Transforming Cycle Rewriting into String Rewriting

David Sabel[1] and Hans Zantema[2,3]

1 Goethe University Frankfurt am Main, Institute for Computer Science
Frankfurt am Main, Germany
sabel@ki.informatik.uni-frankfurt.de

2 TU Eindhoven, Department of Computer Science,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.zantema@tue.nl

3 Radboud University Nijmegen, Institute for Computing and Information Sciences,
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

Abstract

We present new techniques to prove termination of cycle rewriting, that is, string rewriting on cycles, which are strings in which the start and end are connected. Our main technique is to transform cycle rewriting into string rewriting and then apply state of the art techniques to prove termination of the string rewrite system. We present three such transformations, and prove for all of them that they are sound and complete. Apart from this transformational approach, we extend the use of matrix interpretations as was studied before. We present several experiments showing that often our new techniques succeed where earlier techniques fail.

1998 ACM Subject Classification F.4.2 Grammars and other rewriting systems

Keywords and phrases rewriting systems, string rewriting, termination

Digital Object Identifier 10.4230/LIPIcs.RTA.2015.285

1 Introduction

String rewriting can not only be applied on strings, but also on cycles. Cycles can be seen as strings of which the left end is connected to the right end, by which the string has no left end or right end any more. Applying string rewriting on cycles is briefly called cycle rewriting. Rewriting behavior is strongly influenced by allowing cycles, for instance, in string rewriting the single rule $ab \to ba$ is terminating, but in cycle rewriting it is not, since the string ab represents the same cycle as ba. From the viewpoint of graph transformation, cycle rewriting is very natural. For instance, in [2] it was shown that if all rules of a graph transformation system are string rewrite rules, termination of the transformation system coincides with termination of the cycle rewrite system, and not with termination of the string rewrite system.

In many areas cycle rewriting is more natural than string rewriting. For instance, the problem of 5 dining philosophers can be expressed as a cycle $FTFTFTFTFT$ where F denotes a fork, and T denotes a thinking philosopher. Writing L for a philosopher who has picked up her left fork, but not her right fork, and E for an eating philosopher, a classical (deadlocking) modeling of the dining philosophers problem (for arbitrary many philosophers) can be expressed by the cycle rewrite system consisting of the rules $FT \to L$, $LF \to E$, $E \to FTF$. As a cycle rewrite system this is clearly not terminating.

So both string rewriting and cycle rewriting provide natural semantics for string rewrite systems, also called semi-Thue systems. Historically, string rewriting got a lot of attention as being a particular case of term rewriting, while cycle rewriting hardly got any attention until recently.

In [18] a first investigation of termination of cycle rewriting was made. Some techniques were presented to prove cycle termination, implemented in a tool `torpacyc`. Further a transformation ϕ was given such that for every string rewriting system (SRS) R, string termination of R holds if and only if cycle termination of $\phi(R)$ holds. As a consequence, cycle termination is undecidable. However, for making use of the strong power of current tools for proving termination of string rewriting in order to prove cycle termination, we need a transformation the other way around: transformations ψ such that for every SRS R, cycle termination of R holds if and only if string termination of $\psi(R)$ holds. The 'if' direction in this 'if and only if' is called 'sound', the 'only if' is called complete. A new way to prove cycle termination of an SRS R is to apply a tool for proving termination of string rewriting to $\psi(R)$ for a sound transformation ψ. The main topic of this paper is to investigate such transformations, and to exploit them to prove termination of cycle rewriting.

A similar approach to exploit the power of tools for termination of term rewriting to prove a modified property was used before in [8, 7]. However, there the typical observation was that the complete transformations were complicated, and for non-trivial examples termination of $\psi(R)$ could not be proved by the tools, while for much simpler sound (but incomplete) transformations ψ, termination of $\psi(R)$ could often be proved by the tools. In our current setting this is different: we introduce a transformation split, for which we prove that it is sound and complete, but we show that for several systems R for which all approaches from [18] fail, cycle termination of R can be concluded from an automatic termination proof of split(R) generated by AProVE [6, 1] or $\mathsf{T_TT_2}$ [12, 15].

It can be shown that if strings of size n exist admitting cycle reductions in which for every rule the number of applications of that rule is more than linear in n, then all techniques from [18] fail to prove cycle termination. Nevertheless, in quite simple examples this may occur while cycle termination holds. As an example consider the following.

A number of people are in a circle, and each of them carries a number, represented in binary notation with a bounded number of bits. Each of them may increase his/her number by one, as long as it fits in the bounded number of bits. Apart from that, every person may increase the number of bits of the number of its right neighbor by two. In order to avoid trivial non-termination, the latter is only allowed if the leading bit of the number is 0, and the new leading bit is put to 1, and the other to 0, by which effectively one extra bit is added. We will prove that this process will always terminate by giving an SRS in which all of the above steps can be described by a number of cycle rewrite steps, and prove cycle termination.

In order to do so we write P for person, and 0 and 1 for the bits of the binary number. For carry handling we introduce an extra symbol c of which the meaning is a 1 with a carry. Assume for every person its number is stored left from it. So if the number ends in 0, by adding one this last bit 0 is replaced by 1, expressed by the rule $0P \to 1P$. In case the number ends in 1, a carry should be created, since c represents a 1 with a carry this is expressed by the rule $1P \to cP$. Next the carry should be processed. In case it is preceded by 0, this 0 should be replaced by 1, while the c is replaced by 0; this is expressed by the rule $0c \to 10$. In case it is preceded by 1, a new carry should be created while again the old carry is replaced by 0; this is expressed by the rule $1c \to c0$. In this way adding one to any number in binary notation can be expressed by a number of rewrite steps, as long as no overflow occurs. Finally, we have to add a rule to extend the bit size of the number of the right neighbor: the leading bit should be 0, while it is replaced by 100: adding two extra bits of which the leading one is 1 and the other is 0. This is expressed by the rule $P0 \to P100$. Summarizing: we have to prove cycle termination of the SRS consisting of the five rules

$$0P \to 1P,\ 1P \to cP,\ 0c \to 10,\ 1c \to c0,\ P0 \to P100.$$

This is fundamentally impossible by the techniques presented in [18]: by one of the techniques the last rule can be removed, but starting in $0^n P$ a reduction can be made in which all of the remaining four rules are applied an exponential number of times, by which the techniques from [18] fail.

In this paper we give two ways to automatically prove that cycle termination holds for the above example R: $\mathsf{T_TT_2}$ succeeds in proving termination of $\mathsf{split}(R)$, and the other is a variant of matrix interpretations for which we show that it proves cycle termination.

The paper is organized as follows. Section 2 recalls the basics of cycle rewriting. The main section Section 3 presents three transformations split, rotate, and shift, and proves soundness and completeness of all of them. In Section 4 the matrix approach is revisited and extended. In Section 5 experiments on implementations of our techniques are reported. We conclude in Section 6.

2 Preliminaries

A *signature* Σ is a finite alphabet of symbols. With Σ^* we denote the set of strings over Σ. With ε we denote the empty string and for $u, v \in \Sigma^*$, we write uv for the concatenation of the strings u and v. With $|u|$ we denote the length of string $u \in \Sigma^*$ and for $a \in \Sigma$ and $n \in \mathbb{N}$, a^n denotes n replications of symbol a, i.e. $a^0 = \varepsilon$ and $a^i = aa^{i-1}$ for $i > 0$.

Given a binary relation $\to$, we write $\to^i$ for i steps, $\to^{\leq i}$ for at most i steps, $\to^{<i}$ for at most $i-1$ steps, $\to^*$ for the reflexive-transitive closure of $\to$, and $\to^+$ for the transitive closure of $\to_R$. For binary relations $\to_1$ and $\to_2$, we write $\to_1 . \to_2$ for the composition of $\to_1$ and $\to_2$, i.e. $a \to_1 . \to_2 c$ iff there exists a b s.t. $a \to_1 b$ and $b \to_2 c$.

A *string rewrite system* (SRS) is a finite set R of rules $\ell \to r$ where $\ell, r \in \Sigma^*$. The *rewrite relation* $\to_R \subseteq (\Sigma^* \times \Sigma^*)$ is defined as follows: if $w = u\ell v \in \Sigma^*$ and $(\ell \to r) \in R$, then $w \to_R urv$. The *prefix-rewrite relation* $\hookrightarrow_R$ is defined as: if $w = \ell u \in \Sigma^*$ and $(\ell \to r) \in R$, then $w \hookrightarrow_R ru$. The *suffix-rewrite relation* $\hookleftarrow_R$ is defined as: if $w = u\ell \in \Sigma^*$ and $(\ell \to r) \in R$, then $w \hookleftarrow_R ur$.

A (finite or infinite) sequence of rewrite steps $w_1 \to_R w_2 \to_R \cdots$ is called a *rewrite sequence* (sometimes also a *reduction* or a *derivation*). An SRS R is *non-terminating* if there exists a string $w \in \Sigma^*$ and an infinite rewrite sequence $w \to_R w_1 \to_R w_2 \cdots$. Otherwise, R is *terminating*.

We recall the notion of cycle rewriting from [18]. A string can be viewed as a cycle, i.e. the last symbol of the string is connected to the first symbol. To represent cycles by strings, we define the equivalence relation $\sim$ as follows:

$$u \sim v \text{ iff } u = w_1 w_2 \text{ and } v = w_2 w_1 \text{ for some strings } w_1, w_2 \in \Sigma^*$$

With $[u]$ we denote the equivalence class of string u w.r.t. $\sim$.

The *cycle rewrite relation* $\circ\!\!\to_R \subseteq (\Sigma/\!\sim \times \Sigma/\!\sim)$ of an SRS R is defined as

$$[u] \circ\!\!\to_R [v] \text{ iff } \exists w \in \Sigma^* : u \sim \ell w, (\ell \to r) \in R, \text{ and } v \sim rw$$

The cycle rewrite relation $\circ\!\!\to_R$ is called *non-terminating* iff there exists a string $w \in \Sigma^*$ and an infinite sequence $[w] \circ\!\!\to_R [w_1] \circ\!\!\to_R [w_2] \circ\!\!\to_R \cdots$. Otherwise, $\circ\!\!\to_R$ is called *terminating*.

We recall some known facts about cycle rewriting.

▶ **Proposition 1** (see [18]). *Let Σ be a signature, R be an SRS, and $u, v \in \Sigma^*$.*

1. *If $u \to_R v$ then $[u] \circ\!\!\to_R [v]$.*
2. *If $\circ\!\!\to_R$ is terminating, then $\to_R$ is terminating.*
3. *Termination of $\to_R$ does not necessarily imply termination of $\circ\!\!\to_R$.*
4. *Termination of $\circ\!\!\to_R$ is undecidable.*
5. *For every SRS R there exists a transformed SRS $\phi(R)$ s.t. the following three properties are equivalent:*
   - *$\to_R$ is terminating.*
   - *$\to_{\phi(R)}$ is terminating.*
   - *$\circ\!\!\to_{\phi(R)}$ is terminating.*

For an SRS R, the last property implies that termination of $\to_R$ can be proved by proving termination of the translated cycle rewrite relation $\circ\!\!\to_{\phi(R)}$. In [18] it was used to show that termination of cycle rewriting is undecidable and for further results on derivational complexity for cycle rewriting.

3 Transforming Cycle Termination into String Termination

Proposition 1 and the involved transformation ϕ, which transforms string rewriting into cycle rewriting, provide a method to prove string termination by proving cycle termination. However, it does not provide a method to prove termination of the cycle rewrite relation $\circ\!\!\to_R$ by proving termination of the string rewrite relations $\to_R$ or $\to_{\phi(R)}$. Hence, in this section we develop transformations ψ s.t. termination of $\to_{\psi(R)}$ implies termination of $\circ\!\!\to_R$. We call such a transformation ψ *sound*. However, there are "useless" sound transformations, for instance, transformations where $\psi(R)$ is always non-terminating. So at least one wants to find sound transformations which permit to prove termination of non-trivial cycle rewrite relations. However, a better transformation should fulfill the stronger property that $\to_{\psi(R)}$ is terminating if and only if $\circ\!\!\to_R$ is terminating. If termination of $\circ\!\!\to_R$ implies termination of $\to_{\psi(R)}$, then we say ψ is *complete*. For instance, for a complete transformation, non-termination proofs of $\to_{\psi(R)}$ also imply non-termination of $\circ\!\!\to_R$. Hence, our goal is to find sound and complete transformations ψ.

We will introduce and discuss three transformations split, rotate, and shift where the most important one is the transformation split, since it has the following properties: The transformation is sound and complete, and as our experimental results show, it behaves well in practice when proving termination of cycle rewriting. The other two transformations rotate and shift are also sound and complete, but rather complex and – as our experimental results show – they do not behave as well as the transformation split in practice. We include all three transformations in this paper to document some different approaches to transform cycle rewriting into string rewriting.

While we will analyze the transformation split in detail and prove its soundness and completeness, for the other two transformations we briefly list their properties where the corresponding proofs can be found in the longer version [13] of this paper.

3.1 The Transformation Split

The idea of the transformation split is to perform a single cycle rewrite step $[u] \circ\!\!\to_R [v]$ step which uses rule $(\ell \to r) \in R$, by either applying a string rewrite step $u \to_R v$ or by splitting the rule $(\ell \to r)$ into two rules $(\ell_A \to r_A)$ and $(\ell_B \to r_B)$, where $\ell = \ell_A\ell_B$ and $r = r_Ar_B$. Then a cycle rewrite step can be simulated by a prefix and a subsequent suffix rewrite step:

first apply rule $\ell_B \to r_B$ to a prefix of u and then apply rule $\ell_A \to r_A$ to a suffix of the obtained string.

▶ **Example 2.** Let $R = \{abc \to bbbb\}$ and $[bcdda] \multimap\!\!\to_R [bbddbb]$. The rule $abc \to bbbb$ can be split into the rules $a \to bb$ and $bc \to bb$ s.t. $bcdda \hookrightarrow_{\{bc\to bb\}} bbdda \hookleftarrow\!\!\to_{\{a\to bb\}} bbddbb$.

We describe the idea of the transformation split more formally. It uses the following observation of cycle rewriting: if $[u] \multimap\!\!\to_R [v]$, then $u \sim \ell w$, $(\ell \to r) \in R$, and $v \sim rw$. From $u \sim \ell w$ follows that $u = u_1u_2$ and $\ell w = u_2u_1$ for some u_1, u_2. We consider the cases for u_2:

1. If $u_i = \varepsilon$ (for $i = 1$ or $i = 2$), then $u = \ell w$ and $u \hookrightarrow_R rw$ by a prefix string-rewrite step.
2. If ℓ is a prefix of u_2, i.e. $\ell u_2' = u_2$, then $w = u_2'u_1$, $u = u_1\ell u_2' \to_R u_1ru_2'$, and $u_1ru_2' \sim rw$.
3. If u_2 is a proper prefix of ℓ, then there exist ℓ_A, ℓ_B with $\ell = \ell_A\ell_B$ s.t. $u_2 = \ell_A$ and ℓ_B is a true prefix of u_1, i.e. $u_1 = \ell_B w$ and $u = u_1u_2 = \ell_B w \ell_A \hookrightarrow_{\{\ell_B\to r_B\}} r_B w \ell_A \hookleftarrow\!\!\to_{\{\ell_A\to r_A\}} r_B w r_A \sim rw$ if $r_Ar_B = r$.

The three cases show that a cycle rewrite step $[u] \multimap\!\!\to_{\{\ell\to r\}} [v]$ can either be performed by applying a string rewrite step $u \to_{\{\ell\to r\}} v'$ where $v' \sim v$ (cases 1 and 2) or in case 3 by splitting $\ell \to r$ into two rules $\ell_A \to r_A$ and $\ell_B \to r_B$ such that $u \hookrightarrow_{\{\ell_B\to r_B\}} u'$ replaces a prefix of u by r_B and $u' \hookleftarrow\!\!\to_{\{\ell_A\to r_A\}} v'$ replaces a suffix of u' by r_A s.t. $v' \sim v$.

For splitting a rule $(\ell \to r)$ into rules $\ell_A \to r_A$ and $\ell_B \to r_B$, we may choose any decomposition of r for r_A and r_B (s.t. $r = r_Ar_B$). We will work with $r_A = r$ and $r_B = \varepsilon$.

The above cases for cycle rewriting show that a *sound* transformation of the cycle rewrite relation $\multimap\!\!\to_R$ into a string rewrite relation is the SRS which consists of all rules of R and all pairs of rules $\ell_B \to \varepsilon$ and $\ell_A \to r$ for all $(\ell \to r) \in R$ and all ℓ_A, ℓ_B with $|\ell_A| > 0$, $|\ell_B| > 0$, and $\ell = \ell_A\ell_B$. However, this transformation does not ensure that the rules evolved by splitting are used as prefix and suffix rewrite steps only. Indeed, the transformation in this form is useless for nearly all cases, since whenever the right-hand side r of a rule $(\ell \to r) \in R$ contains a symbol $a \in \Sigma$ which is the first or the last symbol in ℓ, then the transformed SRS is non-terminating. For instance, for $R = \{aa \to aba\}$ the cycle rewrite relation $\multimap\!\!\to_R$ is terminating, while the rule $a \to aba$ (which would be generated by splitting the left-hand side of the rule) leads to non-termination of the string rewrite relation. Note that this also holds if we choose any other decomposition of the right-hand side. Hence, in our transformation we introduce additional symbols to ensure:

- $\ell_B \to \varepsilon$ can only be applied to a prefix of the string.
- $\ell_A \to r$ can only be applied to a suffix of the string.
- If $\ell_B \to \varepsilon$ is applied to a prefix, then also $\ell_A \to r$ must be applied, in a synchronized manner (i.e. no other rule $\ell_B' \to \varepsilon$ or $\ell_A' \to r'$ can be applied in between).

In detail, we will prepend the fresh symbol B to the beginning of the string, and append the fresh symbol E to the end of the string. These symbols guarantee, that prefix rewrite steps $\ell u \hookrightarrow_{(\ell\to r)} ru$ can be expressed with usual string rewrite rules by replacing the left hand side ℓ with $\mathsf{B}\ell$ and analogous for suffix rewrite steps $u\ell \hookleftarrow\!\!\to_{(\ell\to r)} ur$ by replacing the left hand side ℓ with $\ell\mathsf{E}$. Let $(\ell_i \to r_i)$ be the i^{th} rule of the SRS which is split into two rules $\ell_B \to \varepsilon$ and $\ell_A \to r_i$, where $\ell_A\ell_B = \ell_i$. After applying the rule $\ell_B \to \varepsilon$ to a prefix of the string, the symbol B will be replaced by the two fresh symbols W (for "wait") and $\mathsf{R}_{i,j}$ where i represents the i^{th} rule and j means that ℓ_i has been split after j symbols (i.e. $|\ell_A| = j$). The fresh symbol L is used to signal that the suffix has been rewritten by rule $\ell_A \to r$. Finally, we use a copy of the alphabet, to ensure completeness of the transformation: for an alphabet Σ, we denote by $\overline{\Sigma}$ a fresh copy of Σ, i.e. $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$. For a word $w \in \Sigma^*$ with $\overline{w} \in \overline{\Sigma}^*$ we denote the word w where every symbol a is replaced by $\overline{a}$. Analogously, for a word $w \in \overline{\Sigma}^*$ with $\underline{w} \in \Sigma$ we denote w where every symbol $\overline{a}$ is replaced by the symbol a.

▶ **Definition 3** (The transformation split). Let $R = \{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$ be an SRS over alphabet Σ. Let $\overline{\Sigma}$ be a fresh copy of Σ and let $\mathsf{B}, \mathsf{E}, \mathsf{W}, \mathsf{R}_{i,j}, \mathsf{L}$ be fresh symbols (fresh for $\Sigma \cup \overline{\Sigma}$). The SRS $\mathsf{split}(R)$ over alphabet $\Sigma \cup \overline{\Sigma} \cup \{\mathsf{B}, \mathsf{E}, \mathbf{L}, \mathsf{W}\} \cup \bigcup_{i=1}^{n}\{\mathsf{R}_{i,j} \mid 1 \le j < |\ell_i|\}$ consists of the following string rewrite rules:

$$\begin{array}{rll}
\ell_i \to r_i & \text{for every rule } (\ell_i \to r_i) \in R & \text{(splitA)}\\
\overline{a}\mathsf{L} \to \mathsf{L}a & \text{for all } a \in \Sigma & \text{(splitB)}\\
\mathsf{W}\mathsf{L} \to \mathsf{B} & & \text{(splitC)}
\end{array}$$

and for every rule $(\ell_i \to r_i) \in R$, for all $1 \le j < |\ell_i|$ and $\ell_A\ell_B = \ell_i$ with $|\ell_A| = j$:

$$\begin{array}{rll}
\mathsf{B}\ell_B \to \mathsf{W}\mathsf{R}_{i,j} & & \text{(splitD)}\\
\mathsf{R}_{i,j}\ell_A\mathsf{E} \to \mathsf{L}r_i\mathsf{E} & & \text{(splitE)}\\
\mathsf{R}_{i,j}a \to \overline{a}\mathsf{R}_{i,j} & \text{for all } a \in \Sigma & \text{(splitF)}
\end{array}$$

We describe the intended use of the rules and the extra symbols. The symbols B and E mark the start and the end of the string, i.e. for a cycle $[u]$ the SRS $\mathsf{split}(R)$ rewrites $\mathsf{B}u\mathsf{E}$.

Let $[u] \mathrel{\circ\!\!\rightarrow}_R [w]$. The rule (splitA) covers the case that also $u \to_R w$ holds. Now assume that for $w' \sim w$ we have $u \hookrightarrow_{\{\ell_B \to \varepsilon\}} v \rightarrowtail_{\{\ell_A \to r\}} w'$ (where $(\ell_A\ell_B \to r) \in R$). Rule (splitD) performs the prefix rewrite step and replaces B by W to ensure that no other such a rule can be applied. Additionally, the symbol $\mathsf{R}_{i,j}$ corresponding to the rule and its splitting is added to ensure that only the right suffix rewrite step is applicable. Rule (splitF) moves the symbol $\mathsf{R}_{i,j}$ to right and rule (splitE) performs the suffix rewrite step. Rules (splitB) and (splitC) are used to finish the simulation of the cycle rewrite step by using the symbol L to restore the original alphabet and to finally replace $\mathsf{W}\mathsf{L}$ by B.

▶ **Example 4.** For $R_1 = \{aa \to aba\}$ the transformed string rewrite system $\mathsf{split}(R_1)$ is:

$$\begin{array}{llllllllll}
aa & \to aba & \text{(splitA)} & \overline{a}\mathsf{L} & \to \mathsf{L}a & \text{(splitB)} & \overline{b}\mathsf{L} & \to \mathsf{L}b & \text{(splitB)}\\
\mathsf{W}\mathsf{L} & \to \mathsf{B} & \text{(splitC)} & \mathsf{B}a & \to \mathsf{W}\mathsf{R}_{1,1} & \text{(splitD)} & \mathsf{R}_{1,1}a\mathsf{E} & \to \mathsf{L}aba\mathsf{E} & \text{(splitE)}\\
\mathsf{R}_{1,1}a & \to \overline{a}\mathsf{R}_{1,1} & \text{(splitF)} & \mathsf{R}_{1,1}b & \to \overline{b}\mathsf{R}_{1,1} & \text{(splitF)} & & &
\end{array}$$

For instance, the cycle rewrite step $[aba] \mathrel{\circ\!\!\rightarrow}_{R_1} [baba]$ is simulated in the transformed system by $\mathsf{B}aba\mathsf{E} \to \mathsf{W}\mathsf{R}_{1,1}ba\mathsf{E} \to \mathsf{W}\overline{b}\mathsf{R}_{1,1}a\mathsf{E} \to \mathsf{W}\overline{b}\mathsf{L}aba\mathsf{E} \to \mathsf{W}\mathsf{L}baba\mathsf{E} \to \mathsf{B}baba\mathsf{E}$. As a further example, for $R_2 = \{abc \to cbacba,\ aa \to a\}$ the transformed SRS $\mathsf{split}(R_2)$ is:

$$\begin{array}{lllllllll}
abc & \to cbacba & \text{(splitA)} & aa & \to a & \text{(splitA)} & \mathsf{W}\mathsf{L} & \to \mathsf{B} & \text{(splitC)}\\
\overline{a}\mathsf{L} & \to \mathsf{L}a & \text{(splitB)} & \overline{b}\mathsf{L} & \to \mathsf{L}b & \text{(splitB)} & \overline{c}\mathsf{L} & \to \mathsf{L}c & \text{(splitB)}\\
\mathsf{B}bc & \to \mathsf{W}\mathsf{R}_{1,1} & \text{(splitD)} & \mathsf{B}c & \to \mathsf{W}\mathsf{R}_{1,2} & \text{(splitD)} & \mathsf{B}a & \to \mathsf{W}\mathsf{R}_{2,1} & \text{(splitD)}\\
\mathsf{R}_{1,1}a\mathsf{E} & \to \mathsf{L}cbacba\mathsf{E} & \text{(splitE)} & \mathsf{R}_{1,2}ab\mathsf{E} & \to \mathsf{L}cbacba\mathsf{E} & \text{(splitE)} & \mathsf{R}_{2,1}a\mathsf{E} & \to \mathsf{L}a\mathsf{E} & \text{(splitE)}\\
\mathsf{R}_{1,1}a & \to \overline{a}\mathsf{R}_{1,1} & \text{(splitF)} & \mathsf{R}_{1,2}a & \to \overline{a}\mathsf{R}_{1,2} & \text{(splitF)} & \mathsf{R}_{2,1}a & \to \overline{a}\mathsf{R}_{2,1} & \text{(splitF)}\\
\mathsf{R}_{1,1}b & \to \overline{b}\mathsf{R}_{1,1} & \text{(splitF)} & \mathsf{R}_{1,2}b & \to \overline{b}\mathsf{R}_{1,2} & \text{(splitF)} & \mathsf{R}_{2,1}b & \to \overline{b}\mathsf{R}_{2,1} & \text{(splitF)}\\
\mathsf{R}_{1,1}c & \to \overline{c}\mathsf{R}_{1,1} & \text{(splitF)} & \mathsf{R}_{1,2}c & \to \overline{c}\mathsf{R}_{1,2} & \text{(splitF)} & \mathsf{R}_{2,1}c & \to \overline{c}\mathsf{R}_{2,1} & \text{(splitF)}
\end{array}$$

Termination of $\mathsf{split}(R_1)$ and $\mathsf{split}(R_2)$ can be proved by AProVE and TTT2.

▶ **Proposition 5** (Soundness of split). *If $\to_{\mathsf{split}(R)}$ is terminating then $\mathrel{\circ\!\!\rightarrow}_R$ is terminating.*

Proof. By construction of $\mathsf{split}(R)$, it holds that if $[u] \mathrel{\circ\!\!\rightarrow}_R [v]$, then $\mathsf{B}u'\mathsf{E} \to^{+}_{\mathsf{split}(R)} \mathsf{B}v'\mathsf{E}$ with $u \sim u'$ and $v \sim v'$. Thus for every infinite sequence $[w_1] \mathrel{\circ\!\!\rightarrow}_R [w_2] \mathrel{\circ\!\!\rightarrow}_R \cdots$ there exists an infinite sequence $\mathsf{B}\ w'_1\ \mathsf{E} \to_{\mathsf{split}(R)} \mathsf{B}\ w'_2\ \mathsf{E} \to_{\mathsf{split}(R)} \cdots$ with $w_i \sim w'_i$ for all i. ◀

3.1.1 Completeness of Split

We use type introduction [17] and use the sorts A, $\overline{A}$, C, and T, and type the symbols used by $\mathsf{split}(R)$ (seen as unary function symbols, and seen as a term rewrite system) as follows:

$$\begin{array}{lll} \mathsf{L} :: A \to \overline{A} & \mathsf{W} :: \overline{A} \to T & a \;\; :: A \to A \quad \text{for all } a \in \Sigma \\ \mathsf{B} :: A \to T & \mathsf{E} \;:: C \to A & \overline{a} \;\; :: \overline{A} \to \overline{A} \quad \text{for all } \overline{a} \in \overline{\Sigma} \\ & & \mathsf{R}_{i,j} :: A \to \overline{A} \quad \text{for all } \mathsf{R}_{i,j} \end{array}$$

We also add a constant c of sort C to the alphabet, s.t. ground terms for any sort exist. First one can verify that all rewrite rules of $\mathsf{split}(R)$ are well-typed: (splitA) rewrites terms of sort A, (splitB), (splitE), and (splitF) rewrite terms of sort $\overline{A}$, and (splitC) and (splitD) rewrite terms of sort T. Since there are no collapsing and duplicating rules, type introduction can be used (see [17]), i.e. (string) termination of the typed system is equivalent to (string) termination of the untyped system.

We consider ground terms of the sorts A, $\overline{A}$, C, and T. For simplicity we use the representation as strings (instead of terms), and we write E instead of Ec. First we show that our analysis of non-termination can be restricted to terms of sort T:

▶ **Lemma 6.** *If a term w of sort S with $S \in \{A, \overline{A}, C\}$ admits an infinite reduction w.r.t.* $\mathsf{split}(R)$*, then there exists a term w' of sort T, which admits an infinite reduction.*

Proof. The only term of sort C is the constant c which is a normal form. If a term w of sort A admits an infinite reduction w.r.t. $\to_{\mathsf{split}(R)}$, then also the term $\mathsf{B}w$ of type T admits an infinite reduction w.r.t. $\to_{\mathsf{split}(R)}$. If a term w of sort $\overline{A}$ admits an infinite reduction w.r.t. $\to_{\mathsf{split}(R)}$, then also the term $\mathsf{W}w$ of type T admits an infinite reduction w.r.t. $\to_{\mathsf{split}(R)}$. ◀

Inspecting the typing of the symbols shows:

▶ **Lemma 7.** *Any term of sort T is of one of the following forms:*

- $\mathsf{B}u\mathsf{E}$ *where* $u \in \Sigma^*$
- $\mathsf{W}w\mathsf{L}u\mathsf{E}$ *where* $w \in \overline{\Sigma}^*$ *and* $u \in \Sigma^*$
- $\mathsf{W}w\mathsf{R}_{i,j}u\mathsf{E}$ *where* $w \in \overline{\Sigma}^*$ *and* $u \in \Sigma^*$

We define a mapping from terms of sort T into strings over Σ as follows:

▶ **Definition 8.** For a term $w :: T$, the string $\Phi(w) \in \Sigma^*$ is defined according to the cases of Lemma 7:

$$\begin{array}{lcll} \Phi(\mathsf{B}u\mathsf{E}) & := & u & \\ \Phi(\mathsf{W}w\mathsf{L}u\mathsf{E}) & := & \underline{w}u & \\ \Phi(\mathsf{W}w\mathsf{R}_{i,j}u\mathsf{E}) & := & \ell_B\underline{w}u & \text{if } (\mathsf{B}\ell_B \to \mathsf{W}\mathsf{R}_{i,j}) \in \mathsf{split}(R) \end{array}$$

▶ **Lemma 9.** *Let w be of sort T and $w \to_{\mathsf{split}(R)} w'$. Then $\Phi(w) \circ\!\!\to_R^* \Phi(w')$.*

Proof. We inspect the cases of Lemma 7 for w:

- If $w = \mathsf{B}u\mathsf{E}$ where $u \in \Sigma^*$, then the step $w \to_{\mathsf{split}(R)} w'$ can use a rule of type (splitA) or (splitD). If rule (splitA) is applied, then $\Phi(w) \to_R \Phi(w')$ and thus $\Phi(w) \circ\!\!\to_R \Phi(w')$. If rule (splitD) is applied, then $w = \mathsf{B}\ell_2 u' \to_{\mathsf{split}(R)} \mathsf{W}\mathsf{R}_{i,j}u' = w'$ and $\Phi(w) = \ell_2 u' = \Phi(w')$.
- If $w = \mathsf{W}v\mathsf{L}u\mathsf{E}$ where $v \in \overline{\Sigma}^*$ and $u \in \Sigma^*$, then the step $w \to_{\mathsf{split}(R)} w'$ can use rules of type (splitA), (splitB), or (splitC). If rule (splitA) is used, then $\Phi(w) \to_R \Phi(w')$ and thus $\Phi(w) \circ\!\!\to_R \Phi(w')$. If rule (splitB) or (splitC) is used, then $\Phi(w) = \Phi(w')$.

- If $w = \mathsf{W}v\mathsf{R}_{i,j}u\mathsf{E}$ where $v \in \overline{\Sigma}^*$ and $u \in \Sigma^*$, then the step $w \to_{\mathsf{split}(R)} w'$ can use a rule of type (splitA), (splitE), or (splitF). If rule (splitA) is used, then $\Phi(w) \to_R \Phi(w')$ and thus $\Phi(w) \mathrel{\circ\!\!\to}_R \Phi(w')$. If rule (splitF) is used, then $\Phi(w) = \Phi(w')$. If rule (splitE) is used, then $w = \mathsf{W}v\mathsf{R}_{i,j}\ell_A\mathsf{E}$ and $w' = \mathsf{W}v\mathsf{L}r_i\mathsf{E}$ and $\Phi(w) = \ell_B\underline{v}\ell_A \sim \ell_A\ell_B\underline{v} \to_R r_i\underline{v} \sim \underline{v}r_i = \Phi(w')$ and thus $\Phi(w) \mathrel{\circ\!\!\to}_R \Phi(w')$. ◀

▶ **Theorem 10** (Soundness and completeness of split). *The transformation* split *is sound and complete, i.e.* $\to_{\mathsf{split}(R)}$ *is terminating if, and only if* $\mathrel{\circ\!\!\to}_R$ *is terminating.*

Proof. Soundness is proved in Proposition 5. It remains to show completeness. W.l.o.g. we assume that $\to_R$ is terminating, since otherwise $\mathrel{\circ\!\!\to}_R$ is obviously non-terminating. Type introduction and Lemma 6 show that it is sufficient to construct a non-terminating cycle rewrite sequence for any term w of sort T where w has an infinite $\to_{\mathsf{split}(R)}$-reduction. For every infinite reduction $w \to_{\mathsf{split}(R)} w_1 \to_{\mathsf{split}(R)} w_2 \cdots$ we use Lemma 9 to construct a cycle rewrite sequence $\Phi(w) \mathrel{\circ\!\!\to}_R^* \Phi(w_1) \mathrel{\circ\!\!\to}_R^* \Phi(w_2) \cdots$. It remains to show that the constructed sequence is infinite: one can observe that the infinite sequence must have infinitely many applications of rule (splitE) (which is translated by $\Phi(\cdot)$ into exactly one $\mathrel{\circ\!\!\to}_R$-step), since every sequence of $\xrightarrow{(\text{splitA})\vee(\text{splitB})\vee(\text{splitC})\vee(\text{splitD})\vee(\text{splitF})}$-steps is terminating (since we assumed that $\to_R$ is terminating). ◀

3.2 Alternative Transformations

We first present the general ideas of the transformations rotate and shift before giving their definitions. We write $\curvearrowright$ for the relation which moves the first element of a string to the end, i.e. $au \curvearrowright ua$ for every $a \in \Sigma$ and $u \in \Sigma^*$. We write $u \curvearrowright^{|\cdot|} v$ if, and only if $u \curvearrowright^{<|u|} v$ holds. Clearly, $u \sim v$ holds if and only if $u \curvearrowright^{<|u|} v$ holds. For a string rewrite system R, we define $\mathrm{len}(R)$ as the size of the largest left-hand side of the rules in R, i.e. $\mathrm{len}(R) = \max_{(\ell_i \to r_i) \in R} |\ell_i|$.

Using these notations we present two approaches to simulate cycle rewriting by string rewriting. One approach is to shift at most $\mathrm{len}(R) - 1$ symbols from the left end to the right end and then to apply a string rewrite step (this is the relation $\curvearrowright^{<\mathrm{len}(R)} . \to_R$). Another approach is to first rotate the string and then to apply a prefix rewrite step (this is the relation $\curvearrowright^{|\cdot|} . \hookrightarrow_R$).

▶ **Example 11.** As in Example 2, let $R = \{abc \to bbbb\}$ and $[bcdda] \mathrel{\circ\!\!\to}_R [bbddbb]$.

For the first approach ("shift") we shift symbols from the left end to the right end of the string until abc becomes a substring. Then we apply a string rewrite step, i.e. $bcdda \curvearrowright^{<\mathrm{len}(R)} . \to_R ddbbbb$, since $bcdda \curvearrowright cddab \curvearrowright ddabc \to_R ddbbbb$.

For the second approach ("rotate"), we rotate the string (by iteratively shifting symbols) until abc becomes a prefix. Then we apply a prefix rewrite step, i.e. $bcdda \curvearrowright^{|\cdot|} . \hookrightarrow_R bbbbdd$, since $bcdda \curvearrowright cddab \curvearrowright ddabc \curvearrowright dabcd \curvearrowright abcdd \hookrightarrow_R bbbbdd$.

It is quite easy to verify that the following proposition holds:

▶ **Proposition 12.** *Let R be an SRS. If $\mathrel{\circ\!\!\to}_R$ is non-terminating, then*
1. $\curvearrowright^{<\mathrm{len}(R)} . \to_R$ *admits an infinite reduction, and*
2. $\curvearrowright^{|\cdot|} . \hookrightarrow_R$ *admits an infinite reduction.*

For an SRS R the SRS $\mathsf{shift}(R)$ encodes the relation $\curvearrowright^{<\mathrm{len}(R)} . \to_R$ and the SRS $\mathsf{rotate}(R)$ encodes the relation $\curvearrowright^{|\cdot|} . \hookrightarrow_R$ where extra symbols are used to separate the steps, and copies of the alphabet underlying R are used to ensure completeness of the transformations. We provide the definitions and some explanations of the transformations shift and rotate, but for space reasons we exclude their proofs of soundness and completeness. They can be found in the longer version [13] of this paper. For the remainder of the section, we fix an SRS R over alphabet $\Sigma_A = \{a_1, \dots, a_n\}$. Let us write $\Sigma_B, \Sigma_C, \Sigma_D, \Sigma_E$ for fresh copies of the alphabet Σ_A. We use the following notation to switch between the alphabets: for $X, Y \in \{A, B, C, D, E\}$ and $w \in \Sigma_X$ we write $\llparenthesis w \rrparenthesis_Y$ to denote the copy of w in the alphabet Y where every symbol is translated from alphabet X to alphabet Y.

▶ **Definition 13** (The transformation shift)**.** Let R be an SRS over alphabet Σ_A and let $N = \max(0, \text{len}(R) - 1)$. The SRS $\mathsf{shift}(R)$ over the alphabet $\Sigma_A \cup \Sigma_B \cup \Sigma_C \cup \{\mathsf{B}, \mathsf{E}, \mathsf{W}, \mathsf{V}, \mathsf{M}, \mathsf{L}, \mathsf{R}, \mathsf{D}\}$ (where $\mathsf{B}, \mathsf{E}, \mathsf{W}, \mathsf{V}, \mathsf{M}, \mathsf{L}, \mathsf{R}, \mathsf{D}$ are fresh for $\Sigma_A \cup \Sigma_B \cup \Sigma_C$) consists of the following rules:

$$\begin{array}{rll r}
\mathsf{B} \to \mathsf{W}\mathsf{M}^N\mathsf{V} & & & \text{(shiftA)} \\
\mathsf{M} \to \varepsilon & & & \text{(shiftB)} \\
\mathsf{M}\mathsf{V}a \to \mathsf{V}\llparenthesis a \rrparenthesis_B & \text{for all } a \in \Sigma_A & & \text{(shiftC)} \\
ba \to ab & \text{for all } a \in \Sigma_A \text{ and all } b \in \Sigma_B & & \text{(shiftD)} \\
b\mathsf{E} \to \llparenthesis b \rrparenthesis_A \mathsf{E} & \text{for all } b \in \Sigma_B & & \text{(shiftE)} \\
\mathsf{W}\mathsf{V} \to \mathsf{R}\mathsf{L} & & & \text{(shiftF)} \\
\mathsf{L}a \to \llparenthesis a \rrparenthesis_C \mathsf{L} & \text{for all } a \in \Sigma_A & & \text{(shiftG)} \\
\mathsf{L}\ell \to \mathsf{D}r & \text{for all } (\ell \to r) \in R & & \text{(shiftH)} \\
c\mathsf{D} \to \mathsf{D}\llparenthesis c \rrparenthesis_A & \text{for all } c \in \Sigma_C & & \text{(shiftI)} \\
\mathsf{R}\mathsf{D} \to \mathsf{B} & & & \text{(shiftJ)}
\end{array}$$

The rules (shiftA) - (shiftE) encode the relation $\curvearrowright^{<\text{len}(R)}$, i.e. for $uv \in \Sigma_A^*$ with $|u| < \text{len}(R)$, the string $\mathsf{B}uv\mathsf{E}$ is rewritten into $\mathsf{W}\mathsf{V}vu\mathsf{E}$ by these five rules. The sequence of symbols M generated by rule (shiftA) denotes the potential of moving at most $\mathsf{len}(R) - 1$ symbols. The rules (shiftB) and (shiftC) either remove one from the potential or start the moving of one symbol. The rule (shiftD) performs the movement of a single symbol until it reaches the end of the string and rule (shiftE) finishes the movement.

The remaining rewrite rules perform a single string rewrite step, i.e. for a rule $(\ell \to r) \in R$ the string $\mathsf{W}\mathsf{V}w_1\ell w_2\mathsf{E}$ is rewritten to $\mathsf{B}w_1 r w_2\mathsf{E}$ by rules (shiftF) - (shiftJ).

This shows that if $u \curvearrowright^{<\text{len}(R)} v \to_R w$, then $\mathsf{B}u\mathsf{E} \to^+_{\mathsf{shift}(R)} \mathsf{B}w\mathsf{E}$ and thus by Proposition 12 soundness of the transformation shift holds.

▶ **Definition 14** (The transformation rotate)**.** Let R be an SRS over alphabet Σ_A. The SRS $\mathsf{rotate}(R)$ over the alphabet $\Sigma_A \cup \Sigma_B \cup \Sigma_C \cup \Sigma_D \cup \Sigma_E \cup \{$Begin, End, Rewrite, Goright, Guess, Rotate, Cut, Moveleft, Wait, Finish, Finish2$\}$ (where Begin, End, Rewrite, Goright, Guess, Rotate, Cut, Moveleft, Wait, Finish, and Finish2 are fresh for $\Sigma_A \cup \Sigma_B \cup \Sigma_C \cup \Sigma_D \cup \Sigma_E$) is:

$$\begin{array}{rll r}
\mathsf{Begin}\,\mathsf{End} \to \mathsf{Rewrite}\ \mathsf{End} & & & \text{(rotateA)} \\
\mathsf{Begin}\,a \to \mathsf{Rotate}\,\mathsf{Cut}\,\llparenthesis a \rrparenthesis_D \mathsf{Guess} & \text{for all } a \in \Sigma_A & & \text{(rotateB)} \\
\mathsf{Guess}\,a \to \llparenthesis a \rrparenthesis_D \mathsf{Guess} & \text{for all } a \in \Sigma_A & & \text{(rotateC)} \\
\mathsf{Guess}\,a \to \mathsf{Moveleft}\,\llparenthesis a \rrparenthesis_C \mathsf{Wait} & \text{for all } a \in \Sigma_A & & \text{(rotateD)} \\
\mathsf{Guess}\,\mathsf{End} \to \mathsf{Finish}\,\mathsf{End} & & & \text{(rotateE)} \\
d\,\mathsf{Moveleft}\,c \to \mathsf{Moveleft}\,c\,\llparenthesis d \rrparenthesis_B & \text{for all } c \in \Sigma_c \text{ and all } d \in \Sigma_D & & \text{(rotateF)} \\
\mathsf{Cut}\,\mathsf{Moveleft}\,c \to \llparenthesis c \rrparenthesis_E \mathsf{Cut}\,\mathsf{Goright} & \text{for all } c \in \Sigma_C & & \text{(rotateG)} \\
\mathsf{Goright}\,b \to \llparenthesis b \rrparenthesis_D \mathsf{Goright} & \text{for all } b \in \Sigma_B & & \text{(rotateH)} \\
\mathsf{Goright}\,\mathsf{Wait}\,a \to \mathsf{Moveleft}\,\llparenthesis a \rrparenthesis_C \mathsf{Wait} & \text{for all } a \in \Sigma_A & & \text{(rotateI)} \\
\mathsf{Goright}\,\mathsf{Wait}\,\mathsf{End} \to \mathsf{Finish}\,\mathsf{End} & & & \text{(rotateJ)} \\
d\,\mathsf{Finish} \to \mathsf{Finish}\,\llparenthesis d \rrparenthesis_A & \text{for all } d \in \Sigma_D & & \text{(rotateK)}
\end{array}$$

$$
\begin{array}{rll}
\mathsf{Cut}\,\mathsf{Finish} \to \mathsf{Finish2} & & \text{(rotateL)}\\
e\,\mathsf{Finish2} \to \mathsf{Finish2}\,(\!|e|\!)_A & \text{for all } e \in \Sigma_E & \text{(rotateM)}\\
\mathsf{Rotate}\,\mathsf{Finish2} \to \mathsf{Rewrite} & & \text{(rotateN)}\\
\mathsf{Rewrite}\,\ell \to \mathsf{Begin}\,r & \text{for all } (\ell \to r) \in R & \text{(rotateO)}
\end{array}
$$

We describe the intended use of the rewrite rules, where we ignore the copies of the alphabet in our explanations. The goal is that for any string $w \in \Sigma_A^*$, the string $\mathsf{Begin}\,w\,\mathsf{End}$ is rewritten to $\mathsf{Begin}\,u\,\mathsf{End}$, where $w \curvearrowright^{|\cdot|} . \hookrightarrow_R u$. The prefix rewrite step is performed by the last rule (rotateO). All other rules perform the rotation $\curvearrowright^{|\cdot|}$ s.t. $\mathsf{Begin}\,w\,\mathsf{End}$ is rewritten into $\mathsf{Rewrite}\,v\,\mathsf{End}$ where $w \sim v$. Instead of moving symbols from the front to the end (as $\curvearrowright$ does), the rules move a suffix of the string in front of the string (which has the same effect).

The first rewrite rule (rotateA) covers the case that w is empty. If $w = a_1 \ldots a_n$, then first choose a position to cut the string into two parts $w_1 w_2$. The symbol Guess is used for the non-deterministic selection of the position. Rule (rotateB) starts the rotate phase and the guessing, rule (rotateC) shifts the Guess-marker and rule (rotateD) stops the guessing. Rule (rotateE) covers the case that $w_2 = \varepsilon$ and no rotation will be performed. After stopping the guessing, every symbol of w_2 is moved in front of w_1, resulting in $w_2 w_1$. A typical situation is $a_{k+1} \ldots a_m a_1 \ldots a_k a_{m+1} \ldots a_n$ and now the symbol a_{m+1} must be moved in between a_m and a_1. To keep track of the position of a_1, the symbol Cut (inserted in front of a_1) marks the original beginning, and to keep track of the position of a_k, the symbol Wait (inserted after a_k) marks this position. The symbol Moveleft guards the movement of a_{m+1} (by rule (rotateF)). When arriving at the right place (rule (rotateG)), the symbol Goright is used to walk along the string (rule (rotateH)) to find the next symbol which has to be moved (rule (rotateI)). If all symbols are moved, rule (rotateJ) is applied to start the clean-up phase. There the symbols Finish and Finish2 are used to remove the markers and to replace the copied symbols of the alphabet with the original ones (rules (rotateK) – (rotateN)).

The construction of rotate(R) shows that $\mathsf{Begin}\,u\,\mathsf{End} \to^*_{\mathsf{rotate}(R)} \mathsf{Begin}\,w\,\mathsf{End}$ whenever $u \curvearrowright^{|\cdot|} v \hookrightarrow_R w$. Thus Proposition 12 implies soundness of rotate.

In the long version [13] of this paper, we also prove completeness of both transformations shift and rotate and thus the following theorem holds:

▶ **Theorem 15** (see [13]). *The transformations* shift *and* rotate *are sound and complete.*

4 Trace Decreasing Matrix Interpretations

In this section we present a variant of matrix interpretations suitable for proving cycle termination. The basics of matrix interpretations for string and term rewriting were presented in [9, 10, 5, 11]. The special case of tropical and arctic matrix interpretations for cycle rewriting was presented in [18], in the setting of *type graphs*. This section extends matrix interpretations for cycle rewriting along the lines suggested by Johannes Waldmann.

Fix a dimension $d > 0$. Define M_d to be the set of $d \times d$ matrices A over $\mathbb{N}$ for which $A_{11} > 0$. On M_d we define the relations $>$ and $\geq$ by

$$A > B \iff A_{11} > B_{11} \land \forall i,j : A_{ij} \geq B_{ij}, \qquad A \geq B \iff \forall i,j : A_{ij} \geq B_{ij}.$$

Write $\times$ for matrix multiplication. Note that $(A \times B) \in M_d$ whenever $A, B \in M_d$. The following lemma is easily checked.

▶ **Lemma 16.** *Let $A, B, C \in M_d$.*

- *If $A > B$ then $A \times C > B \times C$ and $C \times A > C \times B$,*

- *If $A \geq B$ then $A \times C \geq B \times C$ and $C \times A \geq C \times B$.*

A *matrix interpretation* $\langle\cdot\rangle$ for a signature Σ is defined to be a mapping from Σ to M_d. It is extended to $\langle\cdot\rangle : \Sigma^* \to M_d$ by defining inductively $\langle\varepsilon\rangle = I$ and $\langle ua\rangle = \langle u\rangle \times \langle a\rangle$ for all $u \in \Sigma^*$, $a \in \Sigma$, where I is the identity matrix.

▶ **Theorem 17.** *Let $R' \subseteq R$ be SRSs over Σ and let $\langle\cdot\rangle : \Sigma \to M_d$ such that*

- *$\circ\!\!\to_{R'}$ is terminating,*
- *$\langle\ell\rangle \geq \langle r\rangle$ for all $(\ell \to r) \in R'$, and*
- *$\langle\ell\rangle > \langle r\rangle$ for all $(\ell \to r) \in R \setminus R'$.*

Then $\circ\!\!\to_R$ is terminating.

Proof. For a square matrix A of dimension d, its *trace* $\mathsf{tr}(A)$ is defined to be $\sum_{i=1}^{d} A_{ii}$: the sum of its diagonal. It is well known and easy to check that $\mathsf{tr}(A \times B) = \mathsf{tr}(B \times A)$ for all A, B. As a consequence we obtain $\mathsf{tr}(\langle u\rangle) = \mathsf{tr}(\langle v\rangle)$ for u, v satisfying $u \sim v$. Since $A > B$ implies $\mathsf{tr}(A) > \mathsf{tr}(B)$ and $A \geq B$ implies $\mathsf{tr}(A) \geq \mathsf{tr}(B)$, from Lemma 16 we obtain $\mathsf{tr}(\langle u\rangle) \geq \mathsf{tr}(\langle v\rangle)$ if $u \to_{R'} v$, and $\mathsf{tr}(\langle u\rangle) > \mathsf{tr}(\langle v\rangle)$ if $u \to_{R\setminus R'} v$. Combining these observations yields $\mathsf{tr}(\langle u\rangle) \geq \mathsf{tr}(\langle v\rangle)$ if $[u] \circ\!\!\to_{R'} [v]$, and $\mathsf{tr}(\langle u\rangle) > \mathsf{tr}(\langle v\rangle)$ if $[u] \circ\!\!\to_{R\setminus R'} [v]$.

Assume an infinite $\circ\!\!\to_R$ reduction exists: $[u_1] \circ\!\!\to_R [u_2] \circ\!\!\to_R [u_3] \circ\!\!\to_R [u_4] \circ\!\!\to_R \cdots$. Since $\circ\!\!\to_{R'}$ is terminating, it contains infinitely many steps $[u_i] \circ\!\!\to_{R\setminus R'} [u_{i+1}]$, all giving rise to $\mathsf{tr}(\langle u_i\rangle) > \mathsf{tr}(\langle u_{i+1}\rangle)$, while all other steps give rise to $\mathsf{tr}(\langle u_i\rangle) \geq \mathsf{tr}(\langle u_{i+1}\rangle)$. As $\mathsf{tr}(\langle\cdot\rangle)$ always yields a natural number, this yields an infinite descending sequence of natural numbers, contradiction. ◀

Although this proof is not very hard, it is quite subtle: it is essential to first use the full order on matrices and disallow A_{11} to be 0 in order to obtain Lemma 16, and next apply the trace to get the same interpretations for u and v if $u \sim v$. It is not essential to apply this approach to natural numbers with usual addition and multiplication: other well-founded semirings can be used as well. Using the semiring $\mathbb{N} \cup \{\infty\}$ with minimum as semiring addition and $+$ as semiring multiplication yields *tropical matrix interpretations*. Using the semiring $\mathbb{N} \cup \{-\infty\}$ with maximum as semiring addition and $+$ as semiring multiplication yields *arctic matrix interpretations*. For general theory on matrix interpretations for term rewriting we refer to [10, 5]. Validity of tropical and arctic matrix interpretations for cycle rewriting has been proved in [18], in the setting of *type graphs*.

The original versions of matrix interpretations in [9, 5] fail for proving cycle termination since they succeed in proving termination of $aa \to bc$, $bb \to ac$, $cc \to ab$, for which cycle termination does not hold due to $[ccaa] \circ\!\!\to [abaa] \circ\!\!\to [abbc] \circ\!\!\to [aacc]$. The main difference is that in our setting the interpretation of symbols is multiplication by a matrix, while in [9, 5] it combines such a matrix multiplication by adding a vector.

The search for an application of Theorem 17 has been implemented in our tool `torpacyc`, extending the version presented in [18]. It is done by transforming the requirements to SMT format and calling the external SMT solver `Yices` [3, 16]. As an example consider the system from the introduction:

$$0P \to 1P,\ 1P \to cP,\ 0c \to 10,\ 1c \to c0,\ P0 \to P100.$$

First by finding a tropical matrix interpretation, the last rule is removed by `torpacyc`. Next the following matrices are found:

$$\langle P\rangle = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix},\ \langle 0\rangle = \begin{pmatrix} 1 & 2 \\ 0 & 2 \end{pmatrix},\ \langle 1\rangle = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix},\ \langle c\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}.$$

For these interpretations we obtain $\langle 0P\rangle > \langle 1P\rangle$, $\langle 1P\rangle > \langle cP\rangle$, $\langle 0c\rangle \geq \langle 10\rangle$ and $\langle 1c\rangle \geq \langle c0\rangle$, hence by Theorem 17 it suffices to prove cycle termination of $0c \to 10,\ 1c \to c0$, for which `torpacyc` finds a simple counting argument.

4.1 Limitations of the Method

In this section we give an example where the matrix approach fails and the transformational approach succeeds.

The method for proving cycle termination induced by Theorem 17 has similar limitations as the method of matrix interpretations in [10] for string termination: Since the entries of a product of n matrices are bounded by an exponential function in n, the method cannot prove cycle termination of systems which allow reduction sequences where every rewrite rule is applied more often than exponentially often.

▶ **Example 18.** The rewrite system $R_1 := \{ab \to bca, cb \to bbc\}$ allows for string derivations of a length which is a tower of exponentials (see [10]), i.e. the string $a^k b^k$ has such a long derivation, since the derivation $ab^n \to^*_{R_1} b^{2^n-1}c^n a$ exists and this can be iterated for every a in a^k. Moreover, the number of applications of the first and of the second rule of R_1 is a tower of exponentials. This shows that the matrix interpretations in [10] are unable to prove string termination of R_1. The system $\phi(R_1) := \{RE \to LE, aL \to La', bL \to Lb', cL \to Lc', Ra' \to aR, Rb' \to bR, Rc' \to cR, abL \to bcaR, cbL \to bbcR\}$ uses the transformation ϕ from [18] and transforms the string rewrite system R_1 into a cycle rewrite system s.t. R_1 is string terminating iff $\phi(R_1)$ is cycle terminating. One can verify that $[ab^nLE] \circ\!\!\to^*_{\phi(R_1)} [b^{2^n-1}c^n aLE]$ which can also be iterated s.t. $[a^k b^k LE]$ has a cycle rewriting sequence whose length is a tower of k exponentials. Inspecting all nine rules of $\phi(R_1)$, the number of applications of any of the rules in this rewrite sequence is also a tower of k exponentials and thus it is impossible to prove cycle termination of $\phi(R_1)$ using Theorem 17. Consequently, our tool `torpacyc` does not find a termination proof for $\phi(R_1)$.

▶ **Remark 19.** As expected our tool `torpacyc` does not find a termination proof for $\phi(R_1)$ from Example 18. On the other hand, with our transformational approach a cycle termination can be proved: AProVE proves strings termination of $\mathsf{split}(\phi(R_1))$.

A further question is whether matrix interpretations are limited to cycle rewrite systems with exponential derivation lengths only. The following example shows that this is not true:

▶ **Example 20.** The SRS $R_2 := \{ab \to baa, cb \to bbc\}$ (see [10]) has derivations of doubly exponential length (since $ac^kb \to^*_{R_2} b^{2^k} a^{2^{2^k}} c^k$ and any rewrite step adds one symbol), but its string termination can be proved by relative termination and matrix interpretations by first removing the rule $cb \to bbc$ and then removing the other rule. This is possible, since the second rule is applied only exponentially often. For cycle rewriting the encoding ϕ from [18] is $\phi(R_2) = \{RE \to LE, aL \to La', bL \to Lb', cL \to Lc', Ra' \to aR, Rb' \to bR, Rc' \to cR, abL \to baaR, cbL \to bbcR\}$ and $\phi(R_2)$ is cycle terminating iff R_2 is string terminating. The system $\phi(R_2)$ also has doubly exponential cycle derivations, e.g. $[ac^kbLE] \circ\!\!\to^*_{\phi(R_2)} [b^{2^k} a^{2^{2^k}} c^k LE]$. However, `torpacyc` proves cycle termination of $\phi(R_2)$ by first removing the last rule using the matrix interpretation

$$\langle R\rangle = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}, \langle E\rangle = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}, \langle L\rangle = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}, \langle a\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \langle a'\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$
$$\langle b\rangle = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, \langle b'\rangle = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \langle c\rangle = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}, \langle c'\rangle = \begin{pmatrix} 1 & 0 \\ 1 & 3 \end{pmatrix}.$$

Thereafter the remaining rules (which now only have derivations of exponential length) are eliminated by matrix interpretations and counting arguments.

Note that the methods in [18] are not able to prove cycle termination of $\phi(R_2)$, since they can only remove rules which are applied polynomially often in any derivation.

Also the transformational approach successfully proves cycle termination of $\phi(R_2)$: $\mathsf{T_TT_2}$ proves string termination of $\mathsf{split}(\phi(R_2))$. Interestingly, we did not find a termination proof of $\mathsf{split}(\phi(R_2))$ using AProVE.

5 Tools and Experimental Results

We implemented the search for matrix interpretations described in Section 4 in `torpacyc`. We also implemented a tool which transforms an SRS by split, rotate, or shift. We also automatized the proof of cycle termination by a command line tool which can either use `torpacyc` to prove cycle termination or by first performing one of the transformations, and then using one of the termination provers AProVE or $\mathsf{T_TT_2}$ to prove string termination of the transformed problem. Also two kinds of combinations of both approaches are implemented:

- variant 1: first `torpacyc` tries to find a termination proof and if no proof is found, the (perhaps simplified) cycle rewrite system is transformed into a string rewrite system by split and then used as input for AProVE or $\mathsf{T_TT_2}$.
- variant 2: first string non-termination of the rewrite system is checked by AProVE or $\mathsf{T_TT_2}$. If a non-termination proof is found, then also the cycle rewrite system is non-terminating. Otherwise, the same procedure as in variant 1 is used.

The automatic transformation and the prover can also be used online via a web interface available via `http://www.ki.informatik.uni-frankfurt.de/research/cycsrs/` where also the tools and experimental data can be found.

5.1 Experiments

A problem for doing experiments is that no real appropriate test set is available. We played around with several examples like the ones in this paper, but we were also interested in larger scale experiments. To that end we chose two problem sets. The first set is the `SRS_Standard`-branch of the Termination Problem Data Base [14], which is a benchmark set for proving termination of string rewrite systems. The second set consists of 50 000 randomly generated cycle rewrite systems of size 12 over an alphabet of size 3. For all problems we tried to prove cycle termination using the following methods (all with a time limit of 60 seconds):

- `torpacyc`: We applied `torpacyc` to the problems, where version 2014 is described in [18] and version 2015 includes the matrix interpretations described in Section 4.
- split, rotate, shift: We transformed the problem by the transformation into a string termination problem and then applied the termination provers AProVE or $\mathsf{T_TT_2}$, respectively.
- combinations: We also tried the combination of the methods as described before (using AProVE and $\mathsf{T_TT_2}$) where for variant 1, `torpacyc` as well as the second termination prover had a time limit of 30 seconds, and for variant 2, both – the non-termination check and `torpacyc` – had a time limit of 15 seconds, and the termination prover applied to the transformed problem (i.e. AProVE or $\mathsf{T_TT_2}$) had a time limit of 30 seconds.

Table 1 Results of proving cycle termination of the problems in SRS_Standard of the TPDB.

									combination					string	
	torpacyc		split		rotate		shift		variant 1		variant 2			termination	
	2014	2015	AProVE	TTT$_2$	AProVE	TTT$_2$	AProVE	TTT$_2$	AProVE	TTT$_2$	AProVE	TTT$_2$	**any**	AProVE	TTT$_2$
yes	36	46	40	30	10	6	10	8	55	55	54	54	**63**	652	627
no	0	0	309	168	45	0	65	0	310	161	335	173	**336**	97	38
maybe	1289	1269	966	1117	1260	1309	1240	1307	950	1099	926	1088	**916**	566	650

5.1.1 String Rewrite Systems of the Termination Problem Data Base

We tested all 1315 string rewrite problems of the `SRS_Standard`-branch of the Termination Problem Data Base [14]. Table 1 shows the summary of the performed tests, where in the column titled with"any" the output of all tools are combined (per problem). The last two columns show the results of proving string termination of the original problems in our test environment using AProVE and TTT$_2$. Note that a non-termination proof of string rewriting also implies non-termination of cycle rewriting, which does not hold for termination proofs. The results show that having sound and complete transformations is advantageous (compared to having sound transformations only), since we were able to prove cycle non-termination for 336 problems by the transformational approach (and by using string non-termination). Note that `torpacyc` (in both version) has no technique to prove non-termination.

However, the results also show that most of the problems in the test set are too hard to be proved by the techniques (only 399 out of 1315 problems were shown to be terminating or non-terminating). This is not really surprising since the test set contains already 'hard' instances for proving string termination as shown by the results in the last two columns. Moreover, a substantial part of the problems is expected not to be cycle terminating, for instance by containing a renaming of $ab \to ba$. This is confirmed by the result of non-termination by split and AProVE for over 300 of the systems.

Comparing the three transformations, the transformation split leads to much better results than the other two transformations, which holds for termination and for non-termination proofs. Ignoring the non-termination results (since `torpacyc` does not check for non-termination), the following observations are made: the new version of `torpacyc` indeed improves the former version, the power of the transformation split together with AProVE compared to `torpacyc` 2015 is more or less equal, while other transformations and back-ends do not perform as well as these tools. The combination of techniques increases the power, not only since the different tools perform well on different problems, but also, since `torpacyc` passes its output (a perhaps simplified SRS) to the string termination prover.

5.1.2 Randomly Generated String Rewrite Systems

As a further test set which contains much simpler problems than the former test set, we generated 50 000 SRSs of size 12 over an alphabet of size 3, where only SRSs with at least one rule $(\ell \to r)$ with $|r| \geq |\ell|$ are considered (to rule out obviously terminating problems), SRSs with rules $(\ell \to u\ell v)$ are not considered (to rule out obviously non-terminating problems), only SRSs without collapsing rules are considered (since `torpacyc` 2014 cannot handle such problems), and no alpha-equivalent (i.e. renamed and reordered) SRSs are generated.

The summarized results of applying our methods to this problem set are shown in Table 2.

Table 2 Results of proving cycle termination of 50 000 randomly generated problems of size 12 over an alphabet of size 3.

	torpacyc		split		rotate		shift		combination variant 1		combination variant 2		
	2014	2015	AProVE	$\mathsf{T_TT_2}$	AProVE	$\mathsf{T_TT_2}$	AProVE	$\mathsf{T_TT_2}$	AProVE	$\mathsf{T_TT_2}$	AProVE	$\mathsf{T_TT_2}$	**any**
yes	46981	46929	47017	47073	36967	36260	37053	37027	47071	46967	47064	47015	**47124**
no	0	0	2331	2201	184	0	771	0	2328	2011	2334	2174	**2339**
maybe	3019	3071	652	726	12849	13740	12176	12973	601	1022	602	811	**537**

Most of the problems are cycle terminating and both versions of `torpacyc` as well as the transformation split together with AProVE or $\mathsf{T_TT_2}$ can show termination of most of the problems. The problems seem to be too simple to separate the power of these successful approaches and their combinations. However, for the two transformations rotate and shift, the results show their lack in power, since no proof is found for roughly a quarter of all problems.

6 Conclusions

We developed new techniques to prove cycle termination. The main approach is to reduce the problem of cycle termination to the problem of string termination by applying a sound and complete transformation from cycle into string rewriting. We presented and analyzed three such transformations. Apart from that we provided a variant of matrix interpretations which improves the approach presented in [18]. Our implementations and the corresponding experimental results show that both techniques are useful in the sense that they apply for several examples for which the earlier techniques failed.

Together with the sound and complete transformation ϕ in the reverse direction from [18], the existence of a sound and complete transformation like split implies that the problems of cycle termination and string termination of SRSs are equivalent in a strong sense. For instance, it implies that they are in the same level of the arithmetic hierarchy, which is Π_2^0-complete along the lines of [4]. Alternatively, Π_2^0-completeness of cycle termination can be concluded from the sound and complete transformation ϕ combined with the observation that cycle termination is in Π_2^0.

Acknowledgments. We thank Johannes Waldmann for fruitful remarks, in particular for his suggestions leading to Section 4 on trace decreasing matrix interpretations. We also thank the anonymous reviewers for their valuable comments.

References

1 Homepage of AProVE, 2015. `http://aprove.informatik.rwth-aachen.de`.

2 H. J. Sander Bruggink, Barbara König, and Hans Zantema. Termination analysis for graph transformation systems. In Josep Diaz, Ivan Lanese, and Davide Sangiorgi, editors, *Proc. 8th IFIP International Conference on Theoretical Computer Science*, volume 8705 of *Lecture Notes in Comput. Sci.*, pages 179–194. Springer, 2014.

3 Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Comput. Sci.*, pages 737–744. Springer, 2014.

4 Jörg Endrullis, Herman Geuvers, Jakob Grue Simonsen, and Hans Zantema. Levels of undecidability in rewriting. *Inf. Comput.*, 209(2):227–245, 2011.

5 Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reasoning*, 40(2-3):195–220, 2008.

6 Jürgen Giesl, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Proving termination of programs automatically with AProVE. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Proc. 7th International Joint Conference on Automated Reasoning (IJCAR'14)*, volume 8562 of *Lecture Notes in Comput. Sci.*, pages 184–191. Springer, 2014.

7 Jürgen Giesl and Aart Middeldorp. Transformation techniques for context-sensitive rewrite systems. *J. Funct. Program.*, 14(4):379–427, 2004.

8 Jürgen Giesl and Hans Zantema. Liveness in rewriting. In Robert Nieuwenhuis, editor, *Proc. 14th Conference on Rewriting Techniques and Applications (RTA)*, volume 2706 of *Lecture Notes in Comput. Sci.*, pages 321–336. Springer, 2003.

9 Dieter Hofbauer and Johannes Waldmann. Termination of $\{aa\text{->}bc, bb\text{->}ac, cc\text{->}ab\}$. *Inf. Process. Lett.*, 98(4):156–158, 2006.

10 Dieter Hofbauer and Johannes Waldmann. Termination of string rewriting with matrix interpretations. In Frank Pfenning, editor, *Proc. 17th Conference on Rewriting Techniques and Applications (RTA)*, volume 4098 of *Lecture Notes in Comput. Sci.*, pages 328–342. Springer, 2006.

11 Adam Koprowski and Johannes Waldmann. Arctic termination ...below zero. In Andrei Voronkov, editor, *Proc. 19th Conference on Rewriting Techniques and Applications (RTA)*, volume 5117 of *Lecture Notes in Comput. Sci.*, pages 202–216. Springer, 2008.

12 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean termination tool 2. In Ralf Treinen, editor, *Proc. 20th Conference on Rewriting Techniques and Applications (RTA)*, volume 5595 of *Lecture Notes in Comput. Sci.*, pages 295–304. Springer, 2009.

13 David Sabel and Hans Zantema. Transforming cycle rewriting into string rewriting (extended version). `http://www.ki.informatik.uni-frankfurt.de/research/cycsrs/SZ15.pdf`, 2015.

14 The termination problem data base, 2015. `http://termination-portal.org/wiki/TPDB`.

15 Homepage of $\mathsf{T_TT_2}$, 2015. `http://cl-informatik.uibk.ac.at/software/ttt2/`.

16 Homepage of Yices, 2015. `http://yices.csl.sri.com/`.

17 Hans Zantema. Termination of term rewriting: Interpretation and type elimination. *J. Symb. Comput.*, 17(1):23–50, 1994.

18 Hans Zantema, Barbara König, and Harrie Jan Sander Bruggink. Termination of cycle rewriting. In Gilles Dowek, editor, *Proc. Joint 25th Conference on Rewriting Techniques and Applications and 12th Conference on Typed Lambda Calculi and Applications (RTATLCA)*, volume 8560 of *Lecture Notes in Comput. Sci.*, pages 476–490. Springer, 2014.