

Termination of String Rewriting Proved Automatically

H. Zantema (h.zantema@tue.nl)

*Department of Computer Science, TU Eindhoven
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands*

Abstract. In this paper it is described how a combination of polynomial interpretations, recursive path order, RFC match-bounds, the dependency pair method and semantic labelling can be used for automatically proving termination of an extensive class of string rewriting systems (SRSs). The tool implementing this combination of techniques is called TORPA: Termination of Rewriting Proved Automatically. All termination proofs generated by TORPA are easy to read and check, but for many of the SRSs involved finding a termination proof would be a hard job for a human. This paper contains all underlying theory, describes how the search for a termination proof is implemented, and includes many examples.

Keywords: String rewriting, termination, semantic labelling, relative termination, match-bounds

1. Introduction

In the last few decades many techniques have been developed for proving termination of rewriting. In the last years work in this area concentrates on proving termination automatically: the development of tools by which a rewrite system can be entered and by which fully automatically a termination proof is generated. A strong impulse in the development of these tools was given by the termination tools competition on the International Workshop on Termination in Valencia in June 2003. The intension is to have this competition every year; for the second time it was hold in May 2004.

In this paper we describe the tool TORPA: Termination of Rewriting Proved Automatically. This tool has been developed by the author. After having ideas in mind for years the actual implementation started in July 2003. Earlier versions of TORPA have been described in [24] (version 1.1) and in [25] (version 1.2). The present version is 1.3. There are many small SRSs from a wide variety of origins for which termination is proved fully automatically by TORPA within a fraction of a second. In the termination tools competition of May 2004 this version turned out to be the best tool in the category of string rewriting out of five participating tools.

TORPA only works on string rewriting systems (SRSs). On the one hand this is a strong restriction compared to general term rewriting. On the other hand string rewriting is a natural and widely accepted



© 2004 Kluwer Academic Publishers. Printed in the Netherlands.

paradigm with full computational power. For instance, Turing machine computation can be seen as a particular way of string rewriting. To illustrate the power of string rewriting, in Section 10 we present the well-known open problem known as the Collatz problem as the termination property of a surprisingly small SRS.

The main feature of TORPA is that an SRS is given as input and that TORPA generates a proof that this SRS is terminating or non-terminating. This proof is given in text. It is given in such a way that any human familiar with the basic techniques used in TORPA as they are described in this paper, can easily read and check the proof. The five basic techniques used for proving termination are

- polynomial interpretations ([17]),
- recursive path order ([5]),
- dependency pairs ([1]),
- RFC-match-bounds ([7]), and
- semantic labelling ([22]).

Polynomial interpretations, recursive path order and RFC-match-bounds are direct techniques to prove termination, while semantic labelling and dependency pairs are techniques for transforming an SRS to another one in such a way that termination of the original SRS can be concluded from (relative) termination of the transformed SRS. Originally, semantic labelling was the most significant transformation used in TORPA. For very small SRSs, in particular single rules, RFC-match-bounds provide the most powerful technique.

For proving non-termination only one technique is applied: search for looping reductions based on the analysis of the ancestor graph as introduced in [15, 8].

TORPA is applicable for proving *relative termination* rather than termination. The property of relative termination of an SRS R relative to an SRS S means that any reduction with respect to $R \cup S$ contains at most finitely many R -steps. If $S = \emptyset$ this coincides with termination of R . The reason that relative termination is included is twofold:

- There are applications in which the problem to be solved can naturally be expressed as relative termination, and not as termination. In particular here we think of liveness problems.
- The technique of dependency pairs can be seen as a technique by which termination of a rewrite system is proved via proving relative termination of a transformed rewrite system. In this way

in our implementation of the dependency pair method techniques for proving relative termination are used to prove plain termination of an SRS.

For all of the five basic termination techniques we do not use the full general version but only a particular case suitable for automation. On the other hand some of the techniques are extended to cover relative termination rather than termination. The power of TORPA is not only in the techniques themselves (most of them are well-known for several years), but in the way various instances of them are combined yielding both power and efficiency.

In this paper we describe and motivate the choices of the instances of the five techniques, and give many examples of proofs as they are generated by TORPA. For each of the techniques we give a self-contained exposure of the soundness including all proofs, except for recursive path order which plays only a minor role in TORPA. For RFC-match-bounds this is the hardest job. For each of the other three techniques we give the proof of correctness in no more than one page.

Other tools like AProVE [10], TTT [12, 13] and CiME [3] combine dependency pairs and path orders, too, and apply them much more involved than we do. They turn out to be the best tools for proving termination of term rewriting at the moment. However, applied to SRSs all of these tools are weaker than TORPA, as was shown in the competition of termination tools of the 7th International Workshop on Termination in May 2004.

A completely different approach is RFC-match-boundedness as introduced in [7, 9], and implemented in the tool Matchbox by Johannes Waldmann, see [20]. By this tool match-boundedness of SRSs can be proved automatically, either for all strings or for forward closures. In both cases termination of the SRS may be concluded. For many small SRSs, in particular for single rules and for SRSs not containing rules of the shape $ab \rightarrow ba$, this turns out to be remarkably strong. However, Matchbox only involves techniques related to match-boundedness, and for typically hard examples like $aabb \rightarrow bbbaaa$ TORPA is much more efficient than the Matchbox version from before January 2004 when the authors of [7] were informed about the heuristics implemented in TORPA. Recently our approach for RFC-match-boundedness was also implemented in AProVE, by which the score of AProVE in the 2004 termination competition for the category string rewriting was nearly as high as TORPA's score: 87 termination proofs for AProVE and 88 for TORPA.

The TORPA project started by implementing semantic labelling. It has been thought by several people that semantic labelling is unsuitable

for automatically finding termination proofs since a (quasi-)model has to be chosen. However, by restricting to (quasi-)models consisting of only two elements it turns out that the number of candidates for (quasi-)models is not too big, and trying several of these candidates is a fruitful approach. This approach to semantic labelling was the core of the first TORPA version: version 1.1 from October 2003. By absence of general heuristics for choosing a (quasi-)model the program often tries a number of randomly chosen attempts. It turns out that by trying typically 100 valid (quasi-)models this approach works surprisingly well. In the full program such a search in 100 randomly generated (quasi-)models is executed for several variants of the given SRS.

However, it turned out that for many small SRSs, in particular single rules, this labelling approach fails while the RFC match-bound approach succeeds. Therefore in January 2004 TORPA was extended by a version of the RFC match-bound technique, yielding version 1.2.

For the present version 1.3 from May 2004 some more improvements have been implemented. In particular, checking for loopingness has been added, and the order of computation has been rearranged. For instance, since checking for RFC match-boundedness is usually much faster than the search for semantic labelling, in version 1.3 checking for RFC match-boundedness is done before the search for semantic labelling.

Ignoring details the computation scheme of the present version 1.3 can be sketched as follows. Let **basic** denote the application of polynomial interpretations and recursive path order, both for the SRS and its reverse. Let **full label** denote consecutive application of

- 100 times **basic** after a randomly found model-labelling,
- 100 times **basic** after a randomly found model-labelling of the reversed SRS,
- 100 times **basic** after a randomly found quasi-model-labelling,
- 100 times **basic** after a randomly found quasi-model-labelling of the reversed SRS.

Then TORPA consecutively applies

- **basic**,
- **basic** on the dependency pair transformation,
- **basic** on the dependency pair transformation of the reversed SRS,
- check for RFC match-bounds,

- check for RFC match-bounds on the reversed SRS,
- check for loopingness,
- full label,
- full label on the dependency pair transformation,
- full label on the dependency pair transformation of the reversed SRS.

In this process execution stops if a termination proof or non-termination proof is found. It may start from the beginning if it has been detected that rules may be removed.

TORPA is freely available in two versions:

- A full version written in Delphi with a graphical user interface, including facilities for editing SRSs. The executable file of this version is available. This runs directly in a Windows environment without any installation. Detailed information about the input format is obtained by pushing the help button in this program.
- A plain version written in Pascal with a command line interface. Here the SRS is given as the input and the generated text of the termination proof is given as the output. The generated text is the same as in the Windows version, up to differences in the used random generator. From this version both the source code and a Linux executable is available.

Both versions accept the format of the Termination Problem Data Base (TPDB) as it is used for the termination tools competition. The Windows version also accepts a simpler format and allows some specific options.

Both versions can be downloaded from the TORPA homepage

<http://www.win.tue.nl/~hzantema/torpa.html>

The structure of this paper is as follows. First we give preliminaries of string rewriting and relative termination. Then in five consecutive sections we discuss each of the five basic termination techniques, followed by a section on how the techniques are combined and a section on detecting non-termination. Next the Collatz problem is discussed as an example showing the power of string rewriting, applying techniques developed in this paper and even applying TORPA in a proof. In the final section we give conclusions and discuss further research.

Throughout the paper various examples of TORPA output are given. To distinguish text generated by TORPA from the text of the paper the text generated by TORPA is always given in `typewriter font`.

2. Preliminaries

A string rewrite system (SRS) over an alphabet Σ is a set $R \subseteq \Sigma^+ \times \Sigma^*$. Elements $(\ell, r) \in R$ are called *rules* and are written as $\ell \rightarrow r$; ℓ is called the left hand side (lhs) and r is called the right hand side (rhs) of the rule. In TORPA format the arrow \rightarrow is written by the two symbols `->`. A string $s \in \Sigma^*$ rewrites to a string $t \in \Sigma^*$ with respect to an SRS R , written as $s \rightarrow_R t$ if strings $u, v \in \Sigma^*$ and a rule $\ell \rightarrow r \in R$ exist such that $s = ulv$ and $t = urv$.

An SRS R is called *terminating* if no infinite sequence t_1, t_2, t_3, \dots exists such that $t_i \rightarrow_R t_{i+1}$ for all $i = 1, 2, 3, \dots$. Termination is also called *strong normalization*; therefore the property of R being terminating is written as $\text{SN}(R)$. An SRS R is called *terminating* in a string s , notation $\text{SN}(s, R)$ if no infinite sequence t_1, t_2, t_3, \dots exists such that $s = t_1$ and $t_i \rightarrow_R t_{i+1}$ for all $i = 1, 2, 3, \dots$. We write $\infty(s, R)$ for $\neg \text{SN}(s, R)$.

An SRS R is called *terminating relative to* an SRS S , written as $\text{SN}(R/S)$, if no infinite sequence t_1, t_2, t_3, \dots exists such that

- $t_i \rightarrow_{R \cup S} t_{i+1}$ for all $i = 1, 2, 3, \dots$, and
- $t_i \rightarrow_R t_{i+1}$ for infinitely many values of i .

Sometimes the notation R/S is used for the rewrite relation $\rightarrow_S^* \cdot \rightarrow_R \cdot \rightarrow_S^*$; it is easy to see that $\text{SN}(R/S)$ coincides with termination of this rewrite relation. By definition $\text{SN}(R/S)$ and $\text{SN}(R/(S \setminus R))$ are equivalent. Therefore we will use the notation $\text{SN}(R/S)$ only for R and S being disjoint. In writing an SRS $R \cup S$ for which we want to prove $\text{SN}(R/S)$ we write the rules of R by $\ell \rightarrow r$ and the rules of S by $\ell \rightarrow = r$. In TORPA format the arrow $\rightarrow =$ is written by the three symbols `->=`. The rules from R are called *strict* rules; the rules from S are called *non-strict* rules.

First we prove a simple but very fruitful theorem for stepwise proving relative termination.

THEOREM 1. *Let R, S, R' and S' be SRSs satisfying*

- $R \cup S = R' \cup S'$ and $R \cap S = R' \cap S' = \emptyset$, and
- $\text{SN}(R'/S')$ and $\text{SN}((R \cap S')/(S \cap S'))$.

Then $\text{SN}(R/S)$.

Proof. Assume any infinite reduction with respect to $R \cup S = R' \cup S'$. Since $\text{SN}(R'/S')$ after finitely many steps this reduction only consists of S' -steps. Since $\text{SN}((R \cap S')/(S \cap S'))$ after again finitely many steps the remaining part consists only of $S \cap S'$ -steps, all being S -steps. Since $R \cap S = \emptyset$ we conclude that the assumed infinite reduction contains only finitely many R -steps. Hence we proved $\text{SN}(R/S)$. \square

Relative termination including a variant of Theorem 1 was already investigated by Alfons Geser in his PhD thesis in 1990, [6].

The way we will use Theorem 1 is as follows. If we have to prove $\text{SN}(R/S)$ then we try to split up $R \cup S$ into two disjoint parts R' and S' for which $R' \neq \emptyset$ and $\text{SN}(R'/S')$. If this succeeds then we may weaken the proof obligation $\text{SN}(R/S)$ to $\text{SN}((R \cap S')/(S \cap S'))$, i.e., all rules from R' may be removed. This process is repeated as long as it is applicable. If after a number of steps $R \cap S' = \emptyset$ then $\text{SN}((R \cap S')/(S \cap S'))$ trivially holds and the desired proof has been given.

As an example we consider the SRS consisting of the following three rules:

$$ab \rightarrow ba, bc \rightarrow cb, ca \rightarrow ac.$$

We want to prove termination of this SRS. Doing this in one step is not that easy, but proving $\text{SN}(R'/S')$ turns out to be very simple for R' consisting of the last rule and S' consisting of the first two rules, as we will see in the next section. Hence by Theorem 1 the last rule may be removed and it suffices to prove termination of the first two rules. In TORPA this will be done by another two applications of Theorem 1 each removing one rule. Then termination of the original SRS has been proved since no rules remain.

Next we give some basic observations on reversing strings. For a string s write s^{rev} for its reverse. For an SRS R write

$$R^{\text{rev}} = \{ \ell^{\text{rev}} \rightarrow r^{\text{rev}} \mid \ell \rightarrow r \in R \}.$$

LEMMA 2. *Let R and S be disjoint SRSs. Then $\text{SN}(R/S)$ if and only if $\text{SN}(R^{\text{rev}}/S^{\text{rev}})$.*

Proof. This follows from the observation that if $s \rightarrow_R t$ for any SRS R then $s^{\text{rev}} \rightarrow_{R^{\text{rev}}} t^{\text{rev}}$. \square

Lemma 2 is strongly used in TORPA: if $\text{SN}(R/S)$ has to be proved then all techniques are not only applied on R/S but also on $R^{\text{rev}}/S^{\text{rev}}$.

Another way of reversing is given as follows: for any SRS R define

$$R^{-1} = \{r \rightarrow \ell \mid \ell \rightarrow r \in R\}.$$

It is not difficult to prove the following lemma.

LEMMA 3. *Let R, S be finite SRSs such that for every rule $\ell \rightarrow r$ of $R \cup S$ the sizes of ℓ and r are equal. Then $\text{SN}(R/S)$ if and only if $\text{SN}(R^{-1}/S^{-1})$.*

However, the condition of all rules having both sides of equal length only holds exceptionally: we decided not to apply this lemma in TORPA.

3. Polynomial Interpretations

The ideas of (polynomial) interpretations go back to [18, 17, 2]. First we give the underlying theory for doing this for string rewriting.

Let A be a non-empty set and Σ be an alphabet. Let ϵ denote the empty string in Σ^* . If $f_a : A \rightarrow A$ has been defined for every $a \in \Sigma$ then $f_s : A \rightarrow A$ is defined for every $s \in \Sigma^*$ inductively as follows:

$$f_\epsilon(x) = x \text{ for every } x \in A,$$

$$f_{as}(x) = f_a(f_s(x)) \text{ for every } x \in A, a \in \Sigma, s \in \Sigma^*.$$

THEOREM 4. *Let A be a non-empty set and let $>$ be a well-founded order on A . Let $f_a : A \rightarrow A$ be strictly monotone for every $a \in \Sigma$, i.e., $f_a(x) > f_a(y)$ for every $x, y \in A$ satisfying $x > y$.*

Let R and S be disjoint SRSs over Σ such that $f_\ell(x) > f_r(x)$ for all $x \in A$ and $\ell \rightarrow r \in R$, and $f_\ell(x) \geq f_r(x)$ for all $x \in A$ and $\ell \rightarrow r \in S$. Then $\text{SN}(R/S)$.

Proof. Assume an infinite reduction $s_1 \rightarrow_{R \cup S} s_2 \rightarrow_{R \cup S} s_3 \rightarrow_{R \cup S} \dots$. Let i be any positive integer. Then $s_i = ulv$ and $s_{i+1} = urv$ for some $u, v \in \Sigma^*$ and $\ell \rightarrow r \in R \cup S$. Let $x \in A$ be arbitrary. Since all f_a are strictly monotone, the function $f_u : A \rightarrow A$ is strictly monotone too. Hence if $\ell \rightarrow r \in R$ we obtain

$$f_{s_i}(x) = f_u(f_\ell(f_v(x))) > f_u(f_r(f_v(x))) = f_{s_{i+1}}(x),$$

and if $\ell \rightarrow r \in S$ we obtain

$$f_{s_i}(x) = f_u(f_\ell(f_v(x))) \geq f_u(f_r(f_v(x))) = f_{s_{i+1}}(x).$$

Since $>$ is well-founded we conclude that the infinite inequality $f_{s_1}(x) \geq f_{s_2}(x) \geq f_{s_3}(x) \geq \dots$ contains only finitely many strict inequalities,

hence the infinite reduction $s_1 \rightarrow_{R \cup S} s_2 \rightarrow_{R \cup S} s_3 \rightarrow_{R \cup S} \dots$ contains only finitely many R -steps, which we had to prove. \square

In the general case this approach is called *monotone algebras* ([21, 23]). In case A consists of all integers $> N$ with the usual order for some number N , and the functions f_a are polynomials this approach is called *polynomial interpretations*.

Basically, in TORPA only three distinct polynomials are used:

- the identity,
- the successor $\lambda x \cdot x + 1$, and
- $\lambda x \cdot 10x$.

For every symbol a one of these three polynomials is chosen, and then it is checked whether this choice gives rise to $\text{SN}(R/S)$ for some non-empty R according to Theorem 4. If so, then by using Theorem 1 the proof obligation can be weakened and the process is repeated. As a first example consider the output given by TORPA when entering the single rule $ab \rightarrow ba$:

```
TORPA 1.3 is applied to the string rewriting system
a b -> b a

Choose polynomial interpretation:
a: lambda x.10x, rest lambda x.x+1
remove: a b -> b a
Terminating since no rules remain.
```

Here for f_a and f_b the function $\lambda x \cdot 10x$, respectively the successor function, are chosen. Since

$$f_{ab}(x) = f_a(f_b(x)) = 10(x + 1) > 10x + 1 = f_b(f_a(x)) = f_{ba}(x)$$

for every x indeed this single rule may be removed due to Theorem 4.

In forthcoming examples of TORPA output we will omit the first lines TORPA 1.3 is applied to...

As a similar example on relative termination rather than termination consider the two rules $ab \rightarrow ba, a \rightarrow ca$, i.e., $\text{SN}(R/S)$ has to be proved where R consists of the rule $ab \rightarrow ba$ and S consists of the rule $a \rightarrow ca$. Now TORPA yields:

```
Choose polynomial interpretation:
a: lambda x.10x, b: lambda x.x+1, rest identity
remove: a b -> b a
Relatively terminating since no strict rules remain.
```

As a next example consider the SRS mentioned in the previous section:

$$ab \rightarrow ba, bc \rightarrow cb, ca \rightarrow ac.$$

As already indicated in the previous section TORPA repeats applying polynomial interpretations, yielding the following termination proof:

```

Choose polynomial interpretation:
c: lambda x.10x, a: lambda x.x+1, rest identity
remove: c a -> a c
Choose polynomial interpretation:
a: lambda x.10x, b: lambda x.x+1, rest identity
remove: a b -> b a
Choose polynomial interpretation:
b: lambda x.10x, c: lambda x.x+1, rest identity
remove: b c -> c b
Terminating since no rules remain.

```

Now we discuss the way how these polynomial interpretations are implemented in TORPA.

Checking whether $f_\ell(x) > f_r(x)$ or $f_\ell(x) \geq f_r(x)$ for all x for some rule $\ell \rightarrow r$ is easily done: both $f_\ell(x)$ and $f_r(x)$ are of the shape $mx + k$ for $m > 0$ and $k \geq 0$. We observe that $mx + k > m'x + k'$ for all $x \in A$ for a suitable A if and only if $m > m' \vee (m = m' \wedge k > k')$. For instance, $10x > x + 19$ for all $x > 3$, hence we may choose A to consist of all integers > 3 . In all cases the suitable set A will consist of all integers $> N$ for some natural number N ; there is no need to compute N explicitly. The only thing to be done in checking whether $f_\ell(x) > f_r(x)$ for all x , for given f_a for every $a \in \Sigma$, is computing m, k, m', k' such that $f_\ell(x) = mx + k$ and $f_r(x) = m'x + k'$, and check $m > m' \vee (m = m' \wedge k > k')$. Checking whether $f_\ell(x) \geq f_r(x)$ is done similarly.

However, doing this for all possible interpretations would be too expensive. Let n be the number of symbols. Note that in combination with labelling and dependency pairs this can be up to 4 times the number of symbols occurring in the original SRS. If for all symbols all three choices for polynomials would be tried there would be 3^n attempts, being unacceptably big for reasonable size n . Therefore a selection among these 3^n combinations had to be made. In order to reach a polynomial bound on the number of attempts all relevant choices are made for which at least $n - 2$ symbols have the same interpretations. These are

- for respectively $n, 1, 2, n - 1$ or $n - 2$ symbols choose the successor, for the rest choose the identity;
- for one symbol choose $\lambda x \cdot 10x$, from the remaining $n - 1$ symbols choose for respectively $1, n - 2$ or all symbols the successor, for the rest choose the identity;
- for two symbols choose $\lambda x \cdot 10x$, for the rest choose the successor;

- for one or two symbols choose the successor, or for one the successor and for one the identity, and for the rest choose $\lambda x \cdot 10x$.

For the cases in which $\lambda x \cdot 10x$ is not involved essentially only a number of symbols is counted, and Lemma 2 is not applied since this would not increase the power. These cases essentially correspond to Lemma 20 in [11]. For the other two cases Lemma 2 is applied: the interpretations are checked for both the SRS itself and its reverse.

Combining $\lambda x \cdot 10x$ with the identity without using successor does not make sense since it can easily be shown that if an interpretation of this type applies then also an interpretation applies combining only successor and identity.

To show how Lemma 2 is applied consider the single rule $ab \rightarrow baa$. Then TORPA yields:

```
Reverse every lhs and rhs of the system and choose
polynomial interpretation:
b: lambda x.10x, a: lambda x.x+1
remove: a b -> b a a
Terminating since no rules remain.
```

Indeed, without reversing no polynomial interpretation applies. In [21] it was proved that even no interpretation in the natural numbers applies. But after reversing we obtain

$$f_{ba}(x) = f_b(f_a(x)) = 10x + 10 > 10x + 2 = f_a(f_b(x)) = f_{aab}(x)$$

for all x .

The choice of the number 10 in $\lambda x \cdot 10x$ is quite arbitrary. One can make the artificial rule $ab \rightarrow bbbbbbbba$ which can not be proved to be terminating by the present version of polynomial interpretations, and can be proved if $\lambda x \cdot 10x$ was replaced by $\lambda x \cdot Nx$ for any $N > 10$. However, this rule with 10 consecutive b -s is that artificial that we do not worry about this.

As a next example we consider the SRS consisting of the three rules

$$a \rightarrow bf, b \rightarrow c, cd \rightarrow dda.$$

Here TORPA yields:

```
Choose polynomial interpretation:
f: identity, d: lambda x.x+1, rest lambda x.10x
remove: c d -> d d a
Choose polynomial interpretation a: lambda x.x+1, rest identity
remove: a -> b f
Choose polynomial interpretation b: lambda x.x+1, rest identity
remove: b -> c
Terminating since no rules remain.
```

Finally, in case there are no more than three operation symbols some more polynomials are tried. For instance, on the four rules

$$aa \rightarrow b, c \rightarrow ab, bb \rightarrow c, c \rightarrow aaa$$

TORPA yields:

```

Choose polynomial interpretation
a: lambda x.x+1
b: lambda x.x+2
c: lambda x.x+3
remove: b b -> c
Choose polynomial interpretation   c: lambda x.x+1, rest identity
remove: c -> a b
remove: c -> a a a
Choose polynomial interpretation   a b c: lambda x.x+1
remove: a a -> b
Terminating since no rules remain.

```

4. Recursive Path Order

Recursive path order is already an old technique, too; it was introduced by Dershowitz [5]. Restricted to string rewriting it means that for a fixed order $>$ on the finite alphabet Σ , called the *precedence*, there is an order $>_{rpo}$ on Σ^* called *recursive path order*. The main property of this order is that if $\ell >_{rpo} r$ for all rules $\ell \rightarrow r$ of an SRS R , then R is terminating. This order $>_{rpo}$ has the following defining property: $s >_{rpo} t$ if and only if s can be written as $s = as'$ for $a \in \Sigma$, and either

- $s' = t$ or $s' >_{rpo} t$, or
- t can be written as $t = bt'$ for $b \in \Sigma$, and either
 - $a > b$ and $s >_{rpo} t'$, or
 - $a = b$ and $s' >_{rpo} t'$.

For further details we refer to [23]. The basic idea of proving termination of an SRS using recursive path order is that one starts with no restrictions on the order $>$, and then for every rule $\ell \rightarrow r$ one collects restrictions on $>$ by which one can conclude $\ell >_{rpo} r$ by the above defining property. If this succeeds for all rules without conflicting restrictions on $>$, then termination has been proved. Here restrictions on $>$ are conflicting if they violate transitivity and irreflexivity of $>$. For instance, by the only restriction $a > b$ which is not conflicting one easily derives $ab >_{rpo} bba$ from the above defining property, hence proving $\text{SN}(\{ab \rightarrow bba\})$.

However, search for non-conflicting restrictions may include a lot of branching, causing this process to be exponential. For a single term rewrite rule $\ell \rightarrow r$ it has been proved in [14] that checking whether a precedence $>$ exists satisfying $\ell >_{rpo} r$ is NP-complete. Here we restrict to string rewriting, but encounter similar problems on efficiency. Therefore in TORPA the alternative relation $>'_{rpo}$ is used instead of the full recursive path order, where $s >'_{rpo} t$ if and only if s can be written as $s = as'$ for $a \in \Sigma$, and either

- $s' \supseteq t$, or
- t can be written as $t = bt'$ for $b \in \Sigma$, and either
 - $a > b$ and $s >'_{rpo} t'$, or
 - $a = b$ and $s' >'_{rpo} t'$.

Here $s \supseteq t$ means that the string t can be obtained from s by removing zero or more symbols. It is easy to see that $s \supseteq t$ implies $s = t$ or $s >_{rpo} t$, hence $>'_{rpo} \subseteq >_{rpo}$.

If after application of polynomial interpretations $\text{SN}(R/S)$ has to be proved then TORPA tries to find a precedence $>$ such that $\ell >'_{rpo} r$ for all rules $\ell \rightarrow r \in R$, and $\ell >'_{rpo} r$ or $\ell = r$ for all rules $\ell \rightarrow r \in S$. For every rule this yields a number of restrictions of the shape $a > b$. Finally it is checked whether these restrictions violate transitivity and irreflexivity of $>$, i.e., it is checked whether the corresponding directed graph contains a cycle. If not, then $\text{SN}(R/S)$ has been proved. All of these steps can be done efficiently. For instance, on the single rewrite rule $abc \rightarrow bacb$ TORPA yields:

Terminating by recursive path order with precedence:
a>b b>c

Our version $>'_{rpo}$ is weaker than $>_{rpo}$. For instance, for $b > c > a$ we have $abc >_{rpo} cbaa$ while no precedence $>$ exists satisfying $abc >'_{rpo} cbaa$. However, in TORPA such a rule $abc \rightarrow cbaa$ is no problem since then polynomial interpretations apply. In fact we could not find any example for which full recursive path order would succeed while TORPA fails. Therefore the present efficient version $>'_{rpo}$ is kept as the basic version of recursive path order which is applied a number of times as follows.

For a symbol a and an SRS R write $\text{rem}_a(R)$ for the SRS obtained by removing all occurrences of a from all left hand sides and right hand sides of rules in R . It is easy to see that $\text{SN}(R/S)$ can be concluded from $\text{SN}(\text{rem}_a(R)/\text{rem}_a(S))$. If there are n symbols the version of recursive path order as described above is applied $2n+2$ times in TORPA as long

as no proof is found: for R/S and $R^{\text{rev}}/S^{\text{rev}}$, and for $\text{rem}_a(R)/\text{rem}_a(S)$ and $\text{rem}_a(R^{\text{rev}})/\text{rem}_a(S^{\text{rev}})$ for all symbols a . As an example, on the SRS consisting of the two rules

$$apbc \rightarrow caqdbapbap, \quad paqd \rightarrow daqdbapbap.$$

TORPA yields:

```
Terminating since rev(l) > rev(r) for all rules l -> r
where > is the recursive path order with precedence:
c>p c>a c>b c>q q>p q>a q>b, after removing d
```

Experience shows that most termination proofs generated by TORPA do not use recursive path order. One reason for this is that first always polynomial interpretations are tried, and polynomial interpretations are often successful for SRSs for which recursive path order would be successful too. Several extensions of recursive path order are possible, for instance using a precedence in which symbols can be equivalent. However, for the full general version of this extension we again would have efficiency problems; in combination with semantic labelling typically 1200 attempts for recursive path order proofs are done on SRSs having up to $4n$ distinct symbols if the original SRS has n distinct symbols. Moreover, many instances of this extension are already covered by polynomial interpretations. For instance, if the alphabet splits up in two equivalence classes $C_1 > C_2$ then usually also a polynomial interpretation can be found by choosing $f_a = \lambda x \cdot 10x$ for $a \in C_1$ and $f_a = \lambda x \cdot x + 1$ for $a \in C_2$.

5. Dependency Pairs

The technique of dependency pairs was introduced in [1] and is extremely useful for automatically proving termination of term rewriting. A recent exposition is given in [13]. Here we only use a mild version, without explicitly doing argument filtering or dependency graph approximation. It turns out that often the same reduction of the problem caused by these more involved parts of the dependency pair technique is done by applying our versions of labelling and polynomial interpretations. Since we want to present dependency pairs in the terminology of relative termination being quite different from the original presentation in [1], and we want to be self-contained, we present all required theory here. In our approach we do not even need the basic notion of *chains* of dependency pairs.

Let R be an SRS over an alphabet Σ . We assume that all lhs's of R are non-empty. For proving termination this is not a serious restriction since every SRS with an empty lhs is trivially non-terminating. Let Σ_D

be the set of *defined symbols* of R , i.e., the set of symbols occurring as the leftmost symbol of the lhs of a rule in R . For every defined symbol $a \in \Sigma_D$ we introduce a fresh symbol $a^\#$. In case a is a lowercase symbol then usually its capital version is used as the notation for $a^\#$. In TORPA this convention is followed; if a is not a lowercase symbol then instead the symbol is marked by $\#$.

Write $\Sigma^\# = \Sigma \cup \{a^\# \mid a \in \Sigma_D\}$. The SRS $DP(R)$ over $\Sigma^\#$ is defined to consist of all rules of the shape

$$a^\# \ell' \rightarrow b^\# r''$$

for which $a\ell' = \ell$ and $r = r'br''$ for some rule $\ell \rightarrow r$ in R and $a, b \in \Sigma_D$. Rules of $DP(R)$ are called *dependency pairs*. In our view the main theorem of dependency pairs states that $\text{SN}(R)$ if and only if $\text{SN}(DP(R)/R)$. It is used in TORPA as follows: if proving $\text{SN}(R)$ does not succeed by the earlier techniques, then these techniques are applied to trying to prove $\text{SN}(DP(R)/R)$. If this again does not succeed then finally they are applied to trying to prove $\text{SN}(DP(R^{\text{rev}})/R^{\text{rev}})$. In fact the desire for being able to do so was one of the main reasons to generalize the basic methods to relative termination and design TORPA to cover relative termination. Later on this way of using dependency pairs is also combined with semantic labelling.

As an example on dependency pairs consider the SRS consisting of the two rules $ab \rightarrow c, c \rightarrow ba$. Here TORPA yields

```

Dependency pair transformation:
  a b ->= c
  c ->= b a
  A b -> C
  C -> A
Choose polynomial interpretation
b c: lambda x.x+1, rest identity
remove: A b -> C
Choose polynomial interpretation
C: lambda x.x+1, rest identity
remove: C -> A
Relatively terminating since no strict rules remain.

```

In order to prove the main theorem we first need a lemma.

LEMMA 5. *Let R be an SRS in which all lhs's are non-empty. Let $s \in \Sigma^*$ satisfy $\infty(s, R)$. Then s can be written as $s = uav$ for $a \in \Sigma_D$, $u, v \in \Sigma^*$ such that $w, x \in \Sigma^*$ and $\ell \rightarrow r \in R$ exist such that $\text{SN}(v, R)$, $v \rightarrow_R^* w$, $aw = \ell x$ and $\infty(rx, R)$.*

Proof. Choose $s = uav$ for $a \in \Sigma$, $u, v \in \Sigma^*$ such that $\text{SN}(v, R)$ and $\infty(av, R)$. Then an infinite reduction of av should contain a reduction

step on its leftmost position. Let $\ell x \rightarrow_R rx$ be the first one. Then all desired properties hold. \square

THEOREM 6. *Let R be an SRS in which all lhs's are non-empty. Then $\text{SN}(R)$ if and only if $\text{SN}(DP(R)/R)$.*

Proof. For the if-part assume $\infty(s, R)$ for $s \in \Sigma^*$; we have to prove that there is an infinite $DP(R) \cup R$ -reduction containing infinitely many $DP(R)$ steps. Applying Lemma 5 to s yields $s = uav$ for which $\infty(av, R)$ and $\text{SN}(v, R)$. We will show that $a^\#v$ starts an infinite $DP(R) \cup R$ -reduction containing infinitely many $DP(R)$ steps. This follows from the following claim.

Claim: If $\infty(av, R)$ and $\text{SN}(v, R)$ then a', v' exist satisfying $\infty(a'v', R)$, $\text{SN}(v', R)$ and

$$a^\#v \rightarrow_R^* \cdot \rightarrow_{DP(R)} a'^\#v'.$$

We prove this claim by applying Lemma 5 on av . Since $\text{SN}(v, R)$ the resulting u is empty, and we obtain $w, x, \ell \rightarrow r$ such that $v \rightarrow_R^* w$, $aw = \ell x$ and $\infty(rx, R)$. Write $\ell = a\ell_0$. Next we apply Lemma 5 on rx , yielding $rx = u'a'v'$, $\text{SN}(v', R)$, $v' \rightarrow_R^* w'$, $a'w' = \ell'x'$, $\ell' \rightarrow r' \in R$ and $\infty(r'x', R)$. By $a'v' \rightarrow_R^* a'w' = \ell'x' \rightarrow_R r'x'$ we conclude $\infty(a'v', R)$. If $\#u' \geq \#r$ then x can be written as $x = u''a'v'$ implying an infinite reduction $v \rightarrow_R^* w = \ell_0x = \ell_0u''a'v' \rightarrow_R \dots$, contradicting $\text{SN}(v, R)$. Hence $\#u' < \#r$, by which we can write $r = u'a'r_0$. Since a' is the leftmost symbol of ℓ' we have $a' \in \Sigma_D$. Hence $a^\#\ell_0 \rightarrow a'^\#r_0$ is a rule from $DP(R)$. We conclude

$$a^\#v \rightarrow_R^* a^\#w = a^\#\ell_0x \rightarrow_{DP(R)} a'^\#r_0x = a'^\#v',$$

concluding the proofs of the claim and the if-part of the theorem.

For the only-if-part we assume $\text{SN}(R)$ and prove $\text{SN}(DP(R) \cup R)$, from which $\text{SN}(DP(R)/R)$ immediately follows. Due to type elimination / type introduction ([21]) $\text{SN}(DP(R) \cup R)$ is equivalent to termination of well-typed reductions, for the typing $a : s \rightarrow s$ for $a \in \Sigma$ and $a^\# : s \rightarrow t$ for $a \in \Sigma_D$. Reductions of type s are R -reductions that terminate by assumption. Assume an infinite reduction of type t exists, then every string in this reduction is of the shape $a^\#u$ for $a \in \Sigma_D, u \in \Sigma^*$. By removing over-lines and adding the string r' from the rule $al' \rightarrow r'br''$ when the rule $a^\#\ell' \rightarrow b^\#r''$ from $DP(R)$ is applied, this infinite $DP(R)$ -reduction transforms to an infinite R -reduction, contradicting $\text{SN}(R)$. \square

Note that for proving termination by dependency pairs only the if-part of Theorem 6 is used; the only-if-part is only included for completeness of the theory.

The standard dependency pair approach as proposed in [1] essentially coincides with the analysis of infinite well-typed reductions of type t , called *chains*.

It is a natural question whether dependency pairs can be used for relative termination rather than termination. More precisely, if we want to prove $\text{SN}(R/S)$ and all other methods fail then we would try to prove $\text{SN}(T_1(R, S)/T_2(R, S))$ for transformations T_1, T_2 for which we have the theorem

$$\text{SN}(T_1(R, S)/T_2(R, S)) \implies \text{SN}(R/S).$$

A first guess would be $T_1(R, S) = DP(R)$ and $T_2(R, S) = R \cup S \cup DP(S)$, where DP is defined with respect to the defined symbols of $R \cup S$. However, then the desired theorem does not hold. For instance, let R consist of the rule $a \rightarrow b$ and S of the rule $cb \rightarrow ca$. Then clearly $\text{SN}(R/S)$ does not hold, while $\text{SN}(DP(R)/\dots)$ holds by the trivial reason that $DP(R) = \emptyset$.

Instead a valid instance of the desired theorem is

$$\text{SN}(DP(R \cup S)/(R \cup S)) \implies \text{SN}(R/S).$$

This property immediately follows from Theorem 6 and the trivial implication $\text{SN}(R \cup S) \implies \text{SN}(R/S)$. This is applied in TORPA: if $\text{SN}(R/S)$ has to be proved and all other techniques fail then it is tried to prove $\text{SN}(DP(R \cup S)/(R \cup S))$. However, this may only succeed if $\text{SN}(R \cup S)$ holds, which is usually not the case. We leave as an open problem to find a non-trivial variant of the dependency pair transformation able to prove $\text{SN}(R/S)$ in case $\text{SN}(R \cup S)$ does not hold.

6. RFC-match-bounds

A recent very elegant and powerful approach for proving termination of string rewriting is given in [7, 9]: proving match-boundedness by means of compatible automata.

6.1. THEORY OF RFC-MATCH-BOUNDS

Here we present the theory as it is used in TORPA. The hardest part is the theorem on right hand sides of forward closures (RFC). Part of the theory, excluding this hard part, was presented in [9] from an automata theory point of view.

For an SRS R over an alphabet Σ we define the infinite SRS $\text{match}(R)$ over $\Sigma \times \mathbf{N}$ to consist of all rules $(a_1, n_1) \cdots (a_p, n_p) \rightarrow (b_1, m_1) \cdots (b_q, m_q)$ for which $a_1 \cdots a_p \rightarrow b_1 \cdots b_q \in R$ and $m_i = 1 + \min_{j=1, \dots, p} n_j$ for all $i = 1, \dots, q$.

So symbols are labelled by natural numbers, and $\text{match}(R)$ applies on a string of labelled if and only if R applies on the string obtained by removing all labels. In the result of the $\text{match}(R)$ -application in the newly created rhs all symbols are labelled by the successor of the smallest label occurring in the corresponding left hand side. As for dependency pairs this is only well-defined if the lhs's are non-empty. Again we assume that all lhs's are non-empty; again this is not a serious restriction since every SRS with an empty lhs is trivially non-terminating.

LEMMA 7. *Let R be a finite SRS and let $s = b_1 \cdots b_q$ be any string. Assume that some $N \in \mathbf{N}$ exists such that for every reduction*

$$(b_1, 0) \cdots (b_q, 0) \rightarrow_{\text{match}(R)}^* (c_1, n_1) \cdots (c_r, n_r)$$

it holds that $n_i \leq N$ for all $i = 1, \dots, r$. Then R is terminating in s .

Proof. Assume $s = b_1 \cdots b_q$ admits an infinite R -reduction. Then according to the definition of $\text{match}(R)$ this can be lifted to an infinite $\text{match}(R)$ -reduction starting in $(b_1, 0) \cdots (b_q, 0)$. Let m be a number such that for every rhs of R the length is less than m . Now for a symbol (a, k) define its weight $W((a, k)) = m^{N-k}$, and for a string of symbols the weight is the sum of the weights of the symbols. Due to the assumption all symbols (a, k) occurring in this infinite reduction satisfy $k \leq N$, hence $W((a, k)) \in \mathbf{N}$. For every rule $\ell \rightarrow r$ in $\text{match}(R)$ we have $r = (b_1, k), (b_2, k), \dots, (b_q, k)$ while ℓ contains a symbol $(a, k-1)$. Hence

$$W(\ell) \geq W((a, k-1)) = m^{N-k+1} > q \cdot m^{N-k} = W(r).$$

Hence in every step of the infinite $\text{match}(R)$ -reduction the weight in \mathbf{N} strictly decreases, contradiction. \square

The number N as it occurs in Lemma 7 is called a *match bound*.

The intended technique for proving termination is obtained by combining Lemma 7 with right hand sides of forward closures. A theorem of Dershowitz ([4]) states that a right-linear term rewrite system is terminating if and only if it is terminating in some restricted set of terms, namely the set of right hand sides of forward closures (RFC). Restricting to string rewriting here we avoid the technical definition of this set by describing its behavior by means of an auxiliary SRS $R_{\#}$ as was done in [7]. For an SRS R over an alphabet Σ we define the SRS $R_{\#}$ over $\Sigma \cup \{\#\}$ by

$$R_{\#} = R \cup \{ \ell_1 \# \rightarrow r \mid \ell \rightarrow r \in R \wedge \ell = \ell_1 \ell_2 \wedge \ell_1 \neq \epsilon \neq \ell_2 \}.$$

We will prove termination of R if $R_{\#}$ is terminating in $r\#^k$ for every rhs r of R and every $k \in \mathbf{N}$. The ideas of the proof go back to a proof sketch from [4]. Since we are not aware of a more detailed proof and our setting is quite different, we decided to include a full proof here. In order to do so we need some more notation and a few lemmas.

Define

$$R_b = \{ \ell_1\#\ell_2 \rightarrow r\# \mid \ell \rightarrow r \in R \wedge \ell = \ell_1\ell_2 \wedge \ell_1 \neq \epsilon \neq \ell_2 \}.$$

Let S consist of all strings containing exactly one symbol $\#$, i.e.,

$$S = \{ u\#v \mid u, v \in \Sigma^* \}.$$

On S we consider two relations $\xrightarrow{a}_R, \xrightarrow{n}_R$ describing active and non-active rewriting, respectively, defined by

$$\xrightarrow{a}_R = \{ (s, s') \in S \times S \mid s \rightarrow_{R_b} s' \} \cup \{ (u\#v, u'\#v) \mid u \rightarrow_R u' \},$$

$$\xrightarrow{n}_R = \{ (u\#v, u\#v') \mid v \rightarrow_R v' \}.$$

So active rewriting is rewriting overlapping the $\#$ -symbol or left from it, and non-active rewriting is rewriting right from the $\#$ -symbol.

The first lemma describes commutation between between active and non-active steps.

$$\text{LEMMA 8. } \xrightarrow{a}_R \cdot \xrightarrow{n}_R \subseteq \xrightarrow{n}_R \cdot \xrightarrow{a}_R.$$

Proof. Let $u\#v \xrightarrow{a}_R u'\#v' \xrightarrow{n}_R u''\#v''$. By definition of \xrightarrow{n}_R we have $u' = u''$ and $v' \rightarrow_R v''$. We consider the two cases for the \xrightarrow{a}_R -step:

Case 1: $u = u_0\ell_1, v = \ell_2v_0, u' = u_0r$ and $v' = v_0$ for some rule $\ell_1\ell_2 \rightarrow r$ in R . Then we have

$$u\#v = u_0\ell_1\#\ell_2v' \xrightarrow{n}_R u_0\ell_1\#\ell_2v'' \xrightarrow{a}_R u_0r\#v'' = u''\#v''.$$

Case 2: $u \rightarrow_R u'$ and $v = v'$. Then we have

$$u\#v \xrightarrow{n}_R u\#v'' \xrightarrow{a}_R u'\#v'' = u''\#v''.$$

□

We apply the notations ∞ and SN for relations \rightsquigarrow on the set S :

$$\infty(t, \rightsquigarrow) \iff \exists t_1, t_2, t_3, \dots \in S : t = t_1 \wedge \forall i : t_i \rightsquigarrow t_{i+1},$$

$$\text{SN}(t, \rightsquigarrow) \iff \neg(\infty(t, \rightsquigarrow)).$$

The following lemma describes how non-active steps can be filtered out from an infinite reduction.

LEMMA 9. *Let $s \in S$ satisfy $\text{SN}(s, \xrightarrow{n}_R)$ and $\infty(s, \xrightarrow{n}_R \cup \xrightarrow{a}_R)$. Then there exists $t \in S$ satisfying $s \xrightarrow{n}_R^* t$ and $\infty(t, \xrightarrow{a}_R)$.*

Proof. Write $P(s)$ for the property that we have to prove. Since $\text{SN}(s, \xrightarrow{n}_R)$ we may prove this by noetherian induction, i.e. for proving $P(s)$ we may assume the induction hypothesis $P(s')$ for $s \xrightarrow{n}_R s'$. Consider an infinite $\xrightarrow{n}_R \cup \xrightarrow{a}_R$ reduction starting in s . In case this reduction does not contain any \xrightarrow{n}_R step then we may choose $t = s$ and we are done. In the remaining case the reduction starts by $s \xrightarrow{a}_R^* \bar{s} \xrightarrow{n}_R \tilde{s}$ where $\infty(\tilde{s}, \xrightarrow{n}_R \cup \xrightarrow{a}_R)$. Using Lemma 8 one easily proves by induction that $\xrightarrow{a}_R^* \cdot \xrightarrow{n}_R \subseteq \xrightarrow{n}_R \cdot \xrightarrow{a}_R^*$, hence we obtain s' satisfying $s \xrightarrow{n}_R s' \xrightarrow{a}_R^* \tilde{s}$. Now clearly $\text{SN}(s', \xrightarrow{n}_R)$ and $\infty(s', \xrightarrow{n}_R \cup \xrightarrow{a}_R)$, hence by the induction hypothesis $P(s')$ we obtain $t \in S$ satisfying $s \xrightarrow{n}_R s' \xrightarrow{n}_R^* t$ and $\infty(t, \xrightarrow{a}_R)$. \square

Note that Lemma 9 and its proof hold for arbitrary relations \xrightarrow{n}_R and \xrightarrow{a}_R on an arbitrary set S satisfying $\xrightarrow{a}_R \cdot \xrightarrow{n}_R \subseteq \xrightarrow{n}_R \cdot \xrightarrow{a}_R$.

The next lemma relates \rightarrow_R and $\xrightarrow{n}_R \cup \xrightarrow{a}_R$.

LEMMA 10. *Let $u, v \in \Sigma^*$ and $\infty(uv, \rightarrow_R)$. Then*

$$\infty(u\#v, \xrightarrow{n}_R \cup \xrightarrow{a}_R).$$

Proof. From the definition of $\xrightarrow{n}_R \cup \xrightarrow{a}_R$ it follows that if $uv \rightarrow_R w$ then u', v' exist such that $w = u'v'$ and $u\#v \xrightarrow{n}_R \cup \xrightarrow{a}_R u'\#v'$. By repeating this an infinite \rightarrow_R -reduction starting in uv transforms to an infinite $\xrightarrow{n}_R \cup \xrightarrow{a}_R$ -reduction starting in $u\#v$. \square

LEMMA 11. *Let R be a non-terminating SRS not having an empty lhs. Then $\infty(r\#u, \xrightarrow{a}_R)$ for some rhs r of R and some $u \in \Sigma^*$.*

Proof. Choose $v \in \Sigma^*$ of minimal size satisfying $\infty(v, \rightarrow_R)$. Since there is no empty lhs the string v is non-empty. So $v = av'$ for $a \in \Sigma$ and $\text{SN}(v', \rightarrow_R)$. Since $\text{SN}(v', \rightarrow_R)$ an infinite reduction starting in v is of the shape

$$v = av' \xrightarrow{a}_R^* aw = \ell w' \rightarrow_R rw' \rightarrow_R \dots$$

for some rule $\ell \rightarrow r \in R$ and strings w, w' satisfying $v' \xrightarrow{a}_R^* w$. We have $\infty(rw', \rightarrow_R)$, and since $\text{SN}(v', \rightarrow_R)$, $v' \xrightarrow{a}_R^* w$ we have $\text{SN}(w, \rightarrow_R)$. Since w' is a postfix of w we have $\text{SN}(w', \rightarrow_R)$. Now we apply Lemma 9 to $s = r\#w'$. The conditions are fulfilled due to Lemma 10 and

the observation that $\text{SN}(s, \xrightarrow{R}_n)$ follows from $\text{SN}(w', \rightarrow_R)$. Hence there exists $t \in S$ satisfying $s \xrightarrow{R}_n^* t$ and $\infty(t, \xrightarrow{R}_a)$. Since $s = r\#w' \xrightarrow{R}_n^* t$ we can write $t = r\#u$, concluding the proof. \square

In a last lemma we relate $R_\#$ to \xrightarrow{R}_a .

LEMMA 12. *Let $u\#v \xrightarrow{R}_a u'\#v'$ for $u, u', v, v' \in \Sigma^*$. Then either*

- $u \rightarrow_R u'$ and $v = v'$, or
- $u\# \rightarrow_{R_\#} u'$ and $|v'| < |v|$.

Proof. Immediate from the definitions of \xrightarrow{R}_a and $R_\#$. \square

Now we are ready to prove the main RFC-theorem.

THEOREM 13. *Let R be an SRS over an alphabet Σ . Then R is terminating if and only if $R_\#$ terminates in $r\#^k$ for every rhs r of R and every $k \in \mathbf{N}$.*

Proof. The ‘only if’-part is easy: if $\rightarrow_{R_\#}$ admits an infinite reduction starting in $r\#^k$ then in every $R_\# \setminus R$ -step a $\#$ -symbol is removed which can be done only finitely many times. After these finitely many steps an infinite R reduction remains.

For the ‘if’-part assume R is non-terminating. If R contains an empty lhs then the theorem trivially holds. In the remaining case by Lemma 11 we have $\infty(r\#u, \xrightarrow{R}_a)$ for some rhs r of R and some $u \in \Sigma^*$. Write the corresponding infinite reduction as

$$r\#u = u_1\#v_1 \xrightarrow{R}_a u_2\#v_2 \xrightarrow{R}_a u_3\#v_3 \xrightarrow{R}_a \cdots$$

By Lemma 12 for every i we either have $u_i \rightarrow_R u_{i+1}$ and $v_i = v_{i+1}$, or $u_i\# \rightarrow_{R_\#} u_{i+1}$ and $|v_{i+1}| < |v_i|$. This latter case occurs only finitely often, say k times, due to finiteness of $|v_1|$. Since every \rightarrow_R -step is a $\rightarrow_{R_\#}$ step too, this gives rise to an infinite $\rightarrow_{R_\#}$ -reduction of $u_1\#^k = r\#^k$, contradicting the assumption that $R_\#$ terminates in $r\#^k$. \square

The following theorem is an immediate consequence of Lemma 7 and Theorem 13.

THEOREM 14. *Let R be an SRS and let $N \in \mathbf{N}$ such that for all rhs’s $b_1 \cdots b_q$ of R and all $k \in \mathbf{N}$ and all reductions*

$$(b_1, 0) \cdots (b_q, 0)(\#, 0)^k \xrightarrow{*}_{\text{match}(R_\#)} (c_1, n_1) \cdots (c_r, n_r)$$

it holds that $n_i \leq N$ for all $i = 1, \dots, r$. Then R is terminating.

The minimal number N satisfying the condition in Theorem 14 is called the corresponding *RFC-match-bound*.

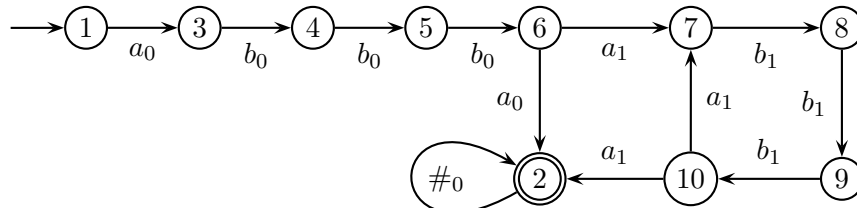
6.2. RFC-MATCH-BOUNDS IN TORPA

In TORPA termination of an SRS is proved by RFC-match-bounds by verifying the condition of Theorem 14. This is done by the construction of a finite automaton M over the alphabet $(\Sigma \cup \{\#\}) \times \mathbf{N}$, where Σ is the alphabet of R , satisfying:

- for every rhs $b_1 \cdots b_q$ of R and every $k \in \mathbf{N}$ the automaton M accepts $(b_1, 0) \cdots (b_q, 0)(\#, 0)^k$, and
- M is closed under $\text{match}(R_\#)$, i.e., if M accepts v and $v \rightarrow_{\text{match}(R_\#)} u$ then M accepts u too.

Such an automaton is called *compatible*¹. The pair $(a, k) \in (\Sigma \cup \{\#\}) \times \mathbf{N}$ will shortly be written as a_k , and the number k is called the *label* of this pair. It is easy to see that if a (finite) compatible automaton M has been found then for N being the biggest label occurring in M the condition of Theorem 14 holds. Hence the only thing to be done for proving termination by this approach is finding a compatible automaton. These observations were given before in [7]; the main new contribution of TORPA is the much more powerful heuristic of searching for such a compatible automaton.

As an example we consider the single rule $aba \rightarrow abbba$. TORPA proves termination by finding the following compatible automaton:



Rather than giving such a picture TORPA yields a list of all transitions, containing all information about the automaton:

¹ In [8] this kind of compatibility is considered more generally for a given language and SRS; here the language and the SRS is left implicit.

```

Terminating due to RFC match bound 1,
proved by the automaton defined by
initial state = 1, final state = 2, transitions:
from 1 to 3, label: a0
from 2 to 2, label: #0
from 3 to 4, label: b0
from 4 to 5, label: b0
from 5 to 6, label: b0
from 6 to 2, label: a0
from 6 to 7, label: a1
from 7 to 8, label: b1
from 8 to 9, label: b1
from 9 to 10, label: b1
from 10 to 2, label: a1
from 10 to 7, label: a1

```

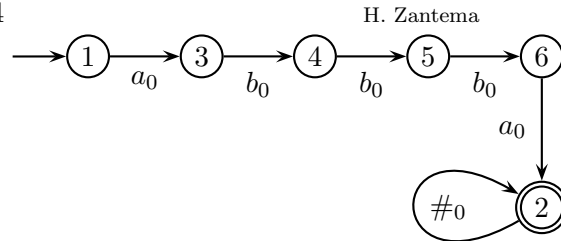
Indeed this automaton is compatible: it accepts $a_0b_0^3a_0\#_0^k$ for every k , and it is closed under $\text{match}(R_\#)$, hence proving termination. For instance, it accepts $a_0b_0^3a_0\#_0$ which rewrites to $a_0b_0^3a_1b_1^3a_1$ by the rule $a_0\#_0 \rightarrow a_1b_1^3a_1$ of $\text{match}(R_\#)$, also accepted by the automaton.

Now we describe how such a compatible automaton is constructed in TORPA. The construction starts by an initial automaton exactly accepting $(b_1, 0) \cdots (b_q, 0)(\#, 0)^k$ for every rhs $b_1 \cdots b_q$ of R and every $k \in \mathbf{N}$. The initial state is 1; the final state is 2. There is a $(\#, 0)$ -transition from 2 to 2, and for every rhs $b_1 \cdots b_q$ of R a path from 1 to 2 labelled by $(b_1, 0) \cdots (b_q, 0)(\#, 0)^k$ is constructed.

Then for every path in the automaton labelled by a lhs of $\text{match}(R_\#)$ it is checked whether there exists a path between the same two states labelled by the corresponding rhs. If not, the automaton is extended by such a path. Here the heuristic comes in. If a path from s_1 to s_2 has to be constructed labelled by the string au for $a \in (\Sigma \cup \{\#\}) \times \mathbf{N}$ and $u \in ((\Sigma \cup \{\#\}) \times \mathbf{N})^*$, then it is checked whether a state s exists for which a path from s to s_2 exists labelled by u . If so, then a transition from s_1 to s is added labelled by a , if not, then a completely fresh path from s_1 to s_2 is constructed labelled by au .

This process is repeated until either no transition need to be added any more or overflow occurs. In the first case the resulting automaton is compatible by construction, proving termination. Overflow occurs if the automaton contains 800 transitions.

In the above example the starting automaton is as follows:



Then states 7, 8, 9, 10 are added for making a path from 6 to 2 labelled by $a_1 b_1^3 a_1$. Then the transition from 10 to 7 is added for making a path from 10 to 2 labelled by $a_1 b_1^3 a_1$. Now nothing is to be added any more, so the desired automaton has been obtained.

This very simple heuristic was found after trying many other and more involved heuristics that turned out to be much less powerful.

Termination of the single rule $aabb \rightarrow bbbaaa$ is easily proved by this approach, yielding exactly the same automaton as given in [7] having 42 states and match bound 4. However, there it was found after an extensive process on intermediate automata of thousands of states, while in our approach no intermediate automata exceeding the final result occurred. The main difference is that in the approach of [7] an exact automaton is computed: the language accepted by the automaton is exactly the set of all $\text{match}(R_{\#})$ -reducts of strings of the shape $(b_1, 0) \cdots (b_q, 0)(\#, 0)^k$ for rhs's $b_1 \cdots b_q$. TORPA computes an approximation: the language accepted by the resulting automaton is a superset of the given set of reducts. However, for most examples we considered both resulting automata coincide. On the other hand, for many examples like the SRS consisting of the single rule $baaabbaa \rightarrow aaabbaaabb$ the exact approach as implemented in Matchbox ([20]) fails by reaching limits of resources, while TORPA succeeds within a fraction of a seconds. In this example the generated automaton has 155 states.

Our approach only works in case all rhs's are non-empty. In case there are empty rhs's the corresponding rules are expanded, i.e., rules of the shape $\ell \rightarrow \epsilon$ are replaced by $\ell a \rightarrow a$, for all $a \in \Sigma$. Obviously this does not change the termination behavior. For instance, applying TORPA on the SRS consisting of the four rules

$$Ab \rightarrow baBA, Ba \rightarrow abAB, Aa \rightarrow \epsilon, Bb \rightarrow \epsilon$$

yields


```

Expand empty rhs:
A b  -> b a B A
B a  -> a b A B
A a A -> A
A a b -> b
A a a -> a
A a B -> B
B b A -> A
B b b -> b
B b a -> a
B b B -> B

```

Reverse every lhs and rhs of the system.

Terminating due to RFC match bound 2,
proved by the automaton defined by

followed by the description of an automaton with 14 states.

Finally, if for a rule the lhs and rhs end in the same symbol, it may occur that termination can not be proved directly, but it can be proved after stripping this common last symbol. Intuitively, by this stripping one expects that proving termination will be harder, but there are examples for which this works, as was observed by Alfons Geser. For instance, applying TORPA on the single rule $ababaaa \rightarrow aaaababab$ yields

Reverse every lhs and rhs of the system.

```

Strip last symbol, yielding the rule
a a a b a b  -> b a b a b a a a

```

Terminating due to RFC match bound 1,
proved by the automaton defined by

followed by the description of an automaton with 16 states.

7. Semantic Labelling

The technique of semantic labelling was introduced in [22]. Here we restrict to the version for string rewriting in which every symbol is labelled by the value of its argument. For this version we will present the theory for relative termination. Then we will show how this theory is applied in TORPA for the case of (quasi-)models containing only two elements. This particular approach for automating semantic labelling grew out from [11]. In fact this was the starting point for TORPA, the other techniques came in later. For efficiency reasons in the present version 1.3 RFC-match-bounds are tried before semantic labelling. For many SRSs like the single rule $aa \rightarrow aba$ termination was proved by semantic labelling in versions 1.1 and 1.2, while in version 1.3 it is proved by RFC-match-bounds. On the other hand, for many other SRSs

RFC-match-bounds fail but semantic labelling is successful, by which semantic labelling is still a powerful ingredient of TORPA.

7.1. THEORY OF SEMANTIC LABELLING

Fix a non-empty set A and maps $f_a : A \rightarrow A$ for all $a \in \Sigma$ for some alphabet Σ . Let f_s for $s \in \Sigma^*$ be defined as before. Let $\overline{\Sigma}$ be the alphabet consisting of the symbols a_x for $a \in \Sigma$ and $x \in A$. The *labelling function* $\mathbf{lab} : \Sigma^* \times A \rightarrow \overline{\Sigma}^*$ is defined inductively as follows:

$$\mathbf{lab}(\epsilon, x) = \epsilon \quad \text{for } x \in A,$$

$$\mathbf{lab}(sa, x) = \mathbf{lab}(s, f_a(x))a_x \quad \text{for } s \in \Sigma^*, a \in \Sigma, x \in A.$$

For an SRS R over Σ define

$$\mathbf{lab}(R) = \{ \mathbf{lab}(l, x) \rightarrow \mathbf{lab}(r, x) \mid l \rightarrow r \in R, x \in A \}.$$

THEOREM 15. *Let R and S be two disjoint SRSs over an alphabet Σ . Let $>$ be a well-founded order on a non-empty set A . Let $f_a : A \rightarrow A$ be defined for all $a \in \Sigma$ such that*

- $f_a(x) \geq f_a(y)$ for all $a \in \Sigma, x, y \in A$ satisfying $x > y$, and
- $f_\ell(x) \geq f_r(x)$ for all $\ell \rightarrow r \in R \cup S, x \in A$.

Let Dec be the SRS over $\overline{\Sigma}$ consisting of the rules $a_x \rightarrow a_y$ for all $a \in \Sigma, x, y \in A$ satisfying $x > y$.

Then $\text{SN}(R/S)$ if and only if $\text{SN}(\mathbf{lab}(R)/(\mathbf{lab}(S) \cup \text{Dec}))$.

Proof. For the only-if-part assume $\text{SN}(R/S)$ and the existence of an infinite reduction of $\mathbf{lab}(R) \cup \mathbf{lab}(S) \cup \text{Dec}$ containing infinitely many $\mathbf{lab}(R)$ steps. Remove all subscripts in this reduction, then every $\mathbf{lab}(R)$ step transforms to an R -step, every $\mathbf{lab}(S)$ step transforms to an S -step, and every Dec -step transforms to equality, contradicting $\text{SN}(R/S)$.

For the converse we consider the following claim.

Claim: If $x \in A$ and $s \rightarrow_R t$ then $\mathbf{lab}(s, x) \rightarrow_{\mathbf{lab}(R)} \cdot \rightarrow_{\text{Dec}}^* \mathbf{lab}(t, x)$.

Choose $x \in A$ arbitrary. According to this claim applying $\mathbf{lab}(\cdot, x)$ to all strings in an infinite $R \cup S$ reduction containing infinitely many R steps gives rise to an infinite $\mathbf{lab}(R) \cup \mathbf{lab}(S) \cup \text{Dec}$ reduction containing infinitely many $\mathbf{lab}(R)$ steps, concluding the proof of the theorem. Hence it remains to prove the claim. In order to do so we first give two basic properties that we need:

- if $u, v \in \Sigma^*$ and $x \in A$ then $\mathbf{lab}(uv, x) = \mathbf{lab}(u, f_v(x))\mathbf{lab}(v, x)$, and

– if $u \in \Sigma^*$ and $x, y \in A$, $x \geq y$, then $\text{lab}(u, x) \rightarrow_{\text{Dec}}^* \text{lab}(u, y)$.

Both properties can be checked directly; for the last one the condition $f_a(x) \geq f_a(y)$ for $x > y$ is essential.

For $s \rightarrow_R t$ we can write $s = ulv$ and $t = urv$ for $u, v \in \Sigma^*$ and $\ell \rightarrow r \in R$. Using the above basic properties and the conditions of the theorem we obtain

$$\begin{aligned}
\text{lab}(s, x) &= \text{lab}(ulv, x) \\
&= \text{lab}(u, f_{\ell v}(x)) \text{lab}(\ell, f_v(x)) \text{lab}(v, x) \\
&\rightarrow_{\text{lab}(R)} \text{lab}(u, f_{\ell v}(x)) \text{lab}(r, f_v(x)) \text{lab}(v, x) \\
&= \text{lab}(u, f_{\ell}(f_v(x))) \text{lab}(r, f_v(x)) \text{lab}(v, x) \\
&\rightarrow_{\text{Dec}}^* \text{lab}(u, f_r(f_v(x))) \text{lab}(r, f_v(x)) \text{lab}(v, x) \\
&= \text{lab}(urv, x) \\
&= \text{lab}(t, x),
\end{aligned}$$

concluding the proof. \square

In case the relation $>$ is empty the set A together with the functions f_a for $a \in \Sigma$ is called a *model* for the SRS, otherwise it is called a *quasi-model*. It is called a model since then for every rule $\ell \rightarrow r$ the interpretation f_{ℓ} of ℓ is equal to the interpretation f_r of r . Note that $\text{Dec} = \emptyset$ in case of a model.

7.2. SEMANTIC LABELLING IN TORPA

The main application of Theorem 15 in TORPA is that if $\text{SN}(R/S)$ has to be proved then for a number of (quasi-)model candidates it is tried to prove $\text{SN}(\text{lab}(R)/(\text{lab}(S) \cup \text{Dec}))$. If this succeeds we are done. For instance, applying TORPA on the SRS consisting of the four rules $aal \rightarrow laa, raa \rightarrow aar, bl \rightarrow bar, rb \rightarrow lb$ may yield:

```

Apply labelling with the following interpretation in {0,1}:
a: lambda x.1-x
l: constant 1
r: constant 1
b: constant 1
and label every symbol by the value of its argument.

```

This interpretation is a model.

```

Labelled system:
a0 a1 l0 -> l0 a1 a0
a0 a1 l1 -> l1 a0 a1
r0 a1 a0 -> a0 a1 r0
r1 a0 a1 -> a0 a1 r1
b1 l0 -> b0 a1 r0
b1 l1 -> b0 a1 r1
r1 b0 -> l1 b0
r1 b1 -> l1 b1

```

followed by a termination proof for this labelled system by simple polynomial interpretations. In the notation of Theorem 15 this means that $A = \{0, 1\}$, $f_a(x) = 1 - x$ and $f_b(x) = f_l(x) = f_r(x) = 1$ for $x \in A$, $R = \{aal \rightarrow laa, raa \rightarrow aar, bl \rightarrow bar, rb \rightarrow lb\}$, $S = \text{lab}(S) = \text{Dec} = \emptyset$. Since $\text{lab}(aal, x) = a_0a_1l_x$ and $\text{lab}(laa, x) = l_xa_{1-x}a_x$ for $x = 0, 1$, the first two rules of the labelled system $\text{lab}(R)$ are as indicated, and the rest is obtained similarly. Hence the termination proof for R has been given by proving termination of $\text{lab}(R)$ by polynomial interpretations.

Now we describe how such a proof is found by TORPA. In TORPA Theorem 15 is only applied for the case where A consists of two elements, named 0 and 1. For every symbol a there are four possibilities for $f_a : A \rightarrow A$:

$$f_a = \lambda x \cdot x, \quad f_a = \lambda x \cdot 0, \quad f_a = \lambda x \cdot 1, \quad f_a = \lambda x \cdot 1 - x.$$

Up to renaming this set $A = \{0, 1\}$ admits only two strict orders $>$: the empty order and the order satisfying $1 > 0$. For the first one (the model case) for all symbols a all four interpretations for f_a are allowed, and the only restriction is that $f_\ell = f_r$ for all rules $\ell \rightarrow r \in R \cup S$. For the second order (the quasi-model case) for all symbols a only the first three interpretations for f_a are allowed, since $f_a = \lambda x \cdot 1 - x$ does not satisfy the requirement that $f_a(x) \geq f_a(y)$ for $x > y$. On the other hand, now the restriction on the rules is weaker: it should hold that $f_\ell(x) \geq f_r(x)$ for all rules $\ell \rightarrow r \in R \cup S$ and $x \in A$.

Disallowing $f_a = \lambda x \cdot 1 - x$ in the quasi-model case is essential for validity of Theorem 15. For instance, consider the SRS R consisting of the two rules $aba \rightarrow abba$ and $ba \rightarrow a$, and $S = \emptyset$. Clearly R is not terminating. By choosing $f_a = \lambda x \cdot 0$ and $f_b = \lambda x \cdot 1 - x$ we indeed have $f_\ell(x) \geq f_r(x)$ for both rules $\ell \rightarrow r$ and $x \in A$. However, $\text{SN}(\text{lab}(R)/\text{Dec})$ is easily proved by TORPA.

Now TORPA works as follows. First the model approach is tried. For SRSs R, S for which $\text{SN}(R/S)$ has to be proved, for all $a \in \Sigma$ the functions f_a are chosen randomly among the four possible functions until $f_\ell = f_r$ for all rules $\ell \rightarrow r \in R \cup S$. Since such a model always exists (for instance, $f_a = \lambda x \cdot x$ for all $a \in \Sigma$ always yields a model), such a model will always be found. Then for this choice $\text{lab}(R)$ and $\text{lab}(S)$ are computed, and it is tried to prove $\text{SN}(\text{lab}(R)/\text{lab}(S))$ by means of polynomial interpretations and recursive path order. If this succeeds the desired proof is generated, otherwise the whole procedure is repeated. There is a basic maximal number of attempts to be done. The default of this number is 100.

It can be the case that there are different solutions. For instance, in the example above we saw that choosing constant 1 for f_l , f_r and f_b , and $f_a = \lambda x \cdot 1 - x$ was successful. However, choosing constant 0 for

f_l , f_r and f_b , and $f_a = \lambda x \cdot 1 - x$ is successful too, just like a few other variants. Subsequent attempts to prove termination by TORPA may yield different solutions, due to the use of the random generator.

In case this first series of attempts was not yet successful a similar procedure is applied for quasi-models: now for all $a \in \Sigma$ the functions f_a are chosen randomly among the three allowed functions until $f_\ell(x) \geq f_r(x)$ for all rules $\ell \rightarrow r \in R \cup S$ and $x \in A$. Again such a quasi-model always exists and hence will always be found. Then it is tried to prove $\text{SN}(\text{lab}(R)/(\text{lab}(S) \cup \text{Dec}))$ by means of polynomial interpretations and recursive path order. Again this is repeated up to a number of times corresponding to the basic maximal number.

For both the model case and the quasi-model case everything is done twice as long as no solution is found: once for R/S and once for $R^{\text{rev}}/S^{\text{rev}}$. In case $S = \emptyset$ then this may done up to four more times for dependency pair transformations: for $DP(R)/R$, $DP(R^{\text{rev}})/R^{\text{rev}}$, $DP(R)^{\text{rev}}/R^{\text{rev}}$ and $DP(R^{\text{rev}})^{\text{rev}}/R$.

As an example, we consider the SRS consisting of the four rules

$$a \rightarrow bc, \quad ab \rightarrow ba, \quad dc \rightarrow da, \quad ac \rightarrow ca.$$

Now the output of TORPA starts by

```
Reverse every lhs and rhs of the system.

Apply labelling with the following interpretation in {0,1}:
a: identity
b: constant 0
c: identity
d: constant 1
and label every symbol by the value of its argument.

This is a quasi-model for 1 > 0.

Labelled system:
a0 -> c0 b0
a1 -> c0 b1
b0 a0 -> a0 b0
b1 a1 -> a0 b1
c1 d0 -> a1 d0
c1 d1 -> a1 d1
c0 a0 -> a0 c0
c1 a1 -> a1 c1
a1 ->= a0
b1 ->= b0
c1 ->= c0
d1 ->= d0
```

and ends by a standard termination proof by polynomial interpretations of this labelled system. We are not aware of any other technique by which termination of this SRS can be proved.

7.3. REMOVING LABELS

It can be the case that some attempt to prove $\text{SN}(R/S)$ by means of proving $\text{SN}(\text{lab}(R)/(\text{lab}(S) \cup \text{Dec}))$ fails, but applying polynomial interpretations succeeds in removing some rules. If the remaining rules are all contained in $\text{lab}(R') \cup \text{lab}(S') \cup \text{Dec}$ for $R' \subseteq R$ and $S' \subseteq S$, at least one of the inclusions being strict, then by applying Theorem 15 in the reverse direction it suffices to prove $\text{SN}(R'/S')$. In fact this is removal of labels. In the default setting of TORPA in trying to apply labelling it is checked every time whether this removal of labels can be applied. If so, then all labels are removed and TORPA starts from scratch trying to prove $\text{SN}(R'/S')$. For instance, applying TORPA to the SRS consisting of the three rules

$$aba \rightarrow abba, \quad bab \rightarrow baab, \quad bbbb \rightarrow bb$$

may yield

```

Apply labelling with the following interpretation in {0,1}:
  a: constant 0
  b: lambda x.1-x
and label every symbol by the value of its argument.

This interpretation is a model.

Labelled system:
  a1 b0 a0 -> a0 b1 b0 a0
  a1 b0 a1 -> a0 b1 b0 a1
  b0 a1 b0 -> b0 a0 a1 b0
  b0 a0 b1 -> b0 a0 a0 b1
  b1 b0 b1 b0 -> b1 b0
  b0 b1 b0 b1 -> b0 b1

Choose polynomial interpretation  a1 : lambda x.x+1,  rest identity
remove: a1 b0 a0 -> a0 b1 b0 a0
remove: a1 b0 a1 -> a0 b1 b0 a1
Choose polynomial interpretation  b1 : lambda x.x+1,  rest identity
remove: b1 b0 b1 b0 -> b1 b0
remove: b0 b1 b0 b1 -> b0 b1
Remaining rules:
  b0 a1 b0 -> b0 a0 a1 b0
  b0 a0 b1 -> b0 a0 a0 b1

Remove all labels, remaining unlabelled system:
  b a b -> b a a b

```

```
Apply labelling with the following interpretation in {0,1}:
a: lambda x.1-x
b: constant 1
and label every symbol by the value of its argument.
```

This interpretation is a model.

Labelled system:

```
b0 a1 b0 -> b1 a0 a1 b0
b0 a1 b1 -> b1 a0 a1 b1
```

```
Choose polynomial interpretation  b0 : lambda x.x+1, rest identity
remove: b0 a1 b0 -> b1 a0 a1 b0
remove: b0 a1 b1 -> b1 a0 a1 b1
Terminating since no rules remain.
```

Without removal of labels a termination proof could not have been given using the techniques described until now. In the Windows version of TORPA you may choose whether this removal of labels should be tried or not by a check box that will be visible when pushing the button ‘customize’.

If a termination proof can be found without the option of removal of labels then a termination proof can be found too with the option. This is due to the fact that both polynomial interpretations and recursive path order are robust under removal of rules. Therefore the option of removal of labels is the default. However, there are examples by which the full termination proof is longer and therefore harder to verify if the option is on. For instance, this holds for the SRS

$$\begin{array}{lll}
 tf \rightarrow tcn & nf \rightarrow fn & of \rightarrow fo \\
 ns \rightarrow fs & os \rightarrow fs & cf \rightarrow fc \\
 cn \rightarrow nc & co \rightarrow oc & co \rightarrow o.
 \end{array}$$

This is essentially the rewrite system as it appears in the first case study of liveness in [11]: termination of this rewrite system implies that all old processes in a waiting line will eventually be served. Surprisingly, by putting the option off for removal of labels, TORPA yields exactly the same labelling proof as was given in [11] guarded by some heuristics.

8. Combined proofs

Combining the basic termination techniques TORPA may typically find termination proofs of the following shape:

- First zero or more rules are removed by polynomial interpretations.
- Then a labelling is applied followed by repeated polynomial interpretations. After removing labels this step may be repeated a number of times.

- Then the dependency pair transformation is applied on the SRS or its reverse.
- Then again a labelling is applied followed by repeated polynomial interpretations. After removing labels this step may be repeated a number of times.
- Finally the termination proof of the original SRS is completed if after repeated polynomial interpretations no strict rules remain.

As an example we consider the SRS consisting of the five rules

$$\begin{aligned}
 f0 &\rightarrow s0 \\
 d0 &\rightarrow 0 \\
 ds &\rightarrow ssdps \\
 fs &\rightarrow dfps \\
 ps &\rightarrow \epsilon
 \end{aligned}$$

in which s , d and f describe successor, doubling and the exponential function, respectively. For this SRS TORPA proceeds as follows. First the first rule is removed by choosing a polynomial interpretation. Then a labelling is found, by which after removing rules by polynomial interpretations and next removing all labels only the last three rules of the original system remain. Next, the dependency pair transformation is applied on the reversed system of these three rules. For this system a quasi-model labelling is found: a system consisting of 20 rules over 12 symbols. After removing rules by polynomial interpretations and again removing all labels, a system is found for which the ultimate termination proof is generated by finding a labelling for the third time. Finding the full proof requires one or two seconds.

In termination proofs found by TORPA combining labelling and dependency pairs always the dependency pair transformation is applied first. It is also valid to do it the other way around: first apply labelling and then do the dependency pair transformation. The implementation of TORPA was extended in such a way that the dependency pair transformation was applied on the labelled systems. However, we did not find any example for which a termination proof was found using this extension while it was not found by the version without this extension. Therefore this extension has been removed from the present version.

Another option for combining techniques is combining labelling and RFC-match-bounds. This option has not yet been investigated.

9. Detecting non-termination

In case the search for a termination proof for an SRS R is not successful, TORPA tries to find a proof for non-termination. This is done by generating a directed graph with labelled nodes in the following way. In the initial graph for every lhs ℓ of R there is a node labelled by ℓ , and there are no edges. Then the following procedure is applied a number of times:

for every rule $\ell \rightarrow r$ of R and every node n of the graph let u be the label of n , and do

- for every r_1, r_2, u' satisfying $r = r_1 r_2$, $u = r_2 u'$ and $r_1 \neq \epsilon \neq r_2$ do $\text{add}(\ell u', n)$,
- for every r_1, r_2, u' satisfying $r = r_1 r_2$, $u = u' r_1$ and $r_1 \neq \epsilon \neq r_2$ do $\text{add}(u' \ell, n)$,
- for every r_1, r_2 satisfying $r = r_1 u r_2$ and $r_1 \neq \epsilon \neq r_2$ do $\text{add}(\ell, n)$,
- for every r_1, r_2 satisfying $u = r_1 r r_2$ do $\text{add}(r_1 \ell r_2, n)$.

Here for a string v and a node n execution of $\text{add}(v, n)$ means that it is checked whether a node exists in the graph labelled by v . If so, then an edge from this node to n is added. If not, then a new node labelled by v is added, together with an edge from this new node to n .

Execution of this procedure is repeated until no more edges are added or some size upper bound has been reached. Essentially the graph obtained is a fragment of the ancestor graph as described in [15, 8]. The graph is constructed in such a way that if there is an edge from a node labelled by u to a node labelled by u' , then $u \rightarrow_R u_1 u' u_2$ for (possibly empty) strings u_1, u_2 . As a consequence, if there is a non-empty path from a node labelled by u to a node labelled by u' , then $u \rightarrow_R^+ u_1 u' u_2$ for strings u_1, u_2 . In particular, if the graph admits a cycle then an infinite reduction of the following shape exists:

$$u \rightarrow_R^+ v u w \rightarrow_R^+ v v u w w \rightarrow_R^+ v v v u w w w \rightarrow_R^+ \dots$$

Such a reduction is called *looping*, see [26]. So in TORPA it is checked whether this graph is cyclic, and if so then non-termination is concluded.

As an example we apply TORPA on the single rule $ab \rightarrow bbaa$:

Non-terminating; looping reduction starting in: aab

Indeed, for $u = aab$ a looping reduction of the above shape is found in which $v = bb$ and $w = aa$:

$$u = aab \rightarrow abbaa \rightarrow bbaabaa = vuw.$$

This approach is not complete; for instance, in [26] a simple non-terminating SRS of two rules is given not admitting a looping reduction.

10. Collatz problem

In this section we will show how a well-known open problem can be coded as the termination problem of a SRS consisting of 7 small rules over 5 symbols. The reason to include this in this paper is twofold: on the one hand it illustrates the power of termination of small SRSs. On the other hand we show how techniques developed in this paper are helpful in the proof that our SRS coding is indeed correct: we even use TORPA in this proof of correctness.

The Collatz problem is called after L. Collatz who posed this problem in 1937; it is also called Syracuse problem or $3n + 1$ problem. The problem is whether for every positive natural number a_0 there exists n such that $a_n = 1$, where

$$a_n = \begin{cases} a_{n-1}/2 & \text{if } a_{n-1} \text{ is even} \\ 3a_{n-1} + 1 & \text{if } a_{n-1} \text{ is odd} \end{cases}$$

for $n = 1, 2, 3, \dots$. It is conjectured that this holds for every positive natural number a_0 ; it has been proved in Mathematica that it holds for all positive natural numbers $< 10^{15}$ ([19]). An overview on this open problem is given in [16]; a recent annotated bibliography by the same author is found on <http://arxiv.org/abs/math.NT/0309224>. At a first glance one may expect that a full programming framework including integer arithmetic is required to express this conjecture. However, this is not true. Let C be the SRS consisting of the following seven rules:

$$\begin{array}{lll} h11 \rightarrow 1h & 11hb \rightarrow 11sb & h1b \rightarrow t11b \\ & 1s \rightarrow s1 & 1t \rightarrow t111 \\ & bs \rightarrow bh & bt \rightarrow bh \end{array}$$

over the five symbols b (blank), h (half), s (shift), t (triple) and 1. This SRS can be seen as a kind of a Turing machine with an elastic tape: the tape alphabet consists of 1 and b where b is the blank symbol, and h , s and t are the machine states. By h the head is shifted to the right while contracting two tape cells to one; by s and t the head is shifted to the left while by t every cell is tripled.

The main goal of this section is to prove that the above conjecture is equivalent to termination of C . We do not make progress in proving the conjecture (we agree with Paul Erdős that “mathematics is not yet ready for such problems”), we only prove this equivalence. As expected, TORPA is not able to prove termination of C .

THEOREM 16. *The SRS C is terminating if and only if for every positive natural number a_0 there exists n such that $a_n = 1$.*

Proof. With respect to C for $n > 1$ we have

$$bh1^{2n}b \rightarrow^* b1^n hb \rightarrow b1^n sb \rightarrow^* bs1^n b \rightarrow bh1^n b,$$

and for $n \geq 0$ we have

$$bh1^{2n+1}b \rightarrow^* b1^n h1b \rightarrow b1^n t11b \rightarrow^* bt1^{3n+2}b \rightarrow bh1^{3n+2}b.$$

This latter reduction describes two steps in the sequence: if $a_k = 2n + 1$ then $a_{k+1} = 3(2n + 1) + 1 = 6n + 4$ and $a_{k+2} = 3n + 2$. Hence if the number 1 does not occur in the infinite sequence a_0, a_1, a_2, \dots then also 2 and 4 do not occur, and by the above reduction patterns we obtain an infinite C -reduction starting from the string $bh1^{a_0}b$. This proves the ‘only if’-part of the theorem.

For the ‘if’-part we assume that the SRS admits an infinite reduction; we will show that then a number a_0 exists such that the infinite sequence a_0, a_1, a_2, \dots does not contain the number 1, concluding the proof the theorem.

By adding symbols b in front and behind the first string in the infinite reduction, this first string is of the shape $bu_1bu_2b \cdots bu_kb$ where u_1, \dots, u_k are strings not containing b . Due to the shape of the rules the only way such a string can be rewritten is that u_i is replaced by u'_i for some i satisfying $bu_ib \rightarrow bu'_ib$, and u_j is unchanged for all $j \neq i$. Since the number of b -s never changes during the infinite reduction and every reduction step takes place in one of the finite number of k positions, for some i there is an infinite reduction starting in the string bu_ib . Omitting the index i now we have an infinite reduction starting in the string bub where u is a string not containing the symbol b . Now we will prove that u contains exactly one shift symbol, where the symbols h, s, t are called shift symbols. If u does not contain shift symbols then bub is a normal form not admitting an infinite reduction.

Assume u contains two or more shift symbols. In the elastic Turing machine interpretation this can be seen as a configuration with more than one head on the same tape. We will show that then bub does

not admit an infinite reduction. In the style of TORPA we choose a labelling in $\{0, 1\}$: we choose

$$f_1(x) = x, \quad f_b(x) = 0, \quad f_h(x) = f_s(x) = f_t(x) = 1$$

for $x \in \{0, 1\}$. Clearly the model requirement $f_\ell = f_r$ holds for all rules $\ell \rightarrow r$. In our argument labelling of 1 and b does not play a role, hence we will only label h, s, t . From the shape of the rules we see that the number of shift symbols does not change during rewriting, hence this number remains always two or more in the reduction of bub . Since f_h, f_s and f_t are all constant 1, all labels left from the rightmost shift symbol will be 1. Hence in the labelled version of the infinite C -reduction of bub the rules $bs_i \rightarrow bh_i$ and $bt_i \rightarrow bh_i$ are only applied for $i = 1$. So we obtain an infinite reduction of the labelled system

$$\begin{array}{lll} h_0 1 1 & \rightarrow & 1 h_0 & 1 1 h_0 b & \rightarrow & 1 1 s_0 b & h_0 1 b & \rightarrow & t_0 1 1 b \\ h_1 1 1 & \rightarrow & 1 h_1 & 1 s_0 & \rightarrow & s_0 1 & 1 t_0 & \rightarrow & t_0 1 1 1 \\ & & & 1 s_1 & \rightarrow & s_1 1 & 1 t_1 & \rightarrow & t_1 1 1 1 \\ & & & b s_1 & \rightarrow & b h_1 & b t_1 & \rightarrow & b h_1 \end{array}$$

This yields a contradiction since this SRS is terminating as is easily proved by TORPA only using polynomial interpretations:

```
Choose polynomial interpretation  h0: lambda x.x+1,  rest identity
remove: 1 1 h0 b -> 1 1 s0 b
remove: h0 1 b -> t0 1 1 b
Choose polynomial interpretation  s1: lambda x.x+1,  rest identity
remove: b s1 -> b h1
Choose polynomial interpretation  t1: lambda x.x+1,  rest identity
remove: b t1 -> b h1
Reverse every lhs and rhs of the system and choose polynomial
interpretation:  t0 and t1: lambda x.10x,  rest lambda x.x+1
remove: h0 1 1 -> 1 h0
remove: h1 1 1 -> 1 h1
remove: 1 t0 -> t0 1 1 1
remove: 1 t1 -> t1 1 1 1
Choose polynomial interpretation:
1: lambda x.10x,  rest lambda x.x+1
remove: 1 s0 -> s0 1
remove: 1 s1 -> s1 1
Terminating since no rules remain.
```

As a consequence, we conclude that u contains exactly one shift symbol. Hence $bub = b1^k \alpha 1^m b$ for some $\alpha \in \{h, s, t\}$ and natural numbers k, m . If $\alpha = h$ then we have $bh1^{2k+m}b \rightarrow^* b1^k h 1^m b = bub$. If $\alpha = s$ then $bub = b1^k s 1^m b \rightarrow^* bh1^{k+m}b$ is the only reduction of length $k+1$ of bub , hence $bh1^{k+m}b$ occurs in the infinite reduction of bub . If $\alpha = t$ then $bub = b1^k t 1^m b \rightarrow^* bh1^{3k+m}b$ is the only reduction of length $k+1$ of bub , hence $bh1^{3k+m}b$ occurs in the infinite reduction of bub . In all cases we found an infinite reduction of a term of the shape $bh1^n b$ for some n .

Now choose $a_0 = n$. Note that reducing $bh1^n b$ is deterministic: in every term at most one redex occurs. Since $bh1^{2k}b \rightarrow^* bh1^k b$ for every $k > 1$, $bh1^{2k}b$ has no infinite reduction for $k \leq 1$, and $bh1^{2k+1}b \rightarrow^* bh1^{3k+2}b$ for every $k \geq 0$, the infinite reduction of $bh1^n b$ gives rise to an infinite sequence a_0, a_1, a_2, \dots not containing 2, and hence not containing 1. \square

The SRS C admits many variations for which Theorem 16 can be proved too. There are versions over more than five symbols for which a simpler proof can be given, but we preferred to keep C as simple as possible: 7 rules over 5 symbols, and every lhs and rhs has length at most 4. We did not succeed in improving any of these three restrictions while keeping the other two.

11. Conclusions and Further Research

TORPA is able to find termination proofs fully automatically for many SRSs. Often a generated proof is of a shape that it is very unlikely that it was ever found by a human. This is not only due to restrictions in the implementation of TORPA: there are many small SRSs for which TORPA automatically finds a termination proof, but for which finding any human proof allowing any presently known technique seems to be a really hard job. On the other hand, the generated proofs are usually not more than a few pages of text, including many details. For people being familiar with the underlying theory as explained in this paper, verifying the generated proofs is always straightforward, at most it may be boring. In particular, for proofs using RFC-match-bounds it may include checking for closedness under rewriting for automata with hundreds of states.

Except for the systematically used transformation of reversing all lhs's and rhs's all techniques used in TORPA also apply to term rewriting rather than string rewriting; only for RFC-match-bounds linearity conditions have to be required. A possible follow up will be a version of TORPA capable of proving termination of term rewriting. In particular, focusing on semantic labelling it may be investigated whether the TORPA approach can deal with term rewriting systems for which tools like AProVE and TTT fail.

For string rewriting, future improvements of TORPA are expected by more involved ways of combining various techniques. In particular, this may be obtained if all of the basic termination techniques can deal with relative termination.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. A. Ben-Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9:137–159, 1987.
3. E. Contejean, C. Marché, B. Monate, and X. Urbain. The CiME rewrite tool. Available at <http://cime.lri.fr/>.
4. N. Dershowitz. Termination of linear rewriting systems. In S. Even and O. Kariv, editors, *Proc. 8th Int. Coll. on Automata, Languages and Programming (ICALP-81)*, volume 115 of *Lecture Notes in Computer Science*, pages 448–458. Springer, 1981.
5. N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
6. A. Geser. *Relative Termination*. PhD thesis, Universität Passau, Germany, 1990.
7. A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting systems. In B. Rován and P. Vojtas, editors, *Proc. 28th Int. Symp. Mathematical Foundations of Computer Science (MFCS-03)*, volume 2747 of *Lecture Notes in Computer Science*, pages 449–459. Springer, 2003. Extended version accepted for publication in *Appl. Algebra Engrg. Comm. Comput.*
8. A. Geser, D. Hofbauer, and J. Waldmann. Deciding termination for ancestor match-bounded string rewriting systems. Nia technical report, National Institute of Aerospace, Hampton, VA, 2004. Available at research.nianet.org/~geser/papers/nia-ancestors.html.
9. A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. Finding finite automata that certify termination of string rewriting. In K. Salomaa and S. Yu, editors, *Proceedings of Ninth International Conference on Implementation and Application of Automata (CIAA04)*, *Lecture Notes in Computer Science*. Springer, 2004.
10. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In V. van Oostrom, editor, *Proceedings of the 15th Conference on Rewriting Techniques and Applications (RTA)*, volume 3091 of *Lecture Notes in Computer Science*, pages 210–220. Springer, 2004. Tool available at <http://www-i2.informatik.rwth-aachen.de/AProVE/>.
11. J. Giesl and H. Zantema. Liveness in rewriting. In R. Nieuwenhuis, editor, *Proceedings of the 14th Conference on Rewriting Techniques and Applications (RTA)*, volume 2706 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2003.
12. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In R. Nieuwenhuis, editor, *Proceedings of the 14th Conference on Rewriting Techniques and Applications (RTA)*, volume 2706 of *Lecture Notes in Computer Science*, pages 311–320, 2003.
13. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In V. van Oostrom, editor, *Proceedings of the 15th Conference on Rewriting Techniques and Applications (RTA)*, volume 3091 of *Lecture Notes in Computer Science*, pages 249–268. Springer, 2004.
14. M. S. Krishnamoorthy and P. Narendran. On recursive path ordering. *Theoretical Computer Science*, 40:323–328, 1985.

15. W. Kurth. *Termination und konfluenz von semi-Thue-systemen mit nur einer regel*. PhD thesis, Technische Universität Clausthal, Germany, 1990.
16. J. C. Lagarias. The $3n + 1$ problem and its generalizations. *Amer. Math. Monthly*, 92:3–23, 1985.
17. D.S. Lankford. On proving term rewriting systems are noetherian. Technical report MTP 3, Louisiana Technical University, 1979.
18. Z. Manna and S. Ness. On the termination of Markov algorithms. In *Proceedings of the Third Hawaii International Conference on System Science*, pages 789–792, Honolulu, HI, 1970.
19. I. Vardi. *Computational recreations in Mathematica*. Addison-Wesley, 1991.
20. J. Waldmann. Matchbox: a tool for match-bounded string rewriting. In V. van Oostrom, editor, *Proceedings of the 15th Conference on Rewriting Techniques and Applications (RTA)*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94. Springer, 2004.
21. H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.
22. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.
23. H. Zantema. Termination. In *Term Rewriting Systems, by Terese*, pages 181–259. Cambridge University Press, 2003.
24. H. Zantema. Termination of string rewriting proved automatically. Technical Report CS-report 03-14, Eindhoven University of Technology, 2003. Available via http://www.win.tue.nl/inf/onderzoek/en_index.html .
25. H. Zantema. TORPA: termination of rewriting proved automatically. In V. van Oostrom, editor, *Proceedings of the 15th Conference on Rewriting Techniques and Applications (RTA)*, volume 3091 of *Lecture Notes in Computer Science*, pages 95–104. Springer, 2004.
26. H. Zantema and A. Geser. Non-looping string rewriting. *Theoret. Informatics Appl.*, 33:279–301, 1999.

