p$ modulo laziness if either*

- *$l \in V$ or*
- *if $l = f(l_1, ..., l_n)$ and $t|_p = f^e(t_1^\alpha, ..., t_n^\alpha)$ $(\alpha \in \{e, l\})$, then for all* eager *subterms t_i^e, l_i matches modulo laziness t_i^e.*

If t_i^l at position $p.i$ is a lazy subterm and $l_i \notin V$, then position $p.i$ is called essential.

Informally, a matching modulo laziness is a partial matching ignoring (possible) clashes at lazy positions. Positions where such clashes occur may be activated (i.e., their label may be changed from lazy to eager).

Definition 2.4 ([11]) *Let $\mathcal{R} = (\Sigma, R)$ be a (left-linear) TRS. Let t be a labelled term and let l be the left-hand side of a rule of R. If l matches modulo laziness $t|_p$, and this matching gives rise to an essential position $p.i$ $(t|_{p.i} = f^l(t_1, ..., t_n))$, then $t \xrightarrow{A} t[f^e(t_1, ..., t_n)]_{p.i}$. The relation $\xrightarrow{A}_{\mathcal{R}}$ is called* activation relation.

Example 2.5 *Let $l = f(a, b)$ be a linear unlabeled left-hand side of a rule, and $t = f^e(a^e, c^l)$. Then l matches t modulo laziness giving rise to the* essential *position 2 and we have $f^e(a^e, c^l) \xrightarrow{A} f^e(a^e, c^e)$. The left-hand side l does not match the labelled term $f^e(a^e, c^e)$ modulo laziness.*

Definition 2.6 ([11]) *Let l be the (linear) left-hand side of a rewrite rule and let t be a labelled term. If l matches $erase(t)$, then the mapping $\sigma_{l,t} : Var(l) \to \mathcal{T}(\Sigma', V')$ is defined as follows. For all $x \in V$, with $l|_q = x$: $\sigma_{l,t}(x) = t|_q$.*

Informally, $\sigma_{l,t}$ is the matcher when matching l against $erase(t)$, where one adds the appropriate labels of t.

This substitution is modified to operate on labelled terms in the following way, yielding the mapping $\sigma_{s,t} \colon V' \to \mathcal{T}(\Sigma', V')$ [11]:

$$\sigma_{s,t}(x^e) = \begin{cases} y^e & \text{if } \sigma_{s,t}(x) = y^\alpha \in V' \\ f^e(t_1, ..., t_n) & \text{if } \sigma_{s,t}(x) = f^\alpha(t_1, ..., t_n) \end{cases}$$

$$\sigma_{s,t}(x^l) = \begin{cases} y^l & \text{if } \sigma_{s,t}(x) = y^\alpha \in V' \\ f^l(t_1, ..., t_n) & \text{if } \sigma_{s,t}(x) = f^\alpha(t_1, ..., t_n) \end{cases}$$

σ is homeomorphically extended to a mapping $\mathcal{T}(\Sigma', V') \to \mathcal{T}(\Sigma', V')$ as usual.

Definition 2.7 ([11]) *Let $\mathcal{R} = (\Sigma, R)$ be a (left-linear) TRS with replacement map*

μ. The active rewrite *relation* $\overset{R}{\to}_\mu$: $\mathcal{T}(\Sigma', V') \times \mathcal{T}(\Sigma', V')$ *is defined as follows: Let t be a labelled term such that the left-hand side of a rewrite rule $l \to r$ matches $erase(t|_p)$ with $\sigma_{l,t|_p}$ and let $p \in Act(t)$. Then $t \overset{R}{\to}_\mu t[\sigma_{l,t|_p}(label_\mu(r))]_p$.*

Informally, the active rewrite relation $\overset{R}{\to}_\mu$ performs rewrite steps according to rewrite rules as usual, but only at active positions, and taking labels into account when constructing the contractum.

Example 2.8 *Let $l = f(x, b) \to g(x)$ be a rewrite rule and $t = f^e(h^e(a^l), b^e)$ be a labelled term. Furthermore, consider a replacement map $\mu(f) = \mu(h) = \{1\}, \mu(g) = \emptyset$. Then $\sigma_{l,t}(x) = h^e(a^l)$ and $\sigma_{l,t}(x^l)$ as appearing in the labelled right-hand side of the rule is $h^l(a^l)$. Thus we have $f^e(h^e(a^l), b^e) \overset{R}{\to}_\mu g^e(h^l(a^l))$.*

The lazy rewrite relation $\overset{LR}{\to}_\mu$ is the union of the activation relation and the active rewrite relation.

Definition 2.9 ([11]) *Let $\mathcal{R}$ be a (left-linear) TRS and let μ be a replacement map for $\mathcal{R}$. The lazy rewrite relation $\overset{LR}{\to}_\mu$ induced by $(\mathcal{R}, \mu)$ is the union of the two relations $\overset{A}{\to}$ and $\overset{R}{\to}_\mu$ ($\overset{LR}{\to}_\mu = \overset{A}{\to}_\mathcal{R} \cup \overset{R}{\to}_\mu$).*

Example 2.10 *Consider the functional program of Example 1.1.*

$$from(x) \to x : from(s(x)) \qquad take(0, xs) \to []$$
$$take(x, []) \to [] \qquad take(s(x), y : ys) \to y : take(x, ys)$$

Now we interpret it as lazy term rewriting system with a replacement map μ given by $\mu(f) = \emptyset$ for all functions f. When evaluating the canonically labelled term $take^e(s^l(0^l), from^l(0^l))$ it is obvious that the subterm $from^l(0^l)$ must be activated and evaluated before take can be computed. According to the lazy rewrite relation we have: $take^e(s^l(0^l), from^l(0^l)) \overset{LR}{\to}_\mu take^e(s^l(0^l), from^e(0^l)) \overset{LR}{\to}_\mu take^e(s^l(0^l), 0^l : from^l(s^l(0^l))) \overset{LR}{\to}_\mu 0^l : take^e(0^l, from^e(s^l(0^l))) \overset{LR}{\to}_\mu 0^l$.

Definition 2.11 *Let $\mathcal{R}$ be a TRS with a replacement map μ. Then $\mathcal{R}$ is $LR(\mu)$-terminating if there is no infinite $\overset{LR}{\to}_\mu$-sequence starting from a term t, whose labelling is canonical or more liberal (i.e., whenever $label_\mu(erase(t))|_p$ is eager, then $t|_p$ is eager as well).*

Informally, we call a labelled term t more liberal than its canonically labelled version $label_\mu(erase(t))$ if it has strictly more eager labels. The reason for considering terms with canonical or more liberal labelling in the definition of $LR(\mu)$-termination is that only such terms appear in lazy reduction sequences starting from canonically labelled terms, in which we are actually interested.

Example 2.12 *Consider a lazy TRS consisting of one rule $f(a) \to b$ with a replacement map $\mu(f) = \emptyset$. Now, when considering the canonically labelled term $f^e(a^l)$ we can reduce it to $f^e(a^e)$ according to the lazy rewrite relation. The latter term is more liberally labelled than its canonically labelled version.*

Note that $LR(\mu)$-termination and well-foundedness of $\overset{LR}{\to}_\mu$ do not coincide in general. The reason is that $LR(\mu)$-termination concerns the non-existence of lazy reduction sequences starting from canonically (or more liberally) labelled terms, while there may still be infinite reduction sequences starting from other terms. Example 2.13 shows that the two notions are indeed different.

Example 2.13 *Consider the TRS* $\{g(f(a),c) \to a,\ h(x,f(b)) \to g(x,h(x,x))\}$ *with a replacement map* $\mu(f) = \mu(g) = \{1\}$ *and* $\mu(h) = \{1,2\}$. *This system is* $LR(\mu)$*-terminating. This can be shown with the transformation of Definition 3.8 and Theorem 3.20. However,* $\overset{LR}{\to}_\mu$ *is not well-founded:* $\underline{g^e(f^e(b^l), h^l(f^e(b^l), f^e(b^l)))} \overset{LR}{\to}_\mu g^e(f^e(b^l), h^e(f^e(b^l), f^e(b^l))) \overset{LR}{\to}_\mu g^e(f^e(b^l), \underline{g^e(f^e(b^l), h^l(f^e(b^l), f^e(b^l)))}) \overset{LR}{\to}_\mu \dots$ *Note that the term* b^l *at position* 1.1 *of the starting term is lazy while it would have been eager in the canonically labelled version of the starting term. This being more lazy is the key for the existence of the infinite reduction sequence.*

3 Transforming Lazy Rewrite Systems

3.1 Lucas' Transformation

We start with the definition of the transformation of [11], because it provides the basic ideas for our new one. The main idea of the transformation is to explicitly mimic activation steps of lazy rewriting through special activation rules in the transformed system which basically exchange function symbols to make them more eager (this goes back to [13]). Activations in lazy rewriting are possible at positions which correspond to a non-variable position of the left-hand side of some rule in a partial matching. This is why in the transformation we are concerned with non-variable lazy positions of left-hand sides of rules.

The transformation is iterative. In each iteration new rules are created until a fixed point is reached. The following definition identifies for a rule $l \to r$ and a position p the positions $p.i$ which are lazy in $label_\mu(l)$. These positions are dealt with in parallel in one step of the transformation.

Definition 3.1 ([11]) *Let* $l \to r$ *be a rewrite rule and* p *a non-variable position of* l*, then*

$$\mathcal{I}(l,p) = \{i \in \{1,...,ar(root(l|_p))\} \mid i \notin \mu(root(l|_p)) \wedge p.i \in Pos_\Sigma(l)\}.$$

Example 3.2 *Consider a rewrite rule* $l = f(a,b,x) \to r$ *and a replacement map* $\mu(f) = \{1\}$. *Then* $\mathcal{I}(l,\epsilon) = \{2\}$.

Definition 3.3 ([11]) *Let* $\mathcal{R} = (\Sigma, R)$ *be a TRS with replacement map* μ *and let* $\mathcal{I}(l,p) = \{i_1,\dots,i_n\} \neq \emptyset$ *for some rule* $l \to r \in R$ *and* $p \in Pos_\Sigma(l)$ *where* $root(l|_p) = f$. *The transformed system* $\mathcal{R}^\diamond = (\Sigma^\diamond, R^\diamond)$ *and* $\mu^\diamond$ *are defined as follows:*

- $\Sigma^\diamond = \Sigma \cup \{f_j \mid 1 \le j \le n\}$ *where* f_j *are new function symbols of arity* $ar(f_j) = ar(f)$
- $\mu^\diamond(f_j) = \mu(f) \cup \{i_j\}$ *for all* $1 \le j \le n$ *and* $\mu^\diamond(g) = \mu(g)$ *for all* $g \in \Sigma$

- $R^{\diamond} = R - \{l \to r\} \cup \{l'_{i_j} \to r \mid 1 \le j \le n\} \cup \{l[x]_{p.i_j} \to l'_j[x]_{p.i_j} \mid 1 \le j \le n\}$

where $l'_j = l[f_j(l|_{p.1}, ..., l|_{p.m})]_p$ *if* $ar(f) = m$, *and* x *is a fresh variable.*

The transformation of Definition 3.3 is iterated until arriving at a system $\mathcal{R}^{\natural} = (\Sigma^{\natural}, R^{\natural})$ and $\mu^{\natural}$ such that $\mathcal{I}(l,p) = \emptyset$ for every rule $l \to r \in R^{\natural}$ and every position $p \in Pos_{\Sigma}(l)$. For further motivation and examples concerning Definition 3.3 we refer to [11].

In [11] it remains unspecified how the pair l, p is selected in one step of the transformation. However, it turns out that the order in which those pairs are considered can be essential.

Example 3.4 *Consider the TRS* $\{f(g(a), a) \to a, b \to f(g(c), b)\}$ *with a replacement map* $\mu(f) = \{1\}$ *and* $\mu(g) = \emptyset$. *This system is not* $LR(\mu)$*-terminating:*

$$b^e \overset{LR}{\to}_{\mu} f^e(g^e(c^l), b^l) \overset{LR}{\to}_{\mu} f^e(g^e(c^l), b^e) \overset{LR}{\to}_{\mu} f^e(g^e(c^l), f^e(g^e(c^l), b^l)) \overset{LR}{\to}_{\mu} \dots$$

However, if we start the transformation with the first rule and position $p = \epsilon$, *and consider position* 1 *of the first rule in the second step of the transformation, then we arrive at the context-sensitive system*

$$\begin{aligned} f_2(g_1(a), a) &\to a & f(g'_1(a), x) &\to f_2(g(a), x) \\ f_2(g(x), a) &\to f_2(g_1(x), a) & f(g(x), y) &\to f(g'_1(x), y) \\ b &\to f(g(c), b) \end{aligned}$$

with $\mu(f) = \mu(g_1) = \mu(g'_1) = \{1\}$ *and* $\mu(f_2) = \{1, 2\}$.[4] *This system is* μ*-terminating (proved with AProVE [5]). The lazy reduction sequence starting from* b *cannot be mimicked anymore, because due to the two transformation steps first the argument of* g *has to be activated which prevents the activation of the* b *in the second argument of* f.

3.2 The New Transformation

3.2.1 Definition

In Lucas' transformation, positions that are dealt with last during the transformation must be activated first in rewrite sequences of the transformed system. This can be seen in Example 3.4 where $\mathcal{I}(f(g(a), a), \epsilon)$ is considered in the first step of the transformation but position 2 must be activated after position 1.1 (whose activation is enabled by a later transformation step considering $\mathcal{I}(f(g(a), x), 1)$).

Thus, the order in which lazy positions of rules are dealt with during the transformation is the reverse order in which they may be activated in the resulting transformed system. Since we want to simulate lazy rewriting, and in lazy rewriting only outermost lazy positions may be activated in a labelled term, we consider more inner lazy positions first in our new transformation. Therefore, with the resulting context-sensitive system more outer positions may be activated only before

[4] Here and subsequently the subscripts of function symbols indicate *additional* replacing positions. So the replacement map of a symbol f_i differs from that of f in that i is replacing in f_i.

more inner ones.

Despite considering more inner positions first in the transformation, we do not want to prioritize any (orthogonal) lazy positions. Thus, we define $\mathcal{I}(l)$ which identifies the innermost lazy positions in a term with respect to a given replacement map μ.

Definition 3.5

$$\mathcal{I}(l) = \{p \in Pos_\Sigma(l) \mid p \text{ is lazy in } label_\mu(l) \wedge \\ \wedge (\nexists q \in Pos_\Sigma(l) : q \text{ lazy in } label_\mu(l) \wedge q > p)\}.$$

Before presenting the formal definition of our new transformation (see Definition 3.8 below), which crucially relies on Definition 3.5, we want give an informal explanation and illustration of its essential features. In our transformation we distinguish two kinds of rules that are generated. On the one hand we have *activation* rules which are characterized by the fact that in these rules the left- and right-hand side differ only at exactly one position, where in the right-hand side a different function symbol as in the left-hand side is used. While having the same arity, the different function symbol in the right-hand side has exactly one more replacing position and the argument at that position is a variable (in both sides). All other rules are *active rewrite* rules.

The actual transformation proceeds in 3 stages. First, a set of initial *activation* rules is created. These rules enable the activation of one *innermost* position of a left-hand side of the original rules of the lazy TRS. As already indicated, by a rule activating position $p.i$ we mean a rule $l \to r$ where l and r differ only in the function symbol at position p and $p.i$ is replacing in r but non-replacing in l.

In the second stage one rule $l \to r$ (activating a position p) created in stage 1 (or stage 2) is replaced by a set of *activation* rules. This set contains two *activation* rules for each innermost lazy non-variable position of l. Let q be such a position (note that q is either above or orthogonal to p). Then the first of these two rules is a rule which activates q. Apart from that, both sides of this rule are identical to l (i.e. position p is still inactive). The second rule activates position p. However, in this rule position q is already active. So the second rule differs from the initial rule $l \to r$ only in that position q is replacing in both sides (cf. Example 3.6).

Example 3.6 *Assume an activation rule $l = f(g(x), a) \to f(g_1(x), a)$ was generated in step 1 of the transformation where $\mu(g) = \mu(f) = \emptyset$ and $\mu(g_1) = \{1\}$. The rule activates position $p = 1.1$. The left-hand side of this rule (if canonically labelled) has two innermost non-variable lazy positions $\mathcal{I}(l) = \{1, 2\}$. Thus, it will be replaced by a set of new rules. We first consider the innermost lazy position $q = 1$. So first, a rule is created which activates q where p is non-replacing (in this special case position p does not even occur in the rule). This rule is $f(y, a) \to f_1(y, a)$. Second, we create a rule which activates p while q already is active: $f_1(g(x), a) \to f_1(g_1(x), a)$. These two rules illustrate that in a system obtained by this kind of transformation more outer positions (like position 1) may be activated only* before *more inner ones (like 1.1). For the second innermost lazy position of the original rule we also obtain two rules:*

$$f(g(x),z) \to f_2(g(x),z) \qquad f_2(g(x),a) \to f_2(g_1(x),a)$$

We have $\mu(f_1) = \{1\}$ and $\mu(f_2) = \{2\}$. Note that all of the generated rules still have non-replacing non-variable positions in their left-hand sides. Hence, each of them must be transformed (and replaced) further in the same way as the original rule of this example was processed.

This construction ensures that with the obtained rules in every derivation q (which was considered after p in the transformation) is activated before p, which is sound as p is a more inner (or parallel) position compared to q (since it was considered first).

The latter construction is repeated until the rules obtained do not have any lazy (non-variable) positions. We would like to point out once again that, as we consider *innermost* positions of terms in stage one and one iteration of stage two in our transformation, the outermost lazy positions of the initial rules of the lazy system are dealt with last. Hence, these are the positions which may be activated first in reduction sequences of the transformed system.

In the third stage of the transformation for each rule of the original lazy system one *active rewrite* rule is created. This rule differs from the original rule from which it was created only in the fact that the left-hand side is fully activated, i.e. it contains no lazy non-variable position. The reason is that whenever an active reduction step is performed on a lazy sequence it can be simulated by first fully activating the redex with the generated activation rules and afterwards performing the actual active step. This is done in derivations of our transformed system.

Since all new function symbols which are introduced by the transformation are substituted for function symbols of the original signature, we define the mapping *orig* from the signature of the transformed system into the original signature which identifies for each new function symbol the original one for which it was substituted.

Definition 3.7 *Let $\mathcal{R} = (\Sigma, R)$ be a TRS with replacement map μ. If in one step of the transformation $f \in \Sigma$ is replaced by a new function symbol f', then $orig(f') = f$. Furthermore, if f' is substituted for a function symbol $g \notin \Sigma$, then $orig(f') = orig(g)$. For function symbols $h \in \Sigma$, we set $orig(h) = h$ and for variables we have $orig(x) = x$.*

Definition 3.8 *Let $\mathcal{R} = (\Sigma, R)$ be a TRS with replacement map μ. The transformed system $\widetilde{\mathcal{R}} = (\widetilde{\Sigma}, \widetilde{R})$ with $\widetilde{\mu}$ is constructed in the following three stages.*

1 **Generation of Initial Activation Rules.** *The transformed signature $\widetilde{\Sigma} \supseteq \Sigma$ and the set $A(l)$ for every rule $l \to r \in R$ are defined as the least sets satisfying*

$$l[x]_{p.i} \to l'[x]_{p.i} \in A(l) \text{ if } p.i \in \mathcal{I}(l) \text{ and } l' = l[f_i(l|_{p.1}, ..., l|_{p.n})]_p \qquad (1)$$
$$\wedge\ f_i \in \widetilde{\Sigma}$$
$$\wedge\ orig(g) = orig(h) \wedge \widetilde{\mu}(g) = \widetilde{\mu}(h) \Rightarrow g = h \text{ for all } g, h \in \widetilde{\Sigma}$$

where $\widetilde{\mu}$ is defined by $\widetilde{\mu}(f) = \mu(f)$ for all $f \in \Sigma$ and $\widetilde{\mu}(f_i) = \mu(orig(f_i)) \cup \{i\}$ if f_i was introduced in (1). Then we have $\widetilde{R} := \bigcup_{l \to r \in R} A(l)$.

2 **Saturation of Activation Rules.**

2.a **Processing one Activation Rule** *Let* $\widetilde{R} = A(l_1) \cup ... \cup A(l_m)$ *and let* $l \to r \in A(l_j)$ *for some* $j \in \{1, ..., m\}$ *such that* $\mathcal{I}(l)$ *is not empty. Then we modify the set* $A(l_j)$ *in the following way:*

$$A(l_j) = A(l_j) - \{l \to r\} \cup \{l[x]_{p.i} \to l'[x]_{p.i}\} \cup \{l' \to r'\}$$

for all $p.i \in \mathcal{I}(l)$ *where* $l' = l[f_i(l|_{p.1}, ..., l|_{p.n})]_p$ *and* $r' = r[f_i'(r|_{p.1}, ..., r|_{p.n})]_p$. *If there is no* $g \in \widetilde{\Sigma}$ *with* $orig(g) = orig(f_i)$ *and* $\widetilde{\mu}(g) = \widetilde{\mu}(root(l|_p)) \cup \{i\}$, *then* $\widetilde{\Sigma} = \widetilde{\Sigma} \cup \{f_i\}$ *and* $\widetilde{\mu}(f_i) = \widetilde{\mu}(root(l|_p)) \cup \{i\}$, *otherwise* $f_i = g$. *Analogously, if there is no* $g \in \widetilde{\Sigma}$ *with* $orig(g) = orig(f_i')$ *and* $\widetilde{\mu}(g) = \widetilde{\mu}(root(r|_p)) \cup \{i\}$, *then* $\widetilde{\Sigma} = \widetilde{\Sigma} \cup \{f_i'\}$ *and* $\widetilde{\mu}(f_i') = \widetilde{\mu}(root(r|_p)) \cup \{i\}$, *otherwise* $f_i' = g$.

$$\widetilde{R} := \bigcup_{l \to r \in R} A(l)$$

2.b **Iteration** *Step* 2.*a is iterated until for all rules* $l \to r$ *of* $\widetilde{R}$ *we have that* $\mathcal{I}(l) = \emptyset$.

3 **Generation of Active Rewrite Rules.** *For each rule* $l \to r \in R$ *we add one active rewrite rule to* $\widetilde{R}$. *For every position* $p \in Pos_\Sigma(l)$, *we consider the set*

$$Symb(p, l) = \{root(r'|_p) \mid l' \to r' \in A(l) \wedge p \in Pos_\Sigma(r')\}.$$

The function symbol which is least restrictive in this set (i.e., the maximal element of $\widetilde{\mu}(f)$ *w.r.t. the subset relation of all* $f \in Symb(p, l)$*) is unique and denoted by* $maxSymb(p, l)$. *We set*

$$\widetilde{R} := \widetilde{R} \cup \bigcup_{l \to r \in R} l'' \to r$$

where l'' *is given by* $Pos(l) = Pos(l'')$, $root(l''|_p) = maxSymb(p, l)$ *for all* $p \in Pos_\Sigma(l)$ *and* $root(l''|_p) = root(l|_p)$ *for all* $p \in Pos_V(l)$. *The signature of the transformed system is not altered in this stage.*

The transformation is terminating and produces a finite context-sensitive rewrite system (CSRS) provided that the input is finite. Furthermore, the function symbol $maxSymb(p, l)$ is unique for every rule $l \to r \in R$ and every $p \in Pos_\Sigma(l)$ (cf. [17] for further details).

Example 3.9 *Consider the TRS from Example 3.4* $\{l_1 = f(g(a), a) \to a, l_2 = b \to f(g(c), b)\}$ *with a replacement map* μ *s.t.* $\mu(f) = \{1\}$ *and* $\mu(g) = \emptyset$. *In the first stage of the transformation we have* $\mathcal{I}(l_1) = \{1.1, 2\}$ *and the following two* initial activation *rules are added (i.e.* $A(l_1)$*).* $\{f(g(x), a) \to f(g_1(x), a), f(g(a), x) \to f_2(g(a), x)\}$ *with* $\widetilde{\mu}(g_1) = \{1\}$ *and* $\widetilde{\mu}(f_2) = \{1, 2\}$. $A(l_2) = \emptyset$, *because* l_2 *does not contain any lazy non-variable positions. In step* 2.*a, the first rule of* $A(l_1)$ *is replaced by* $\{f(g(x), y) \to f_2(g(x), y), f_2(g(x), a) \to f_2(g_1(x), a)\}$ *where the position* $p.i$ *that was used is* 2 *(i.e.,* $\epsilon.2$*) and thus the new function symbol introduced is* f_2. *In the second iteration, the second rule of* $A(l_1)$ *is replaced by* $\{f(g(x), y) \to f(g_1(x), y), f(g_1(a), x) \to f_2(g_1(a), x)\}$. *Here, the position* $p.i$ *that was used is*

1.1 *and thus the new function symbol is* g_1. *Finally, the following* active rewrite rules *are added:* $\{f_2(g_1(a), a) \to a, b \to f(g(c), b)\}$. *For* l_1 *we have* $Symb(\epsilon, l_1) = \{f, f_2\}$, $Symb(1, l_1) = \{g, g_1\}$ *and* $Symb(1.1, l_1) = Symb(2, l_1) = \{a\}$. *Therefore,* $maxSymb(\epsilon, l_1) = f_2$, $maxSymb(1, l_1) = g_1$, $maxSymb(1.1, l_1) = a$ *and* $maxSymb(2, l_1) = a$. *For* l_2 *we trivially have* $maxSymb(\epsilon, l_2) = b$. *Hence, the system* $\widetilde{\mathcal{R}}$ *consists of*

$$\begin{aligned} f(g(x), y) &\to f_2(g(x), y) & f_2(g(x), a) &\to f_2(g_1(x), a) \\ f(g(x), y) &\to f(g_1(x), y) & f(g_1(a), x) &\to f_2(g_1(a), x) \\ f_2(g_1(a), a) &\to a & b &\to f(g(c), b) \end{aligned}$$

with $\widetilde{\mu}(f) = \widetilde{\mu}(g_1) = \{1\}$, $\widetilde{\mu}(f_2) = \{1, 2\}$ *and* $\widetilde{\mu}(g) = \emptyset$. $\widetilde{\mathcal{R}}$ *is not* $\widetilde{\mu}$*-terminating:*

$$\underline{b} \to_{\widetilde{\mu}} \underline{f(g(c), b)} \to_{\widetilde{\mu}} f_2(g(c), \underline{b}) \to_{\widetilde{\mu}} f_2(g(c), \underline{f(g(c), b)}) \to_{\widetilde{\mu}} \dots$$

Remark 3.10 Note that the above system could not have been derived with Lucas' transformation regardless of the order in which the positions are processed there. The reason is that when applied to this example, Lucas' transformation always enforces some order of activation of the two orthogonal lazy positions, whereas the new transformation does not.

The rest of the paper is concerned with the proof of soundness and completeness of the transformation of Definition 3.8 w.r.t. termination. First, we will deal with the simpler case of completeness.

3.2.2 Soundness and Completeness

Theorem 3.11 *Let* $\mathcal{R} = (\Sigma, R)$ *be a left-linear TRS with replacement map* μ, *and let* $\widetilde{\mathcal{R}} = (\widetilde{\Sigma}, \widetilde{R})$, $\widetilde{\mu}$ *be the transformed system (resp. replacement map) according to Definition 3.8. If* $\mathcal{R}$ *is* $LR(\mu)$*-terminating, then* $\widetilde{\mathcal{R}}$ *is* $\widetilde{\mu}$*-terminating.*

In order to prove the soundness of our transformation, we are going to show the existence of an infinite reduction sequence in the transformed system, whenever there is an infinite lazy reduction sequence in the original system. So assume there is an infinite lazy reduction sequence in a TRS $\mathcal{R}$ with replacement map μ. The first observation is that every lazy reduction sequence naturally corresponds to a context-free (i.e. ordinary) $\to_{\mathcal{R}}$-sequence, which performs the active rewrite steps of the lazy reduction sequence. We will construct a $\to_{\widetilde{\mu}}$-reduction sequence in the transformed system $\widetilde{\mathcal{R}}$ that corresponds to a context-free $\to_{\mathcal{R}}$-sequence, cf. Figure 1. Terms in the context-free $\to_{\mathcal{R}}$-sequence and terms in the corresponding $\widetilde{\mu}$-sequence are in a special relationship.

Definition 3.12 *Let* $\mathcal{R} = (\Sigma, R)$ *be a TRS,* μ *a replacement map and let* $s, t \in \mathcal{T}(\Sigma, V)$ *be two terms. Abusing notation we write* $s \to_{\mu^c}^* t$ *if and only if*

(i) *for all positions* $p \in Pos^{\mu}(t)$ *we have* $root(t|_p) = root(s|_p)$, *and*

(ii) *for all minimal positions* $q \in Pos(t) \setminus Pos^{\mu}(t)$ *we have* $s|_q \to_{\mu}^* s'$ *and* $s' \to_{\mu^c}^* t|_q$.

The idea behind $\to_{\mu^c}^*$ is that context-free reduction steps which occur at positions

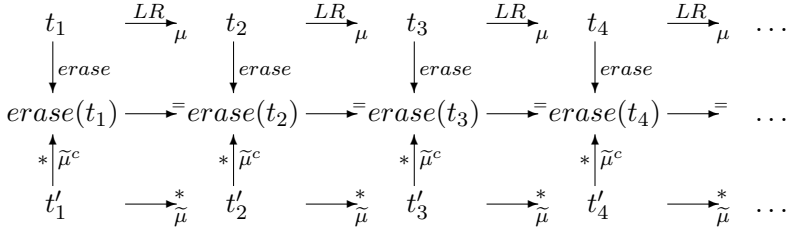

Fig. 1. Relation between the various rewrite sequences occurring in the soundness proof.

that are in the replacing part of the simulating term should be simulated, thus the replacing parts of two terms s and t with $s \to^*_{\mu^c} t$ must be entirely equal. On the other hand, context-free steps that occur at positions which are forbidden in the simulating term are ignored. Yet, if the forbidden subterm in which they occur eventually gets activated, then these steps may still be simulated.

As minimal non-replacing positions in a term are always strictly below the root, the recursive description of $\to^*_{\mu^c}$ in Definition 3.12 is well-defined.

We have $s = t \Rightarrow s \to^*_{\mu^c} t$. Figure 1 illustrates the correspondence between a lazy reduction sequence, the corresponding context-free one, and the $\to_{\widetilde{\mu}}$-sequence. Note that if the lazy reduction sequence is infinite, then there are infinitely many non-empty steps in the context-free reduction sequence, as every labelled term admits only finitely many activation steps.

In the first part of the soundness proof we show the existence of a $\to_{\widetilde{\mu}}$-sequence of the shape as in Figure 1.

The next lemma establishes the relationship between a context-free reduction sequence and a corresponding $\to_{\widetilde{\mu}}$ reduction of Figure 1.

Lemma 3.13 *Let $(\mathcal{R} = (\Sigma, R), \mu)$ be a TRS with replacement map and let $(\widetilde{\mathcal{R}} = (\widetilde{\Sigma}, \widetilde{R}), \widetilde{\mu})$ be the system obtained by the transformation of Definition 3.8. Let s and t be terms from $\mathcal{T}(\Sigma, V)$, such that $s \to^*_{\widetilde{\mu}^c} t$. If $t \xrightarrow{p} t^*$ (with a rule $l \to r$) and $p \in Pos^{\widetilde{\mu}}(s)$, then $s \to^+_{\widetilde{\mu}} s^*$ and $s^* \to^*_{\widetilde{\mu}^c} t^*$. Otherwise, if $t \xrightarrow{p} t^*$ and $p \notin Pos^{\widetilde{\mu}}(s)$, then $s \to^*_{\widetilde{\mu}} s^*$ and $s^* \to^*_{\widetilde{\mu}^c} t^*$.*

Unfortunately, the last lemma and the correspondence of lazy, context-free and $\to_{\widetilde{\mu}}$-reduction sequences of Figure 1 are not sufficient to prove the existence of an infinite $\to_{\widetilde{\mu}}$-sequence in the presence of an infinite lazy reduction sequence, since there may be only finitely many non-empty $\to_{\widetilde{\mu}}$-reductions in the simulating sequence.

Example 3.14 *Consider a TRS $\mathcal{R}$ consisting of the rules $\{a \to f(a), f(b) \to b\}$. This system is not LR_μ-terminating when considering a replacement μ with $\mu(f) = \emptyset$. Consider the infinite lazy reduction sequence*

$$\underline{a^e} \xrightarrow{LR}_\mu f^e(\underline{a^l}) \xrightarrow{LR}_\mu f^e(\underline{a^e}) \xrightarrow{LR}_\mu \dots$$

The transformed system is $\{a \to f(a), f(x) \to f_1(x), f_1(b) \to b\}$ with $\widetilde{\mu}(f) = \emptyset$ and $\widetilde{\mu}(f_1) = \{1\}$. We can simulate the infinite lazy reduction sequence by $\underline{a} \to_{\widetilde{\mu}} \underline{f(a)} \to_{\widetilde{\mu}} f_1(\underline{a}) \to_{\widetilde{\mu}} \dots$ Note that in this simulating derivation function symbols of

the extended signature are introduced which are not immediately "consumed" by a more outer active rewrite step. Example 3.15 shows that this can lead to problems.

Example 3.15 *Consider a non-terminating TRS* $\mathcal{R} = \{f(g(x)) \to f(g(x)), g(a) \to g(b), a \to c\}$ *with a replacement map* μ *given by* $\mu(f) = \{1\}$ *and* $\mu(g) = \emptyset$. *The transformed system* $\widetilde{\mathcal{R}}$ *is* $\{f(g(x)) \to f(g(x)), g(x) \to g_1(x), g_1(a) \to g(b), a \to c\}$ *with* $\widetilde{\mu}(f) = \widetilde{\mu}(g_1) = \{1\}$ *and* $\widetilde{\mu}(g) = \emptyset$. *Consider the context-free reduction sequence* $f(g(a)) \to f(g(c)) \to f(g(c)) \to \ldots$. *If we activate position* 1.1 *in* $f(g(a))$ *in the simulating* $\to_{\widetilde{\mu}}$*-sequence, we cannot further extend* $f(g(a)) \to_{\widetilde{\mu}} f(g_1(a)) \to_{\widetilde{\mu}} f(g_1(c))$, *because the term* $f(g_1(c))$ *is a* $\to_{\widetilde{\mu}}$*-normal form.*

The crucial difference why the activation of a subterm is essential in Example 3.14 and unnecessary in Example 3.15 is that in the former example the activated subterm itself initiates an infinite lazy reduction sequence. This observation will be used in the second part of the soundness proof (cf. Theorem 3.20).

When constructing an infinite reduction sequence in the transformed system corresponding to an infinite lazy sequence in the soundness proof, we will identify those activations that activate a non-terminating subterms s_{inf} and simulate them by activating the corresponding subterm s'_{inf} in the simulating sequence. Afterwards, we will focus only on an infinite lazy reduction sequence initiated by s_{inf}. This way the simulated activation, i.e., the introduction of a function symbol of the new signature, is of no relevance for the further simulation as it happened outside of s'_{inf}.

With the following definition we intend to identify labelled terms in an infinite lazy reduction sequence with non-terminating proper subterms that have possibly been activated. For such terms t, the predicate $mininf(t)$ does not hold.

Definition 3.16 *Let* Σ *be a signature and* μ *be a replacement map for* Σ. *A labelled term* t *is said to be* minimal non-terminating *if it admits an infinite lazy reduction sequence and for each eager labelled proper subterm* $t|_p$ *of* t, *either* $t|_p$ *does not initiate an infinite lazy rewrite sequence, or position* p *is eager in the term* $label_\mu(erase(t))$. *We write* $mininf(t)$ *if* t *has this property.*

Definition 3.17 *Let* $\mathcal{R} = (\Sigma, R)$ *be a TRS with replacement map* μ. *Let* t *be a labelled term* t *and* $t \overset{LR}{\to}_\mu s$ *be an activation step. This activation step is called* inf-activating *(thus it is an* inf-activation step*) if and only if* $mininf(t)$ *but* not $mininf(s)$.

It is easy to see that whenever $mininf(t)$ holds for a labelled term t, there is no active position $p \in Act(t)$ which is non-active in $label_\mu(erase(t))$, such that $t|_p$ initiates an infinite lazy reduction sequence.

In the second part of the soundness proof we will show that each infinite lazy reduction sequence contains either an inf-activation step or an active rewrite step $s \overset{LR}{\to}_\mu t$ at position p such that p is μ-replacing in $erase(s)$. Furthermore, such steps result in non-empty simulations by the $\to_{\widetilde{\mu}}$-sequence.

Lemma 3.18 *Let* $\mathcal{R} = (\Sigma, R)$ *be a TRS with replacement map* μ. *Let* t *be a labelled term satisfying* $mininf(t)$. *Then we have:*

(i) *If* $t \overset{LR}{\to}_\mu s$ *with an inf-activation step at position* q_1 *activating position* q_2 *and* $q_1 < p \leq q_2$ *is the maximal (w.r.t.* $\leq$*) eager position in* s *which does initiate an infinite reduction sequence s.t.* $t|_p$ *does not, then we have* $mininf(s|_p)$.

(ii) *If* $t \overset{LR}{\to}_\mu s$ *with any other step than in (i) (i.e., activation steps which are not inf-activating, or active rewrite steps), then* $mininf(s)$.

The next lemma characterizes infinite lazy reduction sequences starting from minimal non-terminating labelled terms. It states that in such an infinite lazy reduction sequence after finitely many steps there is either an active rewrite step $s_i \overset{LR}{\to}_\mu s_{i+1}$ at some position p which is active in $label_\mu(erase(s_i))$ or there is an inf-activation step. We already know from Lemma 3.13 that active rewrite steps at such positions can be simulated by a non-empty sequence in the transformed system (remember that the active rewrite steps of a lazy reduction sequence correspond to a context-free derivation). In Theorem 3.20 we will prove that the same is true for inf-activation steps.

Lemma 3.19 *Let* $\mathcal{R} = (\Sigma, R)$ *be a TRS with a replacement map* μ. *Let* t_0 *be a labelled term with the property* $mininf(t_0)$. *Let* $P : t_0 \overset{LR}{\to}_\mu t_1 \overset{LR}{\to}_\mu \ldots \overset{LR}{\to}_\mu t_n \overset{LR}{\to}_\mu \ldots$ *be an infinite lazy reduction sequence starting from* t_0. *Then, either there is an active rewrite step* $t_i \overset{LR}{\to}_\mu t_{i+1}$ *at position* p, *where* p *is active in* $label_\mu(erase(t_i))$, *or there is an* inf-activation *step in* P.

Theorem 3.20 *Let* $(\mathcal{R} = (\Sigma, R), \mu)$ *be a left-linear TRS with replacement map and let* $(\widetilde{\mathcal{R}} = (\widetilde{\Sigma}, \widetilde{R}), \widetilde{\mu})$ *be the system obtained by the transformation of Definition 3.8. If* $\widetilde{\mathcal{R}}$ *is* $\widetilde{\mu}$*-terminating, then* $\mathcal{R}$ *is* $LR(\mu)$*-terminating.*

Proof. (Sketch) [5] We will show that the existence of an infinite lazy reduction sequence $P : t_0 \overset{LR}{\to}_\mu t_1 \overset{LR}{\to}_\mu \ldots$ (where t_0 is canonically or more liberally labelled) implies the existence of an infinite reduction sequence in the transformed system. The following invariant will be maintained for every labelled term t_i of an infinite reduction sequence P. Let $s_0 \to_{\widetilde{\mu}} s_1 \to_{\widetilde{\mu}} \ldots$ be the simulating reduction sequence we are going to construct:

There is a s_j and a position o such that

$$s_j|_o \to^*_{\widetilde{\mu}^c} erase(t_i|_o) \wedge mininf(t_i|_o)$$

and position o is $\widetilde{\mu}$-replacing in s_j and active in t_i. Furthermore, $t_i|_o$ is at least as eager as its canonically labelled version (i.e., whenever $label_\mu(erase(t_i|_o))$ has an eager label at some position q, then the label of $t_i|_o$ is eager at that position, too). Note that the latter condition is trivially fulfilled by all terms t_i in P, and thus by all subterms, since no "deactivations" are possible in lazy rewriting and active rewrite steps only introduce canonically labelled terms.

We show that a finite initial subsequence of P implies the existence of a non-empty reduction sequence in the transformed system which preserves the invariant. As each term $t_i|_o$ itself initiates an infinite lazy reduction sequence, this suffices to

[5] For a more detailed version of the proof we refer to [17]

show that there is an infinite reduction sequence in the transformed system.

In order to apply Lemma 3.19, we assume $mininf(t_0)$. This minimality constraint can be satisfied, as w.l.o.g. we can find a t_0 such that *each* proper subterm of t_0 with an eager label does not initiate an infinite lazy reduction sequence.

The infinite reduction sequence we are going to construct in the transformed system starts with the term $s_0 = erase(t_0)$. We have $s_0 \to^*_{\widetilde{\mu}^c} erase(t_0)$.

Lemma 3.19 states that in the lazy reduction sequence starting from t_0 there is either an active rewrite step $t_i \overset{LR}{\to}_\mu t_{i+1}$ at position p such that p is active in $label_\mu(erase(t_i))$, or there is an inf-activation step. Let $t_j \overset{LR}{\to}_\mu t_{j+1}$ be the first such step.

The goal is to show that the reduction sequence $t_0 \overset{LR*}{\to}_\mu t_{j+1}$ can be simulated by a sequence $s_0 \to^+_{\widetilde{\mu}} s_i$ such that $s_i|_o \to^*_{\widetilde{\mu}^c} erase(t_{j+1}|_o)$ and $mininf(t_{j+1}|_o)$ holds for some position o which is active in t_{j+1} and replacing in s_i. We make a case distinction on whether the step $t_j \overset{LR}{\to}_\mu t_{j+1}$ is an active rewrite step or an *inf-activation* step.

(i) Assume the step $t_j \overset{LR}{\to}_\mu t_{j+1}$ is an active rewrite step at position p such that p is active in $label_\mu(erase(t_j))$. Then, by using Lemmata 3.13 and 3.18 we can derive that there exists a reduction sequence $s_0 \to^*_{\widetilde{\mu}} s_i \to^+_{\widetilde{\mu}} s_{i+1}$ such that $s_{i+1} \to^*_{\widetilde{\mu}^c} erase(t_{j+1})$ and $mininf(t_{j+1})$.

(ii) Assume the step $t_j \overset{LR}{\to}_\mu t_{j+1}$ is an inf-activation step. This implies that a subterm $t_{j+1}|_p$ is activated, which is non-$LR(\mu)$-terminating (furthermore, $mininf(t_{j+1}|_p)$ holds according to Lemma 3.18). Then we can construct a simulating sequence $s_0 \to^*_{\widetilde{\mu}} s_i \to^+_{\widetilde{\mu}} s_i^*$ where in the non-empty subsequence $s_i \to^+_{\widetilde{\mu}} s_i^*$ position p is activated and the subterm at position p is reduced in order to obtain $s_i^*|_p \to^*_{\widetilde{\mu}^c} erase(t_{j+1}|_p)$ (cf. Definition 3.12).

Now given an infinite lazy reduction sequence P starting from a labelled term t_0 and a term s_0 with $s_0 \to^*_{\widetilde{\mu}^c} erase(t_0)$, we have shown that a *finite* subsequence $t_0 \overset{LR+}{\to}_\mu t_i$ of a special shape implies the existence of a non-empty sequence $s_0 \to^+_{\widetilde{\mu}} s_j$ such that $s_j|_p \to^*_{\widetilde{\mu}^c} erase(t_i|_p)$, where p is active in t_i and replacing in s_j, $mininf(t_i|_p)$ holds and $t_i|_p$ initiates an infinite lazy reduction sequence. Thus, again the infinite lazy sequence starting at $t_i|_p$ has a finite subsequence, that can be simulated by a non-empty reduction sequence in the transformed system. By repeating this construction, we get an infinite reduction sequence in the transformed system starting at s_0. □

4 Experiments and Discussion

In the proof of Theorem 3.20 we saw that the context-sensitive rewrite system (CSRS) obtained by our transformation cannot simulate lazy reduction sequences in a one-to-one fashion. When simulating infinite lazy reduction sequences, after every inf-activation step in the lazy reduction an entirely new infinite lazy sequence was considered, namely the one initiated by the activated subterm. Thus, the ques-

tion arises whether we can define a transformation from lazy rewrite systems into context-sensitive ones, such that the transformed system is able to fully simulate the lazy reduction system. We conjecture that this is indeed possible ([16]), but would render termination proofs more difficult. The reason is that such a transformation would need to introduce even more rules that alter the status (i.e., lazy or eager) of positions in lazy terms (to be more precise, more activation rules would be needed).

Regarding the size of the transformed system, the number of rules created by our transformation is in general exponentially higher than the number of lazy non-variable subterms in left-hand sides of rules of the lazy system. However, we found that in our test series that was taken from the termination problem database ([18]) this number was not too high in most of the systems. This leads to the question whether our transformation is practically feasible in proofs of termination of real-world lazy TRSs. A series of tests were performed to answer this question, cf. [17]. As this is the first experimental termination analysis of lazy TRSs, neither lazy test TRSs nor benchmarks of other methods for proving lazy termination were readily available. Thus, we used the context-sensitive TRSs from the [18] and interpreted these systems as lazy ones. Out of 53 lazy systems that were tested, lazy termination of 35 could be shown. The tests were performed using *AProVE* ([5]) for proving context-sensitive termination of the CSRSs obtained by our transformation (with a time limit of 10 seconds). When interpreting these results one has to keep in mind that the example TRSs considered were actually supposed to be considered context-sensitive by their authors. Thus, in many cases the changes made by our transformation were only minimal. There were only 3 systems (namely `Ex1_Zan97.trs`, `Ex9_Luc06` and `Ex14_AEGL02`) for which termination could be proved in the context-sensitive case but not in the lazy case. However, all these systems actually become non-terminating when considered to be lazy instead of context-sensitive. This indicates that in many practical cases the analysis of lazy termination with our approach may well be feasible and a priori not too hard. Furthermore, lazy termination analysis can greatly benefit from ongoing research in the field of context-sensitive termination.

Acknowledgement

We would like to thank the anonymous reviewers of the previous workshop submission for numerous useful hints and criticisms.

References

[1] M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas. On-demand strategy annotations revisited. Technical Report DSIC–II/18/03, UPV, Valencia, Spain, 2003.

[2] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, New York, NY, USA, 1998.

[3] W. Fokkink, J. Kamperman, and P. Walters. Lazy rewriting on eager machinery. *ACM Transactions on Programming Languages and Systems*, 22(1):45–86, 2000.

[4] D. P. Friedman and D. S. Wise. CONS should not evaluate its arguments. In S. Michaelson and R. Milner, eds., *Proc. 3rd ICALP*, pp. 257–284. Edinburgh University Press, 1976.

[5] J. Giesl, R. Thiemann, and P. Schneider-Kamp. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In Ulrich Furbach and Natarajan Shankar eds., *Proc. IJCAR'06,* LNCS 4130, pp. 281–286, 2006.

[6] J. Giesl, S. Swiderski, R. Thiemann, and P. Schneider-Kamp. Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages In F. Pfenning, ed., *Proc. RTA'06,* LNCS 4098, pp. 297–312. Springer, 2006.

[7] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and Disproving Termination of Higher-Order Functions In B. Gramlich, ed., *Proc. FROCOS'05,* LNAI 3717, pp. 216–231. Springer, 2005.

[8] P. Henderson and J. H. Morris Jr. A lazy evaluator. In *Conference Record of the Third ACM Symp. on Principles of Programming Languages, Atlanta, Georgia, Jan. 1976*, pp. 95–103, 1976.

[9] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1), 1998.

[10] S. Lucas. Termination of on-demand rewriting and termination of OBJ programs. In H. Sondergaard, ed., *Proc. PPDP'01*, pp. 82–93, 2001. ACM Press, New York.

[11] S. Lucas. Lazy rewriting and context-sensitive rewriting. *In M. Hanus, ed., Proc. WFLP'01,* ENTCS 64, Elsevier, 2002.

[12] S. Lucas. Proving termination of context-sensitive rewriting by transformation. *Information and Computation*, 204(1):1782–1846, 2006.

[13] Q. H. Nguyen. Compact normalisation trace via lazy rewriting. In B. Gramlich and S. Lucas, eds., *Proc. WRS'01,* ENTCS 57, 2001.

[14] M. J. Plasmeijer and M. C. J. D. van Eekelen. *Functional programming and parallel graph rewriting.* Addison-Wesley, 1993.

[15] R. Strandh. Classes of equational programs that compile into efficient machine code. In N. Dershowitz, ed., *Proc. RTA'89*, LNCS 355, pp. 449–461. Springer, 1989.

[16] F. Schernhammer. On context-sensitive term rewriting. Master's thesis, TU Wien, February 2007. 2007.

[17] F. Schernhammer and B. Gramlich. Termination of lazy rewriting revisited. Technical Report E1852-2007-01 (available at `http://www.logic.at/people/schernhammer/papers/`), TU Wien, 2007.

[18] Termination Problem Database. Available at `http://www.lri.fr/~marche/tpdb`.