RWTH Aachen

Department of Computer Science
Technical Report

Liveness in Rewriting

Jürgen Giesl and Hans Zantema

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2002-11

RWTH Aachen · Department of Computer Science · Mar. 2003 (revised version)

The publications of the Department of Computer Science of RWTH Aachen (*Aachen University of Technology*) are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

Liveness in Rewriting*

Jürgen Giesl[1] and Hans Zantema[2]

[1] LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany, giesl@informatik.rwth-aachen.de

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

Abstract. In this paper, we show how the problem of verifying liveness properties is related to termination of term rewrite systems (TRSs). We formalize liveness in the framework of rewriting and present a sound and complete transformation to transform particular liveness problems into TRSs. Then the transformed TRS terminates if and only if the original liveness property holds. This shows that liveness and termination are essentially equivalent. To apply our approach in practice, we introduce a simpler sound transformation which only satisfies the 'only if'-part. By refining existing techniques for proving termination of TRSs we show how liveness properties can be verified automatically. As examples, we prove a liveness property of a waiting line protocol for a network of processes and a liveness property of a protocol on a ring of processes.

1 Introduction

Usually, *liveness* is roughly defined as: "*something will eventually happen*" [1] and it is often remarked that "*termination is a particular case of liveness*". In this paper we present liveness in a general but precise setting and study the relationship between liveness and termination. To this end, we use the framework of abstract reduction and term rewrite systems. Classically, TRSs are applied to model evaluation in programming languages. Our aim is to use rewrite systems to also study liveness questions which are of high importance in practice (e.g., in protocol verification for distributed processes). In particular, we show how to verify liveness properties by existing termination techniques for TRSs.

In Sect. 2 we define a suitable notion of liveness to express eventuality properties using abstract reduction. Then in Sect. 3 this notion is specialized to the framework of term rewriting and we show how liveness can be modelled as a rewrite relation. Afterwards we discuss how actual liveness properties can be verified. To this end, in Sect. 4 we investigate the connection between a particular kind of liveness and termination. More precisely, we present a sound and complete transformation which allows us to express liveness problems as termination problems of ordinary TRSs. This shows that techniques for proving termination of TRSs can also be used to infer liveness properties. In order to apply this approach in practice, based on our preceding results we present a sound (but incomplete) technique to perform termination proofs for liveness properties in Sect. 5, which is significantly easier to mechanize. In contrast to model checking and related methods, our technique does not require finiteness of the state

* Extended version of a paper from the *Proceedings of the 14th Int. Conference on Rewriting Techniques and Applications (RTA-03)*, Valencia, Spain, LNCS, Springer-Verlag, 2003.

space. Our approach differs from other applications of term rewriting techniques to parameterized systems or infinite state spaces, where the emphasis is on verification of other properties like reachability [4]. We demonstrate our approach on case studies of networks with shared resources and on a (token) ring protocol in Sect. 6.

2 Liveness in Abstract Reduction

In this section we give a formal definition of liveness using the framework of abstract reduction. We assume a set S of states and a notion of computation that can be expressed by a binary relation $\to \subseteq S \times S$. So "$t \to u$" means that a computation step from t to u is possible. A *computation sequence* or *reduction* is defined to be a finite sequence $t_1, t_2, \ldots, t_n$ or an infinite sequence $t_1, t_2, t_3, \ldots$ with $t_i \to t_{i+1}$. We write $\to^*$ for the reflexive transitive closure of $\to$, i.e., $\to^*$ represents zero or more computation steps.

To define liveness we assume a set $G \subseteq S$ of 'good' states and a set $I \subseteq S$ of initial states. A reduction is *maximal* if it is either infinite or if its last element is in the set of *normal forms* $\mathsf{NF} = \{t \in S \mid \neg\exists u : t \to u\}$. The liveness property $\mathsf{Live}(I, \to, G)$ holds if every maximal reduction starting in I contains an element of G. Thus, our notion of liveness describes eventuality properties (i.e., it does not capture properties like starvation freedom which are related to fairness).

Definition 1 (Liveness) *Let S be a set of states, $\to \subseteq S \times S$, and $G, I \subseteq S$. Let "$t_1, t_2, t_3, \ldots$" denote an infinite sequence of states. Then $\mathsf{Live}(I, \to, G)$ holds iff*

1. $\forall t_1, t_2, t_3, \ldots : (t_1 \in I \wedge \forall i : t_i \to t_{i+1}) \Rightarrow \exists i : t_i \in G$, *and*
2. $\forall t_1, t_2, \ldots, t_n : (t_1 \in I \wedge t_n \in \mathsf{NF} \wedge \forall i : t_i \to t_{i+1}) \Rightarrow \exists i : t_i \in G$.

For example, *termination* (or *strong normalization* $\mathsf{SN}(I, \to)$) is a special liveness property describing the non-existence of infinite reductions, i.e.,

$$\mathsf{SN}(I, \to) \;=\; \neg(\exists t_1, t_2, t_3, \ldots :\; t_1 \in I \wedge \forall i : t_i \to t_{i+1}).$$

Theorem 2 *The property $\mathsf{SN}(I, \to)$ holds if and only if $\mathsf{Live}(I, \to, \mathsf{NF})$ holds.*

Proof. For the 'if'-part, if $\mathsf{SN}(I, \to)$ does not hold, then there is an infinite reduction $t_1 \to t_2 \to \cdots$ with $t_1 \in I$. Due to NF's definition, this infinite reduction does not contain elements of NF, contradicting Property 1 in Def. 1.

Conversely, if $\mathsf{SN}(I, \to)$ holds, then Property 1 in the definition of $\mathsf{Live}(I, \to, \mathsf{NF})$ holds trivially. Property 2 also holds, since $G = \mathsf{NF}$. □

Thm. 2 states that termination is a special case of liveness. The next theorem proves a kind of converse. For that purpose, we restrict the computation relation $\to$ such that it may only proceed if the current state is not in G.

Definition 3 **($\to_G$)** *Let S, $\to$, G be as in Def. 1. Then $\to_G \subseteq S \times S$ is the relation where $t \to_G u$ holds if and only if $t \to u$ and $t \notin G$.*

Now we show that $\mathsf{Live}(I, \to, G)$ is equivalent to $\mathsf{SN}(I, \to_G)$. The 'only if'-part holds without any further conditions. However, for the 'if'-part we have to demand that G contains all normal forms $\mathsf{NF}(I)$ reachable from I, where $\mathsf{NF}(I) = \{u \in \mathsf{NF} \mid \exists t \in I : t \to^* u\}$. Otherwise, if there is a terminating sequence $t_1 \to \ldots \to t_n$ with all $t_i \notin G$, we might have $\mathsf{SN}(I, \to_G)$ but not $\mathsf{Live}(I, \to, G)$.

Theorem 4 *Let $\mathsf{NF}(I) \subseteq G$. Then $\mathsf{Live}(I, \to, G)$ holds iff $\mathsf{SN}(I, \to_G)$ holds.*

Proof. For the 'if'-part assume $\mathsf{SN}(I, \to_G)$. Property 2 of Def. 1 holds since $\mathsf{NF}(I) \subseteq G$. If Property 1 does not hold then there is an infinite reduction without elements of G starting in I, contradicting $\mathsf{SN}(I, \to_G)$.

Conversely assume that $\mathsf{Live}(I, \to, G)$ holds and that $\mathsf{SN}(I, \to_G)$ does not hold. Then there is an infinite sequence $t_1, t_2, \ldots$ with $t_1 \in I \wedge \forall i : t_i \to_G t_{i+1}$. Hence, $t_i \notin G$ and $t_i \to t_{i+1}$ for all i, contradicting Property 1 in Def. 1. $\square$

Thm. 4 allows us to verify actual liveness properties: if $\mathsf{NF}(I) \subseteq G$, then one can instead verify termination of $\to_G$. If $\mathsf{NF}(I) \not\subseteq G$, then $\mathsf{SN}(I, \to_G)$ still implies the liveness property for all infinite computations. In Sect. 4 and 5 we show how techniques to prove termination of TRSs can be used for termination of $\to_G$.

3 Liveness in Term Rewriting

Now we focus on liveness in rewriting, i.e., we study the property $\mathsf{Live}(I, \to_R, G)$ where $\to_R$ is the rewrite relation corresponding to a TRS R. For an introduction to term rewriting, the reader is referred to [3], for example.

Let Σ be a signature containing at least one constant and let $\mathcal{V}$ be a set of variables. We write $\mathcal{T}(\Sigma, \mathcal{V})$ for the set of terms over Σ and $\mathcal{V}$ and $\mathcal{T}(\Sigma)$ is the set of ground terms. For a term t, $\mathcal{V}(t)$ and $\Sigma(t)$ denote the variables and function symbols occurring in t. Now $\mathcal{T}(\Sigma, \mathcal{V})$ represents computation states and $G \subseteq \mathcal{T}(\Sigma, \mathcal{V})$.

By Thm. 4, $\mathsf{Live}(I, \to, G)$ is equivalent to $\mathsf{SN}(I, \to_G)$, if $\mathsf{NF}(I) \subseteq G$. In order to verify liveness, we want to prove $\mathsf{SN}(I, \to_G)$ by approaches for termination proofs of ordinary TRSs. However, depending on the form of G, different techniques are required. Therefore, in the remainder we restrict ourselves to sets G of the following form:

$$G \;=\; \{t \mid t \text{ does not contain an instance of } p\} \qquad \text{for some term } p.$$

In other words, G contains all terms which cannot be written as $C[p\sigma]$ for any context C and substitution σ. As before, $t \to_G u$ holds iff $t \to_R u$ and $t \notin G$. So a term t may be reduced whenever it contains an instance of the term p.

A typical example of a liveness property is that eventually all processes requesting a resource are granted access to the resource (see Sect. 6.1 and 6.4). If a process waiting for the resource is represented by the unary function symbol old and if terms are used to denote the state of the whole network, then we would define $G = \{t \mid t \text{ does not contain an instance of } \mathsf{old}(x)\}$. Now $\mathsf{Live}(I, \to_R, G)$ means that eventually one reaches a term without the symbol old.

However, for arbitrary terms and TRSs, the notion $\to_G$ is not very useful: if there is a symbol f of arity > 1 or if p contains a variable x (i.e., if p can be written as $C[x]$ for some context C), then termination of $\to_G$ implies termination of the full rewrite relation $\to_R$. The reason is that any infinite reduction $t_1 \to_R t_2 \to_R \cdots$ gives rise to an infinite reduction $f(t_1, p, \ldots) \to_R f(t_2, p, \ldots) \to_R \cdots$ or $C[t_1] \to_R C[t_2] \to_R \cdots$ where in both cases none of the terms is in G. Therefore we concentrate on the particular case of *top rewrite systems* in which there is a designated symbol top. (These TRSs can be regarded as special forms of *typed* rewrite systems [10].)

Definition 5 (Top Rewrite System) *Let Σ be a signature and let $\mathsf{top} \notin \Sigma$ be a new unary function symbol. A term $t \in \mathcal{T}(\Sigma \cup \{\mathsf{top}\}, \mathcal{V})$ is a* top term *if its root is top and top does not occur below the root. Let $\mathcal{T}_{\mathsf{top}}$ denote the set of all ground top terms. A TRS R over the signature $\Sigma \cup \{\mathsf{top}\}$ is a* top rewrite system *iff for all rules $l \to r \in R$ either*

- *l and r are top terms (in this case, we speak of a* top rule*) or*
- *l and r do not contain the symbol top (then we have a* non-top rule*)*

Top rewrite systems typically suffice to model networks of processes, since the whole network is represented by a top term [6]. Clearly, in top rewrite systems, top terms can only be reduced to top terms again. In such systems we consider liveness properties $\mathsf{Live}(\mathcal{T}_{\mathsf{top}}, \to_R, G)$. So we want to prove that every maximal reduction of ground top terms contains a term without an instance of p.

Example 6 (Simple liveness example) Consider the following two-rule TRS R.

$$\mathsf{top}(\mathsf{c}) \to \mathsf{top}(\mathsf{c}) \qquad\qquad \mathsf{f}(x) \to x$$

Clearly, R is not terminating and we even have infinite reductions within $\mathcal{T}_{\mathsf{top}}$:

$$\mathsf{top}(\mathsf{f}(\mathsf{f}(\mathsf{c}))) \to_R \mathsf{top}(\mathsf{f}(\mathsf{c})) \to_R \mathsf{top}(\mathsf{c}) \to_R \mathsf{top}(\mathsf{c}) \to_R \ldots$$

However, in every reduction one eventually reaches a term without f. Hence, if $p = \mathsf{f}(x)$, then the liveness property is fulfilled for all ground top terms. Note that for $\Sigma = \{\mathsf{c}, \mathsf{f}\}$, we have $\mathsf{NF}(\mathcal{T}_{\mathsf{top}}) = \varnothing$ and thus, $\mathsf{NF}(\mathcal{T}_{\mathsf{top}}) \subseteq G$. Hence, by Thm. 4 it is sufficient to verify that $\to_G$ is terminating on $\mathcal{T}_{\mathsf{top}}$. Indeed, the above reduction is not possible with $\to_G$, since $\mathsf{top}(\mathsf{c})$ is a normal form w.r.t. $\to_G$.

4 Liveness and Termination

In this section we investigate the correspondence between liveness and termination in the framework of term rewriting. As in the previous section, we consider liveness properties $\mathsf{Live}(\mathcal{T}_{\mathsf{top}}, \rightarrow_R, G)$ for top rewrite systems R where G consists of those terms that do not contain instances of some subterm p. Provided that $\mathsf{NF}(\mathcal{T}_{\mathsf{top}}) \subseteq G$, by Thm. 4 the liveness property is equivalent to $\mathsf{SN}(\mathcal{T}_{\mathsf{top}}, \rightarrow_G)$.

Our aim is to prove termination of $\rightarrow_G$ on $\mathcal{T}_{\mathsf{top}}$ by means of termination of TRSs. In this way one can use all existing techniques for termination proofs of term rewrite systems (including future developments) in order to prove liveness properties. A first step into this direction was taken in [6], where the termination proof technique of *dependency pairs* was used to verify certain liveness properties of telecommunication processes. However, now our aim is to develop an approach to connect liveness and termination in general.

Given a TRS R and a term p, we define a TRS $L(R, p)$ such that $L(R, p)$ terminates (on all terms) if and only if $\mathsf{SN}(\mathcal{T}_{\mathsf{top}}, \rightarrow_G)$. A transformation where the 'only if'-direction holds is called *sound* and if the 'if'-direction holds, it is called *complete*. The existence of the sound and complete transformation $L(R, p)$ shows that for rewrite relations, liveness and termination are essentially equivalent.

The construction of $L(R, p)$ is motivated by an existing transformation [7, 8] which was developed for a completely different purpose (termination of context-sensitive rewriting). We introduce a number of new function symbols resulting in an extended signature Σ_G. Here, $\mathsf{proper}(t)$ checks whether t is a ground term over the original signature Σ (Lemma 9) and $\mathsf{match}(p, t)$ checks in addition whether p matches t (Lemma 10). In this case, $\mathsf{proper}(t)$ and $\mathsf{match}(p, t)$ reduce to $\mathsf{ok}(t)$. To ease the formulation of the match-rules, we restrict ourselves to *linear* terms p, i.e., a variable occurs at most once in p. Moreover, for every variable x in p we introduce a fresh constant denoted by the corresponding upper-case letter X. We write $\overline{p}$ for the ground term obtained by replacing every variable in p by its corresponding fresh constant and in this way, it suffices to handle ground terms $\overline{p}$ in the match-rules. The new symbol check investigates whether its argument is a ground term over Σ which contains an instance of p (Lemma 11). In this case, $\mathsf{check}(t)$ reduces to $\mathsf{found}(t)$ and to find the instance of p, check may be propagated downwards through the term until one reaches the instance of p.

Finally, $\mathsf{active}(t)$ denotes that t may be reduced, since it contains an instance of p. Therefore, active may be propagated downwards to any desired redex of the term. After the reduction step, active is replaced by mark which is then propagated upwards to the top of the term. Now one checks whether the resulting term still contains an instance of p and none of the newly introduced function symbols. To this end, mark is replaced by check. If an instance of p is found, check is turned into found and found is propagated to the top of the term where it is replaced by active again. The TRS $L(R, p)$ has been designed in such a way that infinite reductions are only possible if this process is repeated infinitely often and Lemmata 12–14 investigate $L(R, p)$'s behavior formally.

Definition 7 **($L(R, p)$)** *Let R be a top rewrite system over $\Sigma \cup \{\mathsf{top}\}$ with $\mathsf{top} \notin \Sigma$ and let $p \in \mathcal{T}(\Sigma, \mathcal{V})$ be linear. The TRS $L(R, p)$ over the signature $\Sigma_G = \Sigma \cup \{\mathsf{top}, \mathsf{match}, \mathsf{active}, \mathsf{mark}, \mathsf{check}, \mathsf{proper}, \mathsf{start}, \mathsf{found}, \mathsf{ok}\} \cup \{X \mid x \in \mathcal{V}(p)\}$ consists* *of the following rules for all non-top rules $l \to r \in R$, all top rules* $\mathsf{top}(t) \to \mathsf{top}(u) \in R$, *all* $f \in \Sigma$ *of arity* $n > 0$ *and* $1 \leq i \leq n$, *and all constants* $c \in \Sigma_G$*:*

$$\begin{aligned}
\mathsf{active}(l) &\to \mathsf{mark}(r) \\
\mathsf{top}(\mathsf{active}(t)) &\to \mathsf{top}(\mathsf{mark}(u)) \\
\mathsf{top}(\mathsf{mark}(x)) &\to \mathsf{top}(\mathsf{check}(x)) \\
\mathsf{check}(f(x_1, .., x_n)) &\to f(\mathsf{proper}(x_1), .., \mathsf{check}(x_i), .., \mathsf{proper}(x_n)) \\
\mathsf{check}(x) &\to \mathsf{start}(\mathsf{match}(\overline{p}, x))
\end{aligned} \tag{1}$$

$$\begin{aligned}
\mathsf{match}(f(x_1, .., x_n), f(y_1, .., y_n)) &\to f(\mathsf{match}(x_1, y_1), .., \mathsf{match}(x_n, y_n)), && \textit{if } f \in \Sigma(p) \\
\mathsf{match}(c, c) &\to \mathsf{ok}(c), && \textit{if } c \in \Sigma(p) \\
\mathsf{match}(c, x) &\to \mathsf{proper}(x), && \textit{if } c \notin \Sigma \textit{ and } c \in \Sigma(\overline{p}) \\
\mathsf{proper}(c) &\to \mathsf{ok}(c), && \textit{if } c \in \Sigma \\
\mathsf{proper}(f(x_1, \ldots, x_n)) &\to f(\mathsf{proper}(x_1), \ldots, \mathsf{proper}(x_n)) \\
f(\mathsf{ok}(x_1), \ldots, \mathsf{ok}(x_n)) &\to \mathsf{ok}(f(x_1, \ldots, x_n)) \\
\mathsf{start}(\mathsf{ok}(x)) &\to \mathsf{found}(x) \\
f(\mathsf{ok}(x_1), .., \mathsf{found}(x_i), .., \mathsf{ok}(x_n)) &\to \mathsf{found}(f(x_1, .., x_n)) \\
\mathsf{top}(\mathsf{found}(x)) &\to \mathsf{top}(\mathsf{active}(x)) \\
\mathsf{active}(f(x_1, \ldots, x_i, \ldots, x_n)) &\to f(x_1, \ldots, \mathsf{active}(x_i), \ldots, x_n) \\
f(x_1, \ldots, \mathsf{mark}(x_i), \ldots, x_n) &\to \mathsf{mark}(f(x_1, \ldots, x_n))
\end{aligned} \tag{2}$$

Example 8 (Transformation of simple liveness example) Recall the system from Ex. 6 again. Here, the transformation yields the following TRS $L(R, p)$.

$$\begin{aligned}
\mathsf{active}(\mathsf{f}(x)) &\to \mathsf{mark}(x) & \mathsf{proper}(\mathsf{c}) &\to \mathsf{ok}(\mathsf{c}) \\
\mathsf{top}(\mathsf{active}(\mathsf{c})) &\to \mathsf{top}(\mathsf{mark}(\mathsf{c})) & \mathsf{proper}(\mathsf{f}(x)) &\to \mathsf{f}(\mathsf{proper}(x)) \\
\mathsf{top}(\mathsf{mark}(x)) &\to \mathsf{top}(\mathsf{check}(x)) & \mathsf{f}(\mathsf{ok}(x)) &\to \mathsf{ok}(\mathsf{f}(x)) \\
\mathsf{check}(\mathsf{f}(x)) &\to \mathsf{f}(\mathsf{check}(x)) & \mathsf{start}(\mathsf{ok}(x)) &\to \mathsf{found}(x) \\
\mathsf{check}(x) &\to \mathsf{start}(\mathsf{match}(\mathsf{f}(X), x)) & \mathsf{f}(\mathsf{found}(x)) &\to \mathsf{found}(\mathsf{f}(x)) \\
\mathsf{match}(\mathsf{f}(x), \mathsf{f}(y)) &\to \mathsf{f}(\mathsf{match}(x, y)) & \mathsf{top}(\mathsf{found}(x)) &\to \mathsf{top}(\mathsf{active}(x)) \\
\mathsf{match}(X, x) &\to \mathsf{proper}(x) & \mathsf{active}(\mathsf{f}(x)) &\to \mathsf{f}(\mathsf{active}(x)) \\
& & \mathsf{f}(\mathsf{mark}(x)) &\to \mathsf{mark}(\mathsf{f}(x))
\end{aligned}$$

Note that it is really necessary to introduce the symbol proper and to check whether the whole term does not contain any new symbols from $\Sigma_G \setminus \Sigma$. If the proper-rules were removed, all remaining proper-terms were replaced by their arguments, and in $f(\mathsf{ok}(x_1), \ldots, \mathsf{found}(x_i), \ldots, \mathsf{ok}(x_n)) \to \mathsf{found}(f(x_1, \ldots, x_n))$, the terms $\mathsf{ok}(x_i)$ were replaced by x_i, then the transformation would not be complete any more. As a counterexample, regard $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{f}\}$ and the TRS

$$\begin{aligned}
\mathsf{top}(\mathsf{f}(\mathsf{b}, x, y)) &\to \mathsf{top}(\mathsf{f}(y, y, y)) \\
\mathsf{top}(\mathsf{f}(x, y, z)) &\to \mathsf{top}(\mathsf{f}(\mathsf{b}, \mathsf{b}, \mathsf{b})) \\
\mathsf{top}(\mathsf{a}) &\to \mathsf{top}(\mathsf{b})
\end{aligned}$$

and let $p = \mathsf{a}$. The TRS satisfies the liveness property since for any ground top term, after at most two steps one reaches a term without a (one obtains either

$\mathsf{top}(\mathsf{b})$ or $\mathsf{top}(\mathsf{f}(\mathsf{b},\mathsf{b},\mathsf{b})))$. However, with the modified transformation we would get the following non-terminating cyclic reduction where u is the term $\mathsf{found}(\mathsf{b})$:

$$\begin{array}{lll} \mathsf{top}(\mathsf{mark}(\mathsf{f}(u,u,u))) & \to \mathsf{top}(\mathsf{check}(\mathsf{f}(u,u,u))) & \to \\ \mathsf{top}(\mathsf{f}(u,\mathsf{check}(u),u)) & \to \mathsf{top}(\mathsf{found}(\mathsf{f}(\mathsf{b},\mathsf{check}(u),u))) & \to \\ \mathsf{top}(\mathsf{active}(\mathsf{f}(\mathsf{b},\mathsf{check}(u),u))) & \to \mathsf{top}(\mathsf{mark}(\mathsf{f}(u,u,u))) & \to \ldots \end{array}$$

To prove soundness and completeness of our transformation, we need several auxiliary lemmata about reductions with $L(R,p)$. The first lemma states that proper really checks whether its argument does not contain symbols from $\Sigma_G \setminus \Sigma$.

Lemma 9 (Reducing proper) *For $t \in \mathcal{T}(\Sigma_G)$ we have $\mathsf{proper}(t) \to^+_{L(R,p)} \mathsf{ok}(u)$ if and only if $t, u \in \mathcal{T}(\Sigma)$ and $t = u$.*

Proof. The proof is identical to the one in [7, Lemma 2] and [8]. □

Now we show that $\mathsf{match}(\overline{p}, t)$ checks whether p matches t and $t \in \mathcal{T}(\Sigma)$.

Lemma 10 (Reducing match) *Let $p \in \mathcal{T}(\Sigma, \mathcal{V})$, let $q \in \mathcal{T}(\Sigma(p), \mathcal{V})$ be linear, and let $t \in \mathcal{T}(\Sigma_G)$. We have $\mathsf{match}(\overline{q}, t) \to^+_{L(R,p)} \mathsf{ok}(u)$ iff $t = u \in \mathcal{T}(\Sigma)$ and $q\sigma = t$ for some σ.*

Proof. For the 'if'-direction, note that if q if a variable, then $\overline{q} = c$ for a constant $c \notin \Sigma$. Hence, $\mathsf{match}(\overline{q}, t) \to_{L(R,p)} \mathsf{proper}(t) \to^+_{L(R,p)} \mathsf{ok}(t)$ by Lemma 9. Otherwise, we use an easy induction on the structure of the term t. If t is a constant c, then $q\sigma = t$ implies $q = \overline{q} = c$ and hence, we have $\mathsf{match}(\overline{q}, t) = \mathsf{match}(c, c) \to_{L(R,p)} \mathsf{ok}(c) = \mathsf{ok}(t)$. Otherwise, t has the form $f(t_1, \ldots, t_n)$. Since q matches t, we obtain $q = f(q_1, \ldots, q_n)$ and $\overline{q} = f(\overline{q_1}, \ldots, \overline{q_n})$ where q_i matches t_i for all i. Note that the induction hypothesis implies $\mathsf{match}(\overline{q_i}, t_i) \to^+_{L(R,p)} \mathsf{ok}(t_i)$.[1] Thus, we obtain $\mathsf{match}(\overline{q}, t) = \mathsf{match}(f(\overline{q_1}, \ldots, \overline{q_n}), f(t_1, \ldots, t_n)) \to_{L(R,p)} f(\mathsf{match}(\overline{q_1}, t_1), \ldots, \mathsf{match}(\overline{q_n}, t_n)) \to^+_{L(R,p)} f(\mathsf{ok}(t_1), \ldots, \mathsf{ok}(t_n)) \to_{L(R,p)} \mathsf{ok}(f(t_1, \ldots, t_n)) = \mathsf{ok}(t)$.

We now prove the 'only if'-direction by induction on the length of the reduction. In other words, we prove "*for all q, t, and u*: $\mathsf{match}(\overline{q}, t) \to^+_{L(R,p)} \mathsf{ok}(u)$ implies that $t = u \in \mathcal{T}(\Sigma)$ and that q matches t". Thus, as induction hypothesis we have "*for all q', t', and u'*: $\mathsf{match}(\overline{q'}, t') \to^+_{L(R,p)} \mathsf{ok}(u')$ implies that $t' = u' \in \mathcal{T}(\Sigma)$ and that q' matches t'" provided that the reduction $\mathsf{match}(\overline{q'}, t') \to^+_{L(R,p)} \mathsf{ok}(u')$ is shorter than the reduction $\mathsf{match}(\overline{q}, t) \to^+_{L(R,p)} \mathsf{ok}(u)$.

If the first reduction step is in t, then $\mathsf{match}(\overline{q}, t) \to_{L(R,p)} \mathsf{match}(\overline{q}, t') \to^+_{L(R,p)} \mathsf{ok}(u)$ for a term t' with $t \to_{L(R,p)} t'$. The induction hypothesis states $t' = u \in \mathcal{T}(\Sigma)$ and $q\sigma = t'$. Note that $t' \in \mathcal{T}(\Sigma)$ implies $t = t'$ which proves the lemma.

[1] Formally, we prove "$\mathsf{match}(\overline{q}, t) \to^+_{L(R,p)} \mathsf{ok}(t)$ *for all q*" by induction on t. Hence, for all t_i which are smaller than t w.r.t. the induction relation, we have the induction hypothesis "$\mathsf{match}(\overline{q}, t_i) \to^+_{L(R,p)} \mathsf{ok}(t)$ *for all q*". So in the induction hypothesis, q can be instantiated to arbitrary terms (e.g., to q_i). Similar forms of induction are used throughout the paper.

Otherwise, the first reduction step is on the root position (since $\overline{q}$ is in normal form). If q is a variable, then q obviously matches t and we obtain $\mathsf{match}(\overline{q}, t) \to_{L(R,p)} \mathsf{proper}(t) \to^+_{L(R,p)} \mathsf{ok}(u)$ and $t = u \in \mathcal{T}(\Sigma)$ by Lemma 9. If q is a constant c, then a root reduction is only possible if $t = c$. We obtain $\mathsf{match}(\overline{q}, t) = \mathsf{match}(\overline{q}, c) \to_{L(R,p)} \mathsf{ok}(c)$. So in this case the lemma also holds.

Finally, if $q = f(q_1, \ldots, q_n)$, for a root reduction we have $t = f(t_1, \ldots, t_n)$. Then $\mathsf{match}(\overline{q}, t) = \mathsf{match}(f(\overline{q_1}, \ldots, \overline{q_n}), f(t_1, \ldots, t_n)) = f(\mathsf{match}(\overline{q_1}, t_1), \ldots, \mathsf{match}(\overline{q_n}, t_n)) \to^+_{L(R,p)} \mathsf{ok}(u)$. To reduce $f(\ldots)$ to $\mathsf{ok}(\ldots)$, all arguments of f must reduce to ok-terms. Hence, $\mathsf{match}(\overline{q_i}, t_i) \to^+_{L(R,p)} \mathsf{ok}(u_i)$ for all i where these reductions are shorter than the reduction $\mathsf{match}(\overline{q}, t) \to^+_{L(R,p)} \mathsf{ok}(u)$. The induction hypothesis implies that $t_i = u_i \in \mathcal{T}(\Sigma)$ and that there are substitutions σ_i with $q_i\sigma_i = t_i$. Since q is linear, we can combine these σ_i to one σ such that $q\sigma = t$. Moreover, this implies $u = f(u_1, \ldots, u_n)$ which proves the lemma. $\square$

The next lemma proves that check works properly, i.e., it checks whether its argument is a term from $\mathcal{T}(\Sigma)$ which contains an instance of p.

Lemma 11 (Reducing check) *Let $p \in \mathcal{T}(\Sigma, \mathcal{V})$ be linear and $t \in \mathcal{T}(\Sigma_G)$. We have $\mathsf{check}(t) \to^+_{L(R,p)} \mathsf{found}(u)$ iff $t = u \in \mathcal{T}(\Sigma)$ and t contains a subterm $p\sigma$.*

Proof. The 'if'-direction is again an easy structural induction proof on t. We prove the 'only if'-direction by induction on the length of the reduction.

If the first reduction step is in t, then the proof is analogous to Lemma 10. Otherwise, let $\mathsf{check}(t) \to_{L(R,p)} \mathsf{start}(\mathsf{match}(\overline{p}, t)) \to^+_{L(R,p)} \mathsf{found}(u)$. By the start-rule, this implies $\mathsf{match}(\overline{p}, t) \to^+_{L(R,p)} \mathsf{ok}(q)$ and $q \to^*_{L(R,p)} u$ for some term q. By Lemma 10, p matches t, $t = q \in \mathcal{T}(\Sigma)$, and therefore, $q = u$.

Now let $t = f(t_1, \ldots, t_n)$ and $\mathsf{check}(f(t_1, \ldots, t_n)) \to_{L(R,p)} f(\mathsf{proper}(t_1), \ldots, \mathsf{check}(t_i), \ldots, \mathsf{proper}(t_n)) \to^+_{L(R,p)} \mathsf{found}(u)$. The only way to reduce an f-term to a found-term is if one argument of f reduces to $\mathsf{found}(\ldots)$ and all others reduce to $\mathsf{ok}(\ldots)$. Therefore, there must be terms $u_1, \ldots, u_n$ such that $\mathsf{check}(t_i) \to^+_{L(R,p)} \mathsf{found}(u_i)$ and $\mathsf{proper}(t_j) \to^+_{L(R,p)} \mathsf{ok}(u_j)$ for all $j \neq i$. Since the length of the reduction from $\mathsf{check}(t_i)$ to $\mathsf{found}(u_i)$ is shorter than the length of $\mathsf{check}(t) \to^+_{L(R,p)} \mathsf{found}(u)$, the induction hypothesis yields $t_i = u_i \in \mathcal{T}(\Sigma)$ and that t_i contains $p\sigma$ for some σ. Lemma 9 implies $t_j = u_j \in \mathcal{T}(\Sigma)$. Hence, we have $u = f(u_1, \ldots, u_n)$ and the lemma is proved. $\square$

Lemma 12 shows that the top-rules (1), (2) are applied in an alternating way.

Lemma 12 (Reducing active and check) *For all $t, u \in \mathcal{T}(\Sigma_G)$ we have*

(a) $\mathsf{active}(t) \not\to^+_{L(R,p)} \mathsf{found}(u)$ *and* $\mathsf{active}(t) \not\to^+_{L(R,p)} \mathsf{ok}(u)$

(b) $\mathsf{check}(t) \not\to^+_{L(R,p)} \mathsf{mark}(u)$ *and* $\mathsf{proper}(t) \not\to^+_{L(R,p)} \mathsf{mark}(u)$

Proof. (a) By induction on $n \in \mathbb{N}$, we show that there is no reduction from $\mathsf{active}(t)$ to $\mathsf{found}(u)$ or to $\mathsf{ok}(u)$ of length n. If the first reduction step is in t, then the claim follows from the induction hypothesis. Otherwise, the reduction starts with a root step. This first step cannot be $\mathsf{active}(t) \to_{L(R,p)} \mathsf{mark}(u)$, since the root symbol mark can never be reduced again. Hence, we must have $t = f(t_1, \ldots, t_i, \ldots, t_n)$ and $\mathsf{active}(t) = \mathsf{active}(f(t_1, \ldots, t_i, \ldots, t_n)) \to_{L(R,p)} f(t_1, \ldots, \mathsf{active}(t_i), \ldots, t_n)$. In order to rewrite this term to a found- or ok-term, in particular $\mathsf{active}(t_i)$ must be rewritten to a found- or ok-term which contradicts the induction hypothesis.

(b) As in (a), we use induction on the length of the reduction. If the reduction starts inside t, then the claim is obvious. The reduction cannot start with $\mathsf{check}(t) \to_{L(R,p)} \mathsf{start}(\mathsf{match}(\overline{p}, t))$, since $\mathsf{start}(\ldots)$ can only be reduced to a found-term and found cannot be reduced any more. If t is a constant, then we obtain a similar contradiction if the reduction has the form $\mathsf{proper}(t) = \mathsf{proper}(c) \to_{L(R,p)} \mathsf{ok}(c)$, since ok can never be reduced any more either.
Otherwise, $t = f(t_1, \ldots, t_i, \ldots, t_n)$ and $\mathsf{check}(t) = \mathsf{check}(f(t_1, \ldots, t_i, \ldots, t_n)) \to_{L(R,p)} f(\mathsf{proper}(t_1), \ldots, \mathsf{check}(t_i), \ldots, \mathsf{proper}(t_n))$ or $\mathsf{proper}(t) = \mathsf{proper}(f(t_1, \ldots, t_n)) \to_{L(R,p)} f(\mathsf{proper}(t_1), \ldots, \mathsf{proper}(t_n))$. But in order to reduce these terms to a mark-term, one of the arguments must be reduced to a mark-term which is a contradiction to the induction hypothesis. □

We now prove that the top-rules are crucial for $L(R,p)$'s termination behavior.

Lemma 13 *Let $L'(R,p) = L(R,p) \setminus \{(1),(2)\}$. Then $L'(R,p)$ is terminating.*

Proof. Termination of $L'(R,p)$ can be proved by the recursive path order [5] using the precedence $\mathsf{active} > \mathsf{check} > \mathsf{match} > \mathsf{proper} > \mathsf{start} > f > \mathsf{ok} > \mathsf{found} > \mathsf{mark}$ for all $f \in \Sigma \cup \{X \mid x \in \mathcal{V}(p)\}$. □

Before relating $L(R,p)$ and $\to_G$, we study the connection of $L(R,p)$ and $\to_R$.

Lemma 14 *Let $t, u \in \mathcal{T}(\Sigma)$. Then we have $\mathsf{active}(t) \to^+_{L(R,p)} \mathsf{mark}(u)$ iff $t \to_R u$ and $\mathsf{top}(\mathsf{active}(t)) \to^+_{L(R,p)} \mathsf{top}(\mathsf{mark}(u))$ iff $\mathsf{top}(t) \to_R \mathsf{top}(u)$.*

Proof. The 'if'-direction is again easy by induction on t. For the 'only if'-direction, we first prove that $\mathsf{active}(t) \to^+_{L(R,p)} \mathsf{mark}(u)$ implies $t \to_R u$ by induction on the length of the reduction. Since $t \in \mathcal{T}(\Sigma)$, the first reduction step must be on the root position. If $\mathsf{active}(t) \to_{L(R,p)} \mathsf{mark}(u)$ on root position, then $t = l\sigma$ and $u = r\sigma$ for a rule $l \to r \in R$ and thus, $t \to_R u$. Otherwise, $t = f(t_1, \ldots, t_n)$ and $\mathsf{active}(t) = \mathsf{active}(f(t_1, \ldots, t_n)) \to_{L(R,p)} f(t_1, \ldots, \mathsf{active}(t_i), \ldots, t_n) \to^+_{L(R,p)} \mathsf{mark}(u)$. Thus, $\mathsf{active}(t_i) \to^+_{L(R,p)} \mathsf{mark}(u_i)$ and $u = f(t_1, \ldots, u_i, \ldots, t_n)$. The induction hypothesis implies $t_i \to_R u_i$ and hence, $t \to_R u$.

Now we show that $\mathsf{top}(\mathsf{active}(t)) \to^+_{L(R,p)} \mathsf{top}(\mathsf{mark}(u))$ implies $\mathsf{top}(t) \to_R \mathsf{top}(u)$. Since $t \in \mathcal{T}(\Sigma)$, the first reduction step must be on the root position or on position 1 (i.e., on the root of $\mathsf{active}(t)$). If $\mathsf{top}(\mathsf{active}(t)) \to_{L(R,p)}$

$\mathsf{top}(\mathsf{mark}(u))$ on root position, then $\mathsf{top}(t) \to \mathsf{top}(u)$ is an instance of an R-rule. If $\mathsf{top}(\mathsf{active}(t)) \to_{L(R,p)} \mathsf{top}(\mathsf{mark}(u))$ on position 1, then $t = l\sigma$ and $u = r\sigma$ for a rule $l \to r \in R$ and thus, $t \to_R u$ and $\mathsf{top}(t) \to_R \mathsf{top}(u)$. Otherwise, $t = f(t_1, \dots, t_n)$ and $\mathsf{top}(\mathsf{active}(t)) = \mathsf{top}(\mathsf{active}(f(t_1, \dots, t_n))) \to_{L(R,p)} \mathsf{top}(f(t_1, \dots, \mathsf{active}(t_i), \dots, t_n)) \to^+_{L(R,p)} \mathsf{top}(\mathsf{mark}(u))$. Thus, $\mathsf{active}(t_i) \to^+_{L(R,p)} \mathsf{mark}(u_i)$ and $u = f(t_1, \dots, u_i, \dots, t_n)$. As shown above, $\mathsf{active}(t_i) \to^+_{L(R,p)} \mathsf{mark}(u_i)$ implies $t_i \to_R u_i$ and hence, $\mathsf{top}(t) \to_R \mathsf{top}(u)$. □

Theorem 15 (Soundness and Completeness) *Let R be a top rewrite system over $\Sigma \cup \{\mathsf{top}\}$ with $\mathsf{top} \notin \Sigma$ and let $p \in \mathcal{T}(\Sigma, \mathcal{V})$ be linear. The TRS $L(R,p)$ is terminating (on all terms) iff the relation $\to_G$ is terminating on $\mathcal{T}_{\mathsf{top}}$.*

Proof. We first show the 'only if'-direction. If $\to_G$ does not terminate on $\mathcal{T}_{\mathsf{top}}$ then there is an infinite reduction $\mathsf{top}(t_1) \to_G \mathsf{top}(t_2) \to_G \dots$ where $t_1, t_2, \dots \in \mathcal{T}(\Sigma)$. By Lemma 14 we have $\mathsf{top}(\mathsf{active}(t_i)) \to^+_{L(R,p)} \mathsf{top}(\mathsf{mark}(t_{i+1}))$. Lemma 11 implies $\mathsf{check}(t_{i+1}) \to^+_{L(R,p)} \mathsf{found}(t_{i+1})$, since each t_{i+1} contains an instance of p. So we obtain the following contradiction to the termination of $L(R,p)$.

$$\begin{aligned}\mathsf{top}(\mathsf{active}(t_1)) \to^+_{L(R,p)} \mathsf{top}(\mathsf{mark}(t_2)) &\to_{L(R,p)} \mathsf{top}(\mathsf{check}(t_2)) \to^+_{L(R,p)} \\ \mathsf{top}(\mathsf{found}(t_2)) &\to_{L(R,p)} \mathsf{top}(\mathsf{active}(t_2)) \to^+_{L(R,p)} \dots\end{aligned}$$

For the 'if'-direction assume that $L(R,p)$ is not terminating. By type introduction [10] one can show that there exists an infinite $L(R,p)$-reduction of ground top terms. Due to Lemma 13 the reduction contains infinitely many applications of the rules (1) and (2). These rules must be applied in alternating order, since $\mathsf{active}(t)$ can never reduce to $\mathsf{found}(u)$ and $\mathsf{check}(t)$ can never reduce to $\mathsf{mark}(u)$ by Lemma 12. So the reduction has the following form where all reductions with the rules (1) and (2) are displayed.

$$\begin{aligned}\dots \to^*_{L(R,p)} \mathsf{top}(\mathsf{mark}(t_1)) &\to_{L(R,p)} \mathsf{top}(\mathsf{check}(t_1)) \to^+_{L(R,p)} \\ \mathsf{top}(\mathsf{found}(u_1)) &\to_{L(R,p)} \mathsf{top}(\mathsf{active}(u_1)) \to^+_{L(R,p)} \\ \mathsf{top}(\mathsf{mark}(t_2)) &\to_{L(R,p)} \mathsf{top}(\mathsf{check}(t_2)) \to^+_{L(R,p)} \\ \mathsf{top}(\mathsf{found}(u_2)) &\to_{L(R,p)} \mathsf{top}(\mathsf{active}(u_2)) \to^+_{L(R,p)} \dots\end{aligned}$$

By Lemma 11 we have $t_i = u_i \in \mathcal{T}(\Sigma)$ and that t_i contains an instance of p. Lemma 14 implies $\mathsf{top}(u_i) \to_R \mathsf{top}(t_{i+1})$. Together, we obtain $\mathsf{top}(t_1) \to_G \mathsf{top}(t_2) \to_G \dots$ in contradiction to the termination of $\to_G$ on $\mathcal{T}_{\mathsf{top}}$. □

Example 16 (Termination proof for simple liveness example) By Thm. 15, one can now use existing techniques for termination proofs of TRSs to verify liveness of systems like the one in Ex. 6. For instance, termination of the transformed TRS from Ex. 8 can easily be shown with dependency pairs [2]. The dependency pairs on cycles of the estimated dependency graph are

$$\mathsf{Top}(\mathsf{active}(\mathsf{c})) \to \mathsf{Top}(\mathsf{mark}(\mathsf{c})) \tag{3}$$

$$\mathsf{Top}(\mathsf{mark}(x)) \to \mathsf{Top}(\mathsf{check}(x)) \tag{4}$$
$$\mathsf{Top}(\mathsf{found}(x)) \to \mathsf{Top}(\mathsf{active}(x)) \tag{5}$$
$$\mathsf{Check}(\mathsf{f}(x)) \to \mathsf{Check}(x) \tag{6}$$
$$\mathsf{Match}(\mathsf{f}(x), \mathsf{f}(y)) \to \mathsf{Match}(x, y) \tag{7}$$
$$\mathsf{Proper}(\mathsf{f}(x)) \to \mathsf{Proper}(x) \tag{8}$$
$$\mathsf{F}(\mathsf{ok}(x)) \to \mathsf{F}(x) \tag{9}$$
$$\mathsf{F}(\mathsf{found}(x)) \to \mathsf{F}(x) \tag{10}$$
$$\mathsf{Active}(\mathsf{f}(x)) \to \mathsf{Active}(x) \tag{11}$$
$$\mathsf{F}(\mathsf{mark}(x)) \to \mathsf{F}(x) \tag{12}$$

The dependency pair (4) is narrowed to

$$\mathsf{Top}(\mathsf{mark}(\mathsf{f}(x))) \to \mathsf{Top}(\mathsf{f}(\mathsf{check}(x))) \tag{13}$$
$$\mathsf{Top}(\mathsf{mark}(x)) \to \mathsf{Top}(\mathsf{start}(\mathsf{match}(\mathsf{f}(X), x))) \tag{14}$$

Then (14) is narrowed further to

$$\mathsf{Top}(\mathsf{mark}(\mathsf{f}(x))) \to \mathsf{Top}(\mathsf{start}(\mathsf{match}(X, x))) \tag{15}$$

Now (3) is no longer on a cycle of the estimated dependency graph. Hence, (5) can be narrowed to

$$\mathsf{Top}(\mathsf{found}(\mathsf{f}(x))) \to \mathsf{Top}(\mathsf{mark}(x)) \tag{16}$$

So to summarize, the Top-dependency pairs are replaced by

$$\begin{array}{rll}
\mathsf{Top}(\mathsf{mark}(\mathsf{f}(x))) \to \mathsf{Top}(\mathsf{f}(\mathsf{check}(x))) & & (13)\\
\mathsf{Top}(\mathsf{mark}(\mathsf{f}(x))) \to \mathsf{Top}(\mathsf{start}(\mathsf{match}(X, x))) & & (15)\\
\mathsf{Top}(\mathsf{found}(\mathsf{f}(x))) \to \mathsf{Top}(\mathsf{mark}(x)) & & (16)
\end{array}$$

Note that due to Lemma 12, every cycle of Top-dependency pairs contains the pair (16). Hence, it suffices if just (16) is strictly decreasing, whereas (13) and (15) only have to be weakly decreasing.

We use an argument filtering which maps $\mathsf{start}(x)$ to x and which maps $\mathsf{match}(x, y)$ to $\mathsf{match}(y)$. Then all resulting constraints are satisfied by the recursive path order using a precedence where active, mark, check, match, proper, ok, found, and f are considered equal.

5 Proving Liveness

In Sect. 5.1 we present a sound transformation which is more suitable for mechanizing liveness proofs than the complete transformation from the preceding section. The reason is that for this new transformation, termination of the transformed TRS is much easier to show. On the other hand, the approach in this section is incomplete, i.e., it cannot succeed for all examples. Subsequently, in Sect. 5.2 we introduce an automatic preprocessing technique based on *semantic labelling* [11] to simplify these termination proofs further. In this way, rewriting techniques can be used to mechanize the verification of liveness properties.

5.1 A Sound Transformation for Liveness

The goal of a sound transformation for liveness properties $\mathsf{Live}(\mathcal{T}_{\mathsf{top}}, \rightarrow_R, G)$ is to transform the original TRS R to another TRS $LS(R,p)$ such that the required property $\mathsf{SN}(\mathcal{T}_{\mathsf{top}}, \rightarrow_G)$ can be concluded from termination of the transformed system. To obtain a simple sound transformation, the idea is to introduce only one new symbol check. A new occurrence of check is created in every application of a top rule. If check finds an instantiation of p then check may be removed. Otherwise, check remains in the term where it may block further reductions.

Definition 17 ($LS(R,p)$) *For a top rewrite system R over $\Sigma \cup \{\mathsf{top}\}$ with $\mathsf{top} \notin \Sigma$ and $p \in \mathcal{T}(\Sigma, \mathcal{V})$, let $LS(R,p)$ consist of the following rules.*

$$\begin{array}{rll} l \rightarrow r & & \textit{for all non-top rules } l \rightarrow r \textit{ in } R \\ \mathsf{top}(t) \rightarrow \mathsf{top}(\mathsf{check}(u)) & & \textit{for all top rules } \mathsf{top}(t) \rightarrow \mathsf{top}(u) \\ \mathsf{check}(f(x_1,..,x_n)) \rightarrow f(x_1,..,\mathsf{check}(x_i),..,x_n) & & \textit{for } f \in \Sigma \textit{ of arity } n \geq 1,\ i = 1,\ldots,n \\ \mathsf{check}(p) \rightarrow p & & \end{array}$$

Example 18 (Simple example revisited) To illustrate the transformation, reconsider the system from Ex. 6. Here, $LS(R, \mathsf{f}(x))$ is the following TRS whose termination can be proved by dependency pairs and the recursive path order, since the dependency pair $\mathsf{Top}(\mathsf{c}) \rightarrow \mathsf{Top}(\mathsf{check}(\mathsf{c}))$ is not on a cycle of the dependency graph.

$$\begin{array}{llll} \mathsf{top}(\mathsf{c}) \rightarrow \mathsf{top}(\mathsf{check}(\mathsf{c})) & (17) & \mathsf{check}(\mathsf{f}(x)) \rightarrow \mathsf{f}(\mathsf{check}(x)) & (19) \\ \mathsf{f}(x) \rightarrow x & (18) & \mathsf{check}(\mathsf{f}(x)) \rightarrow \mathsf{f}(x) & (20) \end{array}$$

Now we show that this transformation is indeed sound. In other words, the above termination proof verifies the liveness property of our example.

Theorem 19 (Soundness) *Let R be a top rewrite system over $\Sigma \cup \{\mathsf{top}\}$ with $\mathsf{top} \notin \Sigma$, let $p \in \mathcal{T}(\Sigma, \mathcal{V})$, and let $G = \{t \mid t \text{ does not contain an instance of } p\}$. If $LS(R,p)$ is terminating then there is no infinite $\rightarrow_G$-reduction of top terms.*

Proof. Assume there is an infinite $\rightarrow_G$-reduction of top terms $\mathsf{top}(t_1) \rightarrow_G \mathsf{top}(t_2) \rightarrow_G \ldots$ Since top does not occur in p, every t_i has the form $C_i[p\sigma_i]$ for some context C_i and substitution σ_i. To prove the theorem, we show that $\mathsf{top}(t_i) \rightarrow^+_{LS(R,p)} \mathsf{top}(t_{i+1})$ for every i, by which we obtain an infinite $LS(R,p)$-reduction.

If $\mathsf{top}(t_i) \rightarrow_R \mathsf{top}(t_{i+1})$ by the application of a non-top rule $l \rightarrow r$ then we also have $\mathsf{top}(t_i) \rightarrow_{LS(R,p)} \mathsf{top}(t_{i+1})$ since $l \rightarrow r$ is also contained in $LS(R,p)$. Otherwise, $\mathsf{top}(t_i) \rightarrow_R \mathsf{top}(t_{i+1})$ by a top rule $\mathsf{top}(t) \rightarrow \mathsf{top}(u)$. Hence, $t_i = t\sigma$ and $t_{i+1} = u\sigma$ for some σ. Since $LS(R,p)$ contains the rules $\mathsf{check}(f(x_1,\ldots,x_n)) \rightarrow f(x_1,\ldots,\mathsf{check}(x_i),\ldots,x_n)$ for all f with arity ≥ 1, we obtain

$$\begin{array}{rlll} \mathsf{top}(t_i) = \mathsf{top}(t\sigma) & \rightarrow_{LS(R,p)} & \mathsf{top}(\mathsf{check}(u\sigma)) & = \mathsf{top}(\mathsf{check}(C_{i+1}[p\sigma_{i+1}])) \\ & \rightarrow^*_{LS(R,p)} & \mathsf{top}(C_{i+1}[\mathsf{check}(p)\sigma_{i+1}]) & \\ & \rightarrow_{LS(R,p)} & \mathsf{top}(C_{i+1}[p\sigma_{i+1}]) & = \mathsf{top}(t_{i+1}) \end{array}$$ □

Example 20 (Sound transformation is not complete) However, the transformation is incomplete as can be shown by the following top rewrite system R

$$\mathsf{top}(\mathsf{f}(x,\mathsf{b})) \;\to\; \mathsf{top}(\mathsf{f}(\mathsf{b},\mathsf{b})) \qquad\qquad \mathsf{a} \;\to\; \mathsf{b}$$

where $\Sigma = \{\mathsf{a},\mathsf{b},\mathsf{f}\}$ and $p = \mathsf{a}$. In this example, normal forms do not contain a any more and every infinite reduction of top terms reaches the term $\mathsf{top}(\mathsf{f}(\mathsf{b},\mathsf{b}))$ which does not contain the symbol a either. Hence, the liveness property holds. However, $LS(R,p)$ admits the following infinite reduction:

$$\mathsf{top}(\mathsf{f}(\mathsf{b},\mathsf{b}))\to \mathsf{top}(\mathsf{check}(\mathsf{f}(\mathsf{b},\mathsf{b})))\to \mathsf{top}(\mathsf{f}(\mathsf{check}(\mathsf{b}),\mathsf{b}))\to \mathsf{top}(\mathsf{check}(\mathsf{f}(\mathsf{b},\mathsf{b})))\to\ldots$$

Thus, the transformation of Def. 17 is incomplete, because even if check remains in a term, this does not necessarily block further (infinite) reductions.

5.2 A Preprocessing Procedure for Verifying Liveness

The aim of our sound transformation from Def. 17 is to simplify (and possibly automate) the termination proofs which are required in order to show liveness properties. Since the TRSs resulting from our transformation have a particular form, we now present a method to preprocess such TRSs. This preprocessing is especially designed for this form of TRSs and in this way, their termination proofs can often be simplified significantly. The method consists of four steps which can be performed automatically:

(a) First one deletes rules which cannot cause non-termination.
(b) Then one applies the well-known transformation technique of *semantic labelling* [11] with a particularly chosen model and labelling. (This restricted form of semantic labelling can be done automatically.)
(c) Then one again deletes rules which cannot cause non-termination.
(d) Finally one uses an existing automatic technique (e.g., the recursive path order or dependency pairs) to prove termination of the resulting TRS.

To delete rules in Step (a) and (c) we use the following lemma. For a function symbol $f \in \Sigma$ and a term $t \in \mathcal{T}(\Sigma,\mathcal{V})$, let $\#_f(t)$ be the number of f-symbols occurring in t. For $\varnothing \neq \Sigma' \subseteq \Sigma$ let $\#_{\Sigma'}(t) = \sum_{f\in\Sigma'} \#_f(t)$.

Lemma 21 *Let R be a TRS such that*

- *R is* non-duplicating, *i.e., for every rule $l \to r$, no variable occurs more often in r than in l, and*
- *$\#_{\Sigma'}(l) \geq \#_{\Sigma'}(r)$ for all rules $l \to r$ in R*

for some $\Sigma' \subseteq \Sigma$. Let R' consist of those rules $l \to r$ from R which satisfy $\#_{\Sigma'}(l) > \#_{\Sigma'}(r)$. Then R is terminating if and only if $R \setminus R'$ is terminating.

Proof. The 'only if'-part holds since $R \setminus R' \subseteq R$. For the 'if'-part assume that $R \setminus R'$ is terminating and that we have an infinite R-reduction. Due to the conditions of the lemma we have $\#_{\Sigma'}(t) \geq \#_{\Sigma'}(u)$ for every step $t \to_R u$ and $\#_{\Sigma'}(t) > \#_{\Sigma'}(u)$ for every step $t \to_{R'} u$. Hence, due to well-foundedness of the natural numbers, the infinite R-reduction contains only finitely many R'-steps. After removing the finite initial part containing all these R'-steps, the remaining part is an infinite $R \setminus R'$-reduction, which gives a contradiction. $\square$

The application of Lemma 21 is easily automated as follows: for all sets $\Sigma' \subseteq \Sigma$ with $|\Sigma'| \leq n$ for some (small) $n \in \mathbb{N}$, it is checked whether $\#_{\Sigma'}(l) \geq \#_{\Sigma'}(r)$ for all rules $l \to r$. If so, then all rules $l \to r$ satisfying $\#_{\Sigma'}(l) > \#_{\Sigma'}(r)$ are removed. This process is repeated until no rule can be removed any more.

As a first example, we apply Lemma 21 to the TRS from Ex. 18. By counting the occurrences of f, we note that the number of f-symbols strictly decreases in Rule (18) and it remains the same in all other rules. Hence, due to Lemma 21 we can drop this rule when proving termination of the TRS. It turns out that in this case repetition of this process does not succeed in removing more rules.

In our termination procedure, in Step (b) we apply a particular instance of *semantic labelling* [11]. Before describing this instance we briefly explain how semantic labelling works as a tool to prove termination of a TRS R over the signature Σ: One starts by choosing a *model* for the TRS R. Thus, one defines a non-empty carrier set M and for every function symbol $f \in \Sigma$ of arity n, an interpretation $f_M : M^n \to M$ is chosen. As usual, every variable assignment $\alpha : \mathcal{V} \to M$ can be extended to terms from $\mathcal{T}(\Sigma, \mathcal{V})$ by inductively defining $\alpha(f(t_1, \ldots, t_n)) = f_M(\alpha(t_1), \ldots, \alpha(t_n))$. The interpretation is a *model* for R if $\alpha(l) = \alpha(r)$ for every rule $l \to r$ in R and every variable assignment $\alpha : \mathcal{V} \to M$.

Using this model, the TRS R over the signature Σ is transformed into a *labelled* TRS $\overline{R}$ over the *labelled* signature $\overline{\Sigma}$. Here, every function symbol $f \in \Sigma$ of arity n may be labelled by n elements from M, i.e., $\overline{\Sigma} = \{f_{a_1,\ldots,a_n} \mid f \in \Sigma, n = \text{arity}(f), a_i \in M\}$ where the arity of $f_{a_1,\ldots,a_n}$ is the same as the arity of f. For any variable assignment $\alpha : \mathcal{V} \to M$, we define a function $\mathsf{lab}_\alpha : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathcal{T}(\overline{\Sigma}, \mathcal{V})$ which labels every function symbol by the interpretations of its arguments:

$$
\begin{aligned}
\mathsf{lab}_\alpha(x) &= x, \quad \text{for } x \in \mathcal{V} \\
\mathsf{lab}_\alpha(f(t_1, \ldots, t_n)) &= f_{\alpha(t_1),\ldots,\alpha(t_n)}(\mathsf{lab}_\alpha(t_1), \ldots, \mathsf{lab}_\alpha(t_n))
\end{aligned}
$$

Now the TRS $\overline{R}$ is defined to consist of all rules $\mathsf{lab}_\alpha(l) \to \mathsf{lab}_\alpha(r)$ for all variable assignments $\alpha : \mathcal{V} \to M$ and all rules $l \to r$ in R. The main theorem of semantic labelling states that R is terminating if and only if $\overline{R}$ is terminating.

In general, semantic labelling permits a lot of freedom and is hard to automate, since one may choose arbitrary models. Moreover, in full semantic labelling one may also use arbitrary labellings. However, we will restrict ourselves to the case where $M = \{0, 1\}$. Now there are only finitely many possibilities for the interpretations f_M in the model. This means that with this restriction the termination method consisting of the steps (a) - (d) is fully decidable.

To improve efficiency and to avoid checking all possibilities of a two-element model for semantic labelling, we now propose heuristics for choosing the interpretations f_M in such a model. These heuristics are adapted to the special form of TRSs resulting from our transformation in Def. 17 when verifying liveness properties. The main objective is that we want to distinguish between terms that contain instances of p and terms that do not. Therefore, our aim is to interpret the former terms by 0 and the latter terms by 1. Since the intention of check is that an occurrence of p should be found, $\mathsf{check}(x)$ will be interpreted as the constant function 0. Since top only occurs at the top, for $\mathsf{top}(x)$ we may also choose a constant function. Having these objectives in mind, we arrive at the following heuristic for choosing the operations f_M in the model $M = \{0, 1\}$:

- $\mathsf{top}_M(x) = \mathsf{check}_M(x) = f_M(x_1, \ldots, x_n) = 0$ for $x = 0, 1$, where f is the root symbol of p;
- $c_M = 1$ for every constant c, except if $p = c$;
- $f_M(x_1, \ldots, x_n) = \min(x_1, \ldots, x_n)$ for all other symbols f as long as this does not conflict with the model requirement $\alpha(l) = \alpha(r)$. In particular, for the remaining unary symbols f one tries to choose $f_M(x) = x$.

Applying these heuristics to our example results in the following interpretation:

$$\mathsf{top}_M(x) = \mathsf{check}_M(x) = \mathsf{f}_M(x) = 0 \quad \text{for } x \in M = \{0, 1\} \quad \text{and} \quad \mathsf{c}_M = 1$$

One checks that this is a model for the TRS. Here it is essential that we first removed Rule (18), since $\mathsf{f}_M(x) = 0 \neq x$ if $x = 1$. The labelling results in the TRS

$$\begin{array}{rll}
\mathsf{top}_1(\mathsf{c}) \to \mathsf{top}_0(\mathsf{check}_1(\mathsf{c})) & & \\
\mathsf{check}_0(\mathsf{f}_i(x)) \to \mathsf{f}_0(\mathsf{check}_i(x)) & & \text{for } i \in \{0, 1\} \\
\mathsf{check}_0(\mathsf{f}_i(x)) \to \mathsf{f}_i(x) & & \text{for } i \in \{0, 1\}
\end{array}$$

In Step (c) of our termination procedure, we apply Lemma 21 again. By counting the occurrences of top_1, we can drop the first rule. By counting f_1, the second rule can be removed if i is 1, and by counting check_0 we can delete the third rule. So the remaining TRS just contains the rule $\mathsf{check}_0(\mathsf{f}_0(x)) \to \mathsf{f}_0(\mathsf{check}_0(x))$ whose termination is trivial to prove by the recursive path order.

This example indicates that preprocessing a TRS according to Steps (a) - (c) often simplifies the termination proof considerably. For the original TRS of Ex. 18, one needs dependency pairs for the termination proof, whereas after the transformation a very simple recursive path order is sufficient.

6 Case Studies of Liveness

To demonstrate the applicability of our approach, we now regard four examples of liveness properties for networks of processes. To model protocols with TRSs, we use the approach to represent the state of a whole network of processes by a top term. In Sect. 6.4 we also discuss a straightforward extension of our approach: instead of one unary top symbol, the framework can easily be extended to several[2] top symbols of arbitrary arity.

[2] Similarly, an extension to liveness w.r.t. several terms $p_1, \ldots, p_n$ instead of just one p is also easily possible.

6.1 A Shared Resource With a Single Waiting Line

The following case study is motivated by protocols similar to the *bakery protocol* [9]. We describe a network of processes which want to gain access to a shared resource. The processes waiting for the resource are served one after another. Since the maximal size of the waiting line is fixed, a new process can only enter the waiting line if a process in the current line has been "served" (i.e., if it has been granted access to the resource). The maximal length n of the waiting line is arbitrary, and we will show that the liveness property holds for all $n \in \mathbb{N}$. Hence, techniques like classical model checking are not applicable here.

The processes in the line are served on a "first in - first out" basis (this corresponds to the serving of clients in a shop). So at the front end of the waiting line, a process may be served, where serving is denoted by a constant serve. If a process is served, its place in the line is replaced by a free place, denoted by free. If the place in front of some process is free, this process may take the free place, creating a free place on its original position. If the line has a free place at its back end, a new process new may enter the waiting line, taking over the position of the free place. Apart from new processes represented by new we also consider old processes represented by old, which were already in the line initially. We want to verify the liveness property that eventually all old processes will be served. Introducing the symbol top at the back end of the waiting line, this network is described by the following top rewrite system R:

$$
\begin{array}{rclcrcl}
\mathsf{top}(\mathsf{free}(x)) & \to & \mathsf{top}(\mathsf{new}(x)) & \qquad & \mathsf{new}(\mathsf{serve}) & \to & \mathsf{free}(\mathsf{serve}) \\
\mathsf{new}(\mathsf{free}(x)) & \to & \mathsf{free}(\mathsf{new}(x)) & & \mathsf{old}(\mathsf{serve}) & \to & \mathsf{free}(\mathsf{serve}) \\
\mathsf{old}(\mathsf{free}(x)) & \to & \mathsf{free}(\mathsf{old}(x)) & & & &
\end{array}
$$

Note that the above TRS admits infinite reductions of top terms. For instance,

$$
\mathsf{top}(\mathsf{new}(\mathsf{serve})) \to_R \mathsf{top}(\mathsf{free}(\mathsf{serve})) \to_R \mathsf{top}(\mathsf{new}(\mathsf{serve})) \to_R \ldots
$$

describes that the protocol for serving processes and for letting new processes enter may go on forever. But we will prove that after finitely many steps one reaches a term without the symbol old, i.e., eventually all old processes are served. In our terminology this liveness property is represented by $\mathsf{Live}(\mathcal{T}_{\mathsf{top}}, \to_R, G)$ where $G = \{t \mid t$ does not contain an instance of $\mathsf{old}(x)\}$. Note that this liveness property does not hold for various variations of this system. For instance, if processes are allowed to swap by $\mathsf{new}(\mathsf{old}(x)) \to \mathsf{old}(\mathsf{new}(x))$, or if new processes are always allowed to line up by $\mathsf{top}(x) \to \mathsf{top}(\mathsf{new}(x))$, then liveness is destroyed.

Since $\mathsf{top}(\mathsf{serve})$ is the only ground top term that is in normal form, we conclude that $\mathsf{NF}(\mathcal{T}_{\mathsf{top}}) \subseteq G$. Hence by Thm. 4 the required liveness property is equivalent to $\mathsf{SN}(\mathcal{T}_{\mathsf{top}}, \to_G)$. To prove this termination property of $\to_G$, according to Thm. 19 we may prove termination of the TRS $LS(R, p)$:

$$\mathsf{top}(\mathsf{free}(x)) \to \mathsf{top}(\mathsf{check}(\mathsf{new}(x))) \quad (21)$$
$$\mathsf{new}(\mathsf{free}(x)) \to \mathsf{free}(\mathsf{new}(x)) \quad (22)$$
$$\mathsf{old}(\mathsf{free}(x)) \to \mathsf{free}(\mathsf{old}(x)) \quad (23)$$
$$\mathsf{new}(\mathsf{serve}) \to \mathsf{free}(\mathsf{serve}) \quad (24)$$
$$\mathsf{old}(\mathsf{serve}) \to \mathsf{free}(\mathsf{serve}) \quad (25)$$
$$\mathsf{check}(\mathsf{free}(x)) \to \mathsf{free}(\mathsf{check}(x)) \quad (26)$$
$$\mathsf{check}(\mathsf{new}(x)) \to \mathsf{new}(\mathsf{check}(x)) \quad (27)$$
$$\mathsf{check}(\mathsf{old}(x)) \to \mathsf{old}(\mathsf{check}(x)) \quad (28)$$
$$\mathsf{check}(\mathsf{old}(x)) \to \mathsf{old}(x) \quad (29)$$

While standard techniques for automated termination proofs of TRSs do not succeed for this TRS, with the preprocessing steps (a) - (c) termination can easily be shown automatically.

According to (a), we first delete rules which do not influence termination. By counting the occurrences of old, with Lemma 21 we can remove Rule (25). Then in Step (b), we apply the heuristics for semantic labelling and arrive at

$$\mathsf{top}_M(x) = \mathsf{check}_M(x) = \mathsf{old}_M(x) = 0, \;\; \mathsf{new}_M(x) = \mathsf{free}_M(x) = x, \;\; \mathsf{serve}_M = 1$$

for $x \in M = \{0, 1\}$. Indeed this is a model for the TRS. For that purpose, we had to remove Rule (25) since $\mathsf{old}_M(\mathsf{serve}_M) = 0 \neq 1 = \mathsf{free}_M(\mathsf{serve}_M)$. The corresponding labelled TRS $\overline{R}$ is

$$\begin{array}{rcll}
\mathsf{top}_i(\mathsf{free}_i(x)) & \to & \mathsf{top}_0(\mathsf{check}_i(\mathsf{new}_i(x))) & (21_i)\\
\mathsf{new}_i(\mathsf{free}_i(x)) & \to & \mathsf{free}_i(\mathsf{new}_i(x)) & (22_i)\\
\mathsf{old}_i(\mathsf{free}_i(x)) & \to & \mathsf{free}_0(\mathsf{old}_i(x)) & (23_i)\\
\mathsf{new}_1(\mathsf{serve}) & \to & \mathsf{free}_1(\mathsf{serve}) & (24)\\
\mathsf{check}_i(\mathsf{free}_i(x)) & \to & \mathsf{free}_0(\mathsf{check}_i(x)) & (26_i)\\
\mathsf{check}_i(\mathsf{new}_i(x)) & \to & \mathsf{new}_0(\mathsf{check}_i(x)) & (27_i)\\
\mathsf{check}_0(\mathsf{old}_i(x)) & \to & \mathsf{old}_0(\mathsf{check}_i(x)) & (28_i)\\
\mathsf{check}_0(\mathsf{old}_i(x)) & \to & \mathsf{old}_i(x), & (29_i)
\end{array}$$

for $i \in \{0, 1\}$. It remains to prove termination of this TRS of 15 rules. According to Step (c) we repeatedly apply Lemma 21. By consecutively choosing $\Sigma' = \{f\}$ for f being top_1, old_1, new_1, free_1, free_0, and check_0, the rules (21_1), (28_1), (24) and (27_1), (23_1) and (26_1), (21_0), and finally (29_0) and (29_1) are removed. Termination of the remaining system consisting of the rules (22_0), (22_1), (23_0), (26_0), (27_0), and (28_0) is easily proved by the recursive path order, using a precedence satisfying $\mathsf{check}_0 > \mathsf{old}_0 > \mathsf{free}_0$, $\mathsf{check}_0 > \mathsf{new}_0 > \mathsf{free}_0$, and $\mathsf{new}_1 > \mathsf{free}_1$. Hence, the liveness property of this example can be proved automatically.

6.2 Emptying Local Memories of Processes

The following example is a variant of a process verification problem from [6]. To simplify the presentation, we abstract from all details and present a variant which represents the basic underlying problem. Essentially, a process keeps on sending messages from its local memory (which are removed from the memory afterwards). In general, a process can also receive messages which result in new items being stored in its local memory. However, in order to empty the local memories of the processes in the end, only the empty message is sent to the process while it keeps on sending the first m messages from its memory repeatedly for some fixed number m. Of course, if the memory contains less than m items, then it can only send the messages that are in its memory. Moreover, if its memory is empty it sends the empty message. The goal is to verify that eventually the memory of the process will indeed be empty. In the following TRS we describe the case where $m = 1$. Here, top has the value of the local memory as its argument and the goal is to prove that eventually this memory will be empty. The memory is modelled as a list and thus, we want to prove that eventually no non-empty list is present any more. So our aim is to prove liveness w.r.t. the term $p = x : y$, where ":" is the list constructor. (More precisely, we prove that in every infinite reduction of ground top terms, one will eventually reach a term without ":". Whether there exist normal forms containing ":" depends on the set I of initial terms.) If x is the current local memory, then $\mathsf{send}(x)$ indicates that the local memory contains the list x of messages and that now the first message from x should be sent and deleted afterwards. The term $\mathsf{sent}(x)$ denotes that sending has taken place and that x is the remaining list of messages in the local memory.

$$\begin{aligned}
\mathsf{top}(\mathsf{sent}(x)) &\to \mathsf{top}(\mathsf{send}(x)) \\
\mathsf{send}(\mathsf{nil}) &\to \mathsf{sent}(\mathsf{nil}) \\
\mathsf{send}(x : y) &\to \mathsf{sent}(y)
\end{aligned}$$

Our sound transformation of Def. 17 results in the following TRS $LS(R, x : y)$.

$$\mathsf{top}(\mathsf{sent}(x)) \to \mathsf{top}(\mathsf{check}(\mathsf{send}(x))) \tag{30}$$
$$\mathsf{send}(\mathsf{nil}) \to \mathsf{sent}(\mathsf{nil}) \tag{31}$$
$$\mathsf{send}(x : y) \to \mathsf{sent}(y) \tag{32}$$
$$\mathsf{check}(\mathsf{sent}(x)) \to \mathsf{sent}(\mathsf{check}(x)) \tag{33}$$
$$\mathsf{check}(\mathsf{send}(x)) \to \mathsf{send}(\mathsf{check}(x)) \tag{34}$$
$$\mathsf{check}(x : y) \to \mathsf{check}(x) : y \tag{35}$$
$$\mathsf{check}(x : y) \to x : \mathsf{check}(y) \tag{36}$$
$$\mathsf{check}(x : y) \to x : y \tag{37}$$

Due to Lemma 21, by counting the occurrences of ":", we can drop Rule (32). Then the function symbols are labelled according to our heuristic: $M = \{0, 1\}$ and

$$\mathsf{top}_M(x) = \mathsf{check}_M(x) = x :_M y = 0,\ \mathsf{sent}_M(x) = \mathsf{send}_M(x) = x,\ \mathsf{nil}_M = 1$$

for $x, y \in \{0, 1\}$. Indeed, this is a model of the TRS. Labelling the rules results in

$$\mathsf{top}_i(\mathsf{sent}_i(x)) \to \mathsf{top}_0(\mathsf{check}_i(\mathsf{send}_i(x)))$$

$$
\begin{aligned}
\mathsf{send}_1(\mathsf{nil}) &\to \mathsf{sent}_1(\mathsf{nil})\\
\mathsf{check}_i(\mathsf{sent}_i(x)) &\to \mathsf{sent}_0(\mathsf{check}_i(x))\\
\mathsf{check}_i(\mathsf{send}_i(x)) &\to \mathsf{send}_0(\mathsf{check}_i(x))\\
\mathsf{check}_0(x :_{ij} y) &\to \mathsf{check}_i(x) :_{0j} y\\
\mathsf{check}_0(x :_{ij} y) &\to x :_{i0} \mathsf{check}_j(y)\\
\mathsf{check}_0(x :_{ij} y) &\to x :_{ij} y
\end{aligned}
$$

for $i \in \{0,1\}$. We apply Lemma 21 and count the occurrences of top_1 to drop the first rule if $i = 1$. Now, by the same lemma we can drop the second rule by counting the occurrences of send_1. Moreover, in the fourth rule, i must be 0. Similarly, i must also be 0 in the third rule as can be determined by Lemma 21 when counting the occurrences of sent_1. This allows us to drop the first rule for $i = 0$ as well by counting the occurrences of sent_0. By regarding the occurrences of check_1, Lemma 21 also determines that in the fifth rule, i must be 0 and in the sixth rule, j must be 0. Moreover, the last rule can be deleted by counting the occurrences of check_0. So we obtain

$$
\begin{aligned}
\mathsf{check}_0(\mathsf{sent}_0(x)) &\to \mathsf{sent}_0(\mathsf{check}_0(x))\\
\mathsf{check}_0(\mathsf{send}_0(x)) &\to \mathsf{send}_0(\mathsf{check}_0(x))\\
\mathsf{check}_0(x :_{0j} y) &\to \mathsf{check}_0(x) :_{0j} y\\
\mathsf{check}_0(x :_{i0} y) &\to x :_{i0} \mathsf{check}_0(y)
\end{aligned}
$$

Termination is trivial to prove with the recursive path order by choosing check_0 to be maximal in the precedence.

6.3 Communication on a Ring of Processes

We consider the following protocol on a ring of processes (similar to a token ring protocol). Every process is in one of the three states sent, rec (received), or no (nothing). Initially at least one of the processes is in state rec which means that it has received a message (token). Now the protocol is defined as follows:

> If a process is in state rec then it may send its message to its right neighbor which then will be in state rec, while the process itself then will be in state sent.

Clearly, at least one process will always be in state rec, and this procedure can go on forever; we will prove that eventually no process will be in state no. This means that eventually all processes have received the message; a typical liveness property to be proved. The requirement $\mathsf{NF}(I) \subseteq G$ and in fact $\mathsf{NF}(I) = \varnothing$ (for I consisting of all configurations containing rec) is easily seen to hold on the protocol level. According to Thm. 4, for proving the desired liveness property it suffices to show $\mathsf{SN}(I, \to_G)$. The protocol is encoded by unary symbols sent, rec, and no, where the right neighbor of each of these symbols corresponds to the root of its argument. To obtain a ring topology we add a unary symbol top and a constant bot. For a symbol with the argument bot, its right neighbor is defined to be the symbol just below top. So again the state of the whole ring network is represented by a top-term $\mathsf{top}(f_1(\ldots(f_n(\mathsf{bot}))\ldots))$. Here the size n of the ring is arbitrary. In order to pass messages from the bot-process n to the top-process 1, an auxiliary unary symbol up is introduced.

$$\mathsf{rec}(\mathsf{rec}(x)) \to \mathsf{sent}(\mathsf{rec}(x)) \tag{38}$$
$$\mathsf{rec}(\mathsf{sent}(x)) \to \mathsf{sent}(\mathsf{rec}(x)) \tag{39}$$
$$\mathsf{rec}(\mathsf{no}(x)) \to \mathsf{sent}(\mathsf{rec}(x)) \tag{40}$$
$$\mathsf{rec}(\mathsf{bot}) \to \mathsf{up}(\mathsf{sent}(\mathsf{bot})) \tag{41}$$
$$\mathsf{rec}(\mathsf{up}(x)) \to \mathsf{up}(\mathsf{rec}(x)) \tag{42}$$
$$\mathsf{sent}(\mathsf{up}(x)) \to \mathsf{up}(\mathsf{sent}(x)) \tag{43}$$
$$\mathsf{no}(\mathsf{up}(x)) \to \mathsf{up}(\mathsf{no}(x)) \tag{44}$$
$$\mathsf{top}(\mathsf{rec}(\mathsf{up}(x))) \to \mathsf{top}(\mathsf{rec}(x)) \tag{45}$$
$$\mathsf{top}(\mathsf{sent}(\mathsf{up}(x))) \to \mathsf{top}(\mathsf{rec}(x)) \tag{46}$$
$$\mathsf{top}(\mathsf{no}(\mathsf{up}(x))) \to \mathsf{top}(\mathsf{rec}(x)) \tag{47}$$

Now we prove that every infinite top reduction reaches a term without no, proving the desired liveness property. Applying Thm. 19 for $p = \mathsf{no}(x)$, this can be done by proving termination of $LS(R,p)$, which consists of Rules (38) - (44) and

$$\mathsf{top}(\mathsf{rec}(\mathsf{up}(x))) \to \mathsf{top}(\mathsf{check}(\mathsf{rec}(x))) \tag{48}$$
$$\mathsf{top}(\mathsf{sent}(\mathsf{up}(x))) \to \mathsf{top}(\mathsf{check}(\mathsf{rec}(x))) \tag{49}$$
$$\mathsf{top}(\mathsf{no}(\mathsf{up}(x))) \to \mathsf{top}(\mathsf{check}(\mathsf{rec}(x))) \tag{50}$$
$$\mathsf{check}(\mathsf{up}(x)) \to \mathsf{up}(\mathsf{check}(x)) \tag{51}$$
$$\mathsf{check}(\mathsf{sent}(x)) \to \mathsf{sent}(\mathsf{check}(x)) \tag{52}$$
$$\mathsf{check}(\mathsf{rec}(x)) \to \mathsf{rec}(\mathsf{check}(x)) \tag{53}$$
$$\mathsf{check}(\mathsf{no}(x)) \to \mathsf{no}(\mathsf{check}(x)) \tag{54}$$
$$\mathsf{check}(\mathsf{no}(x)) \to \mathsf{no}(x) \tag{55}$$

Termination is easily proved completely automatically according to our heuristics. First, by respectively choosing Σ' to be $\{\mathsf{no}\}$ and $\{\mathsf{rec}, \mathsf{up}\}$ in Lemma 21, the rules (38), (40), (48), and (50) are removed. According to our heuristics we now apply semantic labelling with the following model: $M = \{0, 1\}$ and

$$\mathsf{top}_M(x) = \mathsf{no}_M(x) = \mathsf{check}_M(x) = 0, \ \ \mathsf{bot}_M = 1,$$
$$\mathsf{rec}_M(x) = \mathsf{sent}_M(x) = \mathsf{up}_M(x) = x$$

for $x \in \{0, 1\}$. It is easily checked that for all 11 rules the left-hand side and the right-hand side yield the same value in this model. Now the labelled system reads

$$
\begin{aligned}
\mathsf{rec}_i(\mathsf{sent}_i(x)) &\to \mathsf{sent}_i(\mathsf{rec}_i(x)) && (39_i)\\
\mathsf{rec}_1(\mathsf{bot}) &\to \mathsf{up}_1(\mathsf{sent}_1(\mathsf{bot})) && (41)\\
\mathsf{rec}_i(\mathsf{up}_i(x)) &\to \mathsf{up}_i(\mathsf{rec}_i(x)) && (42_i)\\
\mathsf{sent}_i(\mathsf{up}_i(x)) &\to \mathsf{up}_i(\mathsf{sent}_i(x)) && (43_i)\\
\mathsf{no}_i(\mathsf{up}_i(x)) &\to \mathsf{up}_0(\mathsf{no}_i(x)) && (44_i)\\
\mathsf{top}_i(\mathsf{sent}_i(\mathsf{up}_i(x))) &\to \mathsf{top}_0(\mathsf{check}_i(\mathsf{rec}_i(x))) && (49_i)\\
\mathsf{check}_i(\mathsf{up}_i(x)) &\to \mathsf{up}_0(\mathsf{check}_i(x)) && (51_i)\\
\mathsf{check}_i(\mathsf{sent}_i(x)) &\to \mathsf{sent}_0(\mathsf{check}_i(x)) && (52_i)\\
\mathsf{check}_i(\mathsf{rec}_i(x)) &\to \mathsf{rec}_0(\mathsf{check}_i(x)) && (53_i)\\
\mathsf{check}_0(\mathsf{no}_i(x)) &\to \mathsf{no}_0(\mathsf{check}_i(x)) && (54_i)\\
\mathsf{check}_0(\mathsf{no}_i(x)) &\to \mathsf{no}_i(x) && (55_i)
\end{aligned}
$$

for $i \in \{0,1\}$. On this labelled TRS we apply Lemma 21: by consecutively choosing $\Sigma' = \{f\}$ for f being top_1, no_1, rec_1, up_1, sent_1, sent_0, and check_0, the rules (49_1), (54_1), (41) and (53_1), (44_1) and (51_1), (52_1), (49_0), and (55_0) and (55_1) are removed. In the remaining system of 11 rules we get termination by the recursive path order using a precedence with

$$\mathsf{check}_0 > \mathsf{rec}_i > \mathsf{sent}_i > \mathsf{up}_i \text{ for } i \in \{0,1\} \quad \text{and} \quad \mathsf{check}_0 > \mathsf{no}_0 > \mathsf{up}_0.$$

6.4 A Shared Resource With Several Waiting Lines

In order to handle more general protocols, our approach can easily be extended to several top symbols of arbitrary arity. Then we have a signature Σ of non-top symbols and a signature Σ_{top} of top symbols with $\Sigma \cap \Sigma_{\mathsf{top}} = \varnothing$. Now *top terms* are terms whose root is from Σ_{top} and below the root there are only variables and symbols from Σ. Let $\mathcal{T}_{\Sigma_{\mathsf{top}}}$ denote the set of ground top terms. Again, a TRS is a *top rewrite system* if every rule is either a top rule (i.e., both sides are top terms) or a non-top rule (i.e., both sides are terms from $\mathcal{T}(\Sigma, \mathcal{V})$).

The sound and complete transformation of Def. 7 can be adapted to such top rewrite systems as follows: We add a new symbol top which was not present in the original signature $\Sigma \cup \Sigma_{\mathsf{top}}$. Moreover, in addition to the symbol check which checks whether its argument is from $\mathcal{T}(\Sigma)$ and contains an instance of p, we use another symbol $\mathsf{check}_{\mathsf{top}}$ which checks whether its argument is from $\mathcal{T}_{\Sigma_{\mathsf{top}}}$ and contains an instance of p. So the top-rules in $L(R,p)$ are $\mathsf{top}((\mathsf{mark}(x)) \to \mathsf{top}(\mathsf{check}_{\mathsf{top}}(x))$ and $\mathsf{top}(\mathsf{found}(x)) \to \mathsf{top}(\mathsf{active}(x))$. Moreover, $L(R,p)$ contains $\mathsf{active}(l) \to \mathsf{mark}(r)$ for all rules $l \to r \in R$ (including the top rules of R). The rule for $\mathsf{check}_{\mathsf{top}}$ is similar to the one for check, but now one requires that the argument starts with a symbol from Σ_{top}. So for every $f \in \Sigma_{\mathsf{top}}$ we have the rules

$$\mathsf{check}_{\mathsf{top}}(f(x_1,\ldots,x_n)) \to f(\mathsf{proper}(x_1),\ldots,\mathsf{check}(x_i),\ldots,\mathsf{proper}(x_n)).$$

The rules for check, match, proper, and start as well as the rules $f(\mathsf{ok}(x_1),\ldots,\mathsf{ok}(x_n)) \to \mathsf{ok}(f(x_1,\ldots,x_n))$ remain unchanged (i.e., in these rules we have $f \in$

Σ). In contrast to that, the rules

$$f(\mathsf{ok}(x_1),\ldots,\mathsf{found}(x_i),\ldots,\mathsf{ok}(x_n)) \to \mathsf{found}(f(x_1,\ldots,x_n)),$$
$$\mathsf{active}(f(x_1,\ldots,x_i,\ldots,x_n)) \to f(x_1,\ldots,\mathsf{active}(x_i),\ldots,x_n),$$
$$f(x_1,\ldots,\mathsf{mark}(x_i),\ldots,x_n) \to \mathsf{mark}(f(x_1,\ldots,x_n))$$

are required for all $f \in \Sigma \cup \Sigma_{\mathsf{top}}$.

For this modified transformation we have a theorem corresponding to Thm. 15: $L(R,p)$ is terminating iff $\to_G$ is terminating on $\mathcal{T}_{\Sigma_{\mathsf{top}}}$. To prove this claim, one needs lemmata corresponding to Lemma 9 – Lemma 14. Lemma 9 and 10 remain unchanged. Lemma 11 is extended by the observation that for all $t \in \mathcal{T}(\Sigma_G)$ we have $\mathsf{check}_{\mathsf{top}}(t) \to^{+}_{L(R,p)} \mathsf{found}(u)$ iff $t = u \in \mathcal{T}_{\Sigma_{\mathsf{top}}}$ and t contains a subterm $p\sigma$. In Lemma 12 (b), we need the additional observation that for all $t, u \in \mathcal{T}(\Sigma_G)$ we have $\mathsf{check}_{\mathsf{top}}(t) \not\to^{+}_{L(R,p)} \mathsf{mark}(u)$. Lemma 13 remains unchanged, but in the proof the precedence used must be updated to $\mathsf{active} > \mathsf{check}_{\mathsf{top}} > \mathsf{check} > \ldots$

The sound transformation of Def. 17 also has to be modified slightly. In order to handle top rewrite systems with many top symbols of arbitrary arity, for every top rule $f(s_1,\ldots,s_n) \to g(t_1,\ldots,t_m)$ in R, $LS(R,p)$ should contain the rules $f(s_1,\ldots,s_n) \to g(t_1,\ldots,\mathsf{check}(t_i),\ldots,t_m)$ for all $1 \le i \le m$. The remaining rules of $LS(R,p)$ are constructed as in Def. 17. The soundness of this transformation can easily be shown as in Thm. 19.

The following case study is an example of a top rewrite system with several non-unary top symbols. We modify the example in Sect. 6.1 by considering two waiting lines of processes which want to gain access to a shared resource. Again we want to prove the liveness property that eventually all old processes (which are already in the lines) will be served. This problem is considerably more difficult than the one in Sect. 6.1, since liveness only holds if the lines are synchronized in a suitable way. For instance, if there is no communication at all between the two lines and new processes may freely choose one of the two lines, it can be the case that in one line serving processes goes on forever while in the other line no processes are served at all. If then the non-serving line still contains an old process, the desired liveness property does not hold. On the other hand, every new process should be free to choose any of the two lines. A subtle way to achieve this is to impose the extra requirement that both lines must offer free positions in an alternating way. We represent two waiting lines l_1 and l_2 in one term $\mathsf{top}(l_1,l_2)$. In order to keep track of the information which of the lines is allowed to generate a free position for a new process, instead of one symbol top we will use two symbols top_1 and top_2, where $\mathsf{top}_i(l_1,l_2)$ means that waiting line l_i is allowed to offer a free position for a new process, for $i \in \{1,2\}$. For the behavior inside a waiting line we have exactly the same rules as in Sect. 6.1. We obtain the following top rewrite system where the binary symbols top_1 and top_2 act as the top symbols (i.e., $\Sigma_{\mathsf{top}} = \{\mathsf{top}_1, \mathsf{top}_2\}$).

$$\mathsf{top}_1(\mathsf{free}(x), y) \to \mathsf{top}_2(\mathsf{new}(x), y)$$
$$\mathsf{top}_1(\mathsf{free}(x), y) \to \mathsf{top}_2(x, \mathsf{new}(y))$$

$$\begin{aligned}
\mathsf{top}_2(x,\mathsf{free}(y)) &\to \mathsf{top}_1(\mathsf{new}(x),y)\\
\mathsf{top}_2(x,\mathsf{free}(y)) &\to \mathsf{top}_1(x,\mathsf{new}(y))\\
\mathsf{new}(\mathsf{free}(x)) &\to \mathsf{free}(\mathsf{new}(x))\\
\mathsf{old}(\mathsf{free}(x)) &\to \mathsf{free}(\mathsf{old}(x))\\
\mathsf{new}(\mathsf{serve}) &\to \mathsf{free}(\mathsf{serve})\\
\mathsf{old}(\mathsf{serve}) &\to \mathsf{free}(\mathsf{serve})
\end{aligned}$$

Again we want to prove that in an infinite reduction of ground top terms, after a finite number of steps a term without the symbol old will be achieved. This corresponds to the liveness property that eventually all old processes will be served. Here, $\mathsf{NF}(\mathcal{T}_{\mathsf{top}}) = \{\mathsf{top}_1(\mathsf{serve},\mathsf{free}^n(\mathsf{serve})) \mid n \in \mathbb{N}\} \cup \{\mathsf{top}_2(\mathsf{free}^n(\mathsf{serve}),\mathsf{serve}) \mid n \in \mathbb{N}\} \subseteq G$ and thus, it suffices to prove termination of $\to_G$ on $\mathcal{T}_{\mathsf{top}}$. Applying the sound transformation of Def. 17 for $p = \mathsf{old}(x)$ this can be done by proving termination of the following TRS. Here, we extended the sound transformation to handle several top symbols of arbitrary arity as described above.

$$\begin{aligned}
\mathsf{top}_1(\mathsf{free}(x),y) &\to \mathsf{top}_2(\mathsf{check}(\mathsf{new}(x)),y) && (56)\\
\mathsf{top}_1(\mathsf{free}(x),y) &\to \mathsf{top}_2(\mathsf{new}(x),\mathsf{check}(y)) && (57)\\
\mathsf{top}_1(\mathsf{free}(x),y) &\to \mathsf{top}_2(\mathsf{check}(x),\mathsf{new}(y)) && (58)\\
\mathsf{top}_1(\mathsf{free}(x),y) &\to \mathsf{top}_2(x,\mathsf{check}(\mathsf{new}(y))) && (59)\\
\mathsf{top}_2(x,\mathsf{free}(y)) &\to \mathsf{top}_1(\mathsf{check}(\mathsf{new}(x)),y) && (60)\\
\mathsf{top}_2(x,\mathsf{free}(y)) &\to \mathsf{top}_1(\mathsf{new}(x),\mathsf{check}(y)) && (61)\\
\mathsf{top}_2(x,\mathsf{free}(y)) &\to \mathsf{top}_1(\mathsf{check}(x),\mathsf{new}(y)) && (62)\\
\mathsf{top}_2(x,\mathsf{free}(y)) &\to \mathsf{top}_1(x,\mathsf{check}(\mathsf{new}(y))) && (63)\\
\mathsf{new}(\mathsf{free}(x)) &\to \mathsf{free}(\mathsf{new}(x)) && (64)\\
\mathsf{old}(\mathsf{free}(x)) &\to \mathsf{free}(\mathsf{old}(x)) && (65)\\
\mathsf{new}(\mathsf{serve}) &\to \mathsf{free}(\mathsf{serve}) && (66)\\
\mathsf{old}(\mathsf{serve}) &\to \mathsf{free}(\mathsf{serve}) && (67)\\
\mathsf{check}(\mathsf{free}(x)) &\to \mathsf{free}(\mathsf{check}(x)) && (68)\\
\mathsf{check}(\mathsf{new}(x)) &\to \mathsf{new}(\mathsf{check}(x)) && (69)\\
\mathsf{check}(\mathsf{old}(x)) &\to \mathsf{old}(\mathsf{check}(x)) && (70)\\
\mathsf{check}(\mathsf{old}(x)) &\to \mathsf{old}(x) && (71)
\end{aligned}$$

According to (a), due to Lemma 21, Rule (67) is removed by counting the number of old symbols. Then we apply semantic labelling with the following model according to our heuristics: $M = \{0,1\}$ and

$$\begin{gathered}
\mathsf{top}_{1M}(x) = \mathsf{top}_{2M}(x) = \mathsf{check}_M(x) = \mathsf{old}_M(x) = 0,\\
\mathsf{new}_M(x) = \mathsf{free}_M(x) = x, \quad \mathsf{serve}_M = 1
\end{gathered}$$

for $x \in \{0,1\}$. One checks that M is a model for the TRS. It remains to prove termination of the labelled system consisting of the following 45 rules

$$
\begin{aligned}
\mathsf{top}_{1_{ij}}(\mathsf{free}_i(x), y) &\to \mathsf{top}_{2_{0j}}(\mathsf{check}_i(\mathsf{new}_i(x)), y) && (56_{ij})\\
\mathsf{top}_{1_{ij}}(\mathsf{free}_i(x), y) &\to \mathsf{top}_{2_{i0}}(\mathsf{new}_i(x), \mathsf{check}_j(y)) && (57_{ij})\\
\mathsf{top}_{1_{ij}}(\mathsf{free}_i(x), y) &\to \mathsf{top}_{2_{0j}}(\mathsf{check}_i(x), \mathsf{new}_j(y)) && (58_{ij})\\
\mathsf{top}_{1_{ij}}(\mathsf{free}_i(x), y) &\to \mathsf{top}_{2_{i0}}(x, \mathsf{check}_j(\mathsf{new}_j(y))) && (59_{ij})\\
\mathsf{top}_{2_{ij}}(x, \mathsf{free}_j(y)) &\to \mathsf{top}_{1_{0j}}(\mathsf{check}_i(\mathsf{new}_i(x)), y) && (60_{ij})\\
\mathsf{top}_{2_{ij}}(x, \mathsf{free}_j(y)) &\to \mathsf{top}_{1_{i0}}(\mathsf{new}_i(x), \mathsf{check}_j(y)) && (61_{ij})\\
\mathsf{top}_{2_{ij}}(x, \mathsf{free}_j(y)) &\to \mathsf{top}_{1_{0j}}(\mathsf{check}_i(x), \mathsf{new}_j(y)) && (62_{ij})\\
\mathsf{top}_{2_{ij}}(x, \mathsf{free}_j(y)) &\to \mathsf{top}_{1_{i0}}(x, \mathsf{check}_j(\mathsf{new}_j(y))) && (63_{ij})\\
\mathsf{new}_i(\mathsf{free}_i(x)) &\to \mathsf{free}_i(\mathsf{new}_i(x)) && (64_i)\\
\mathsf{old}_i(\mathsf{free}_i(x)) &\to \mathsf{free}_0(\mathsf{old}_i(x)) && (65_i)\\
\mathsf{new}_1(\mathsf{serve}) &\to \mathsf{free}_1(\mathsf{serve}) && (66)\\
\mathsf{check}_i(\mathsf{free}_i(x)) &\to \mathsf{free}_0(\mathsf{check}_i(x)) && (68_i)\\
\mathsf{check}_i(\mathsf{new}_i(x)) &\to \mathsf{new}_0(\mathsf{check}_i(x)) && (69_i)\\
\mathsf{check}_0(\mathsf{old}_i(x)) &\to \mathsf{old}_0(\mathsf{check}_i(x)) && (70_i)\\
\mathsf{check}_0(\mathsf{old}_i(x)) &\to \mathsf{old}_i(x) && (71_i)
\end{aligned}
$$

for $i, j \in \{0, 1\}$.

As a first step we apply Lemma 21 for the symbol old_1 by which Rule (70_1) is removed. Next we apply Lemma 21 for the symbol $\mathsf{top}_{1_{11}}$ by which Rules $(56_{11}), (57_{11}), (58_{11}), (59_{11})$ are removed. Then we use Lemma 21 for the symbol $\mathsf{top}_{2_{11}}$ by which Rules $(60_{11}), (61_{11}), (62_{11}), (63_{11})$ are removed. Hence in the remaining TRS the symbols $\mathsf{top}_{1_{11}}$ and $\mathsf{top}_{2_{11}}$ do not occur any more.

Next, we count the occurrences of the two symbols $\mathsf{top}_{1_{10}}$ and $\mathsf{top}_{2_{10}}$. In Rules (56_{10}), (58_{10}), (60_{10}), and (62_{10}) the number of $\mathsf{top}_{1_{10}}$ and $\mathsf{top}_{2_{10}}$ strictly decreases, whereas in all other rules this number remains the same. Hence we remove Rules (56_{10}), (58_{10}), (60_{10}), and (62_{10}). In a similar way we remove Rules (57_{01}), (59_{01}), (61_{01}), and (63_{01}) by counting the symbols $\mathsf{top}_{1_{01}}$ and $\mathsf{top}_{2_{01}}$.

Next, we apply Lemma 21 by counting the three symbols free_0, free_1, and new_1. The number of occurrences of these symbols decreases in Rules (56_{0j}), (57_{00}), (58_{00}), (59_{i0}), (60_{0j}), (61_{00}), (62_{00}), (63_{i0}), and (69_1) for all $i, j \in \{0, 1\}$ and it remains the same in all other rules. By dropping the former rules we obtain the following system of just 15 rules.

$$
\begin{aligned}
\mathsf{top}_{1_{10}}(\mathsf{free}_1(x), y) &\to \mathsf{top}_{2_{10}}(\mathsf{new}_1(x), \mathsf{check}_0(y)) && (57_{10})\\
\mathsf{top}_{1_{01}}(\mathsf{free}_0(x), y) &\to \mathsf{top}_{2_{01}}(\mathsf{check}_0(x), \mathsf{new}_1(y)) && (58_{01})\\
\mathsf{top}_{2_{10}}(x, \mathsf{free}_0(y)) &\to \mathsf{top}_{1_{10}}(\mathsf{new}_1(x), \mathsf{check}_0(y)) && (61_{10})\\
\mathsf{top}_{2_{01}}(x, \mathsf{free}_1(y)) &\to \mathsf{top}_{1_{01}}(\mathsf{check}_0(x), \mathsf{new}_1(y)) && (62_{01})\\
\mathsf{new}_i(\mathsf{free}_i(x)) &\to \mathsf{free}_i(\mathsf{new}_i(x)) && (64_i)\\
\mathsf{old}_i(\mathsf{free}_i(x)) &\to \mathsf{free}_0(\mathsf{old}_i(x)) && (65_i)\\
\mathsf{new}_1(\mathsf{serve}) &\to \mathsf{free}_1(\mathsf{serve}) && (66)\\
\mathsf{check}_i(\mathsf{free}_i(x)) &\to \mathsf{free}_0(\mathsf{check}_i(x)) && (68_i)\\
\mathsf{check}_0(\mathsf{new}_0(x)) &\to \mathsf{new}_0(\mathsf{check}_0(x)) && (69_0)\\
\mathsf{check}_0(\mathsf{old}_0(x)) &\to \mathsf{old}_0(\mathsf{check}_0(x)) && (70_0)\\
\mathsf{check}_0(\mathsf{old}_i(x)) &\to \mathsf{old}_i(x) && (71_i)
\end{aligned}
$$

for $i \in \{0,1\}$. Now Lemma 21 is no longer applicable. Termination of this system can be proved automatically with dependency pairs. The resulting dependency pairs on cycles of the estimated dependency graph are

$$\begin{aligned}
\mathsf{Top}_{1_{10}}(\mathsf{free}_1(x), y) &\to \mathsf{Top}_{2_{10}}(\mathsf{new}_1(x), \mathsf{check}_0(y)) && (72)\\
\mathsf{Top}_{1_{01}}(\mathsf{free}_0(x), y) &\to \mathsf{Top}_{2_{01}}(\mathsf{check}_0(x), \mathsf{new}_1(y)) && (73)\\
\mathsf{Top}_{2_{10}}(x, \mathsf{free}_0(y)) &\to \mathsf{Top}_{1_{10}}(\mathsf{new}_1(x), \mathsf{check}_0(y)) && (74)\\
\mathsf{Top}_{2_{01}}(x, \mathsf{free}_1(y)) &\to \mathsf{Top}_{1_{01}}(\mathsf{check}_0(x), \mathsf{new}_1(y)) && (75)\\
\mathsf{New}_i(\mathsf{free}_i(x)) &\to \mathsf{New}_i(x)\\
\mathsf{Old}_i(\mathsf{free}_i(x)) &\to \mathsf{Old}_i(x)\\
\mathsf{Check}_i(\mathsf{free}_i(x)) &\to \mathsf{Check}_i(x)\\
\mathsf{Check}_0(\mathsf{new}_0(x)) &\to \mathsf{Check}_0(x)\\
\mathsf{Check}_0(\mathsf{old}_0(x)) &\to \mathsf{Check}_0(x)
\end{aligned}$$

for $i \in \{0,1\}$. We use an argument filtering which eliminates the first arguments of $\mathsf{Top}_{1_{10}}$, $\mathsf{Top}_{2_{10}}$, $\mathsf{top}_{1_{10}}$, and $\mathsf{top}_{2_{10}}$ and which eliminates the second argument of $\mathsf{Top}_{1_{01}}$, $\mathsf{Top}_{2_{01}}$, $\mathsf{top}_{1_{01}}$, and $\mathsf{top}_{2_{01}}$. So every term $\mathsf{Top}_{1_{10}}(s,t)$ is replaced by $\mathsf{Top}_{1_{10}}(t)$, etc. Moreover, the argument filtering maps check_0 to its argument, i.e., every term $\mathsf{check}_0(t)$ is replaced by t. Then the resulting constraints are satisfied by the recursive path order using a precedence where all Top-symbols are equal, all top-symbols are equal, and also new_i, free_i, old_i, and check_1 are considered equal for $i \in \{0,1\}$. More precisely, the dependency pairs (72) and (75) and all rules are weakly decreasing and the remaining dependency pairs are strictly decreasing. This is sufficient, since only one dependency pair on each cycle has to be strictly decreasing and (72) and (75) do not form cycles on their own. (Dependency pair (72) is only on a cycle with (74) and (75) is only on a cycle with (73).) This concludes the termination proof of the TRS $LS(R,p)$, and therefore the liveness property of the system with two waiting lines is proved.

Similar to the above example with two waiting lines, a corresponding liveness property can be proved for any such system with n waiting lines (where $n \geq 1$). The n^2 top rules of the TRS R would be

$$\begin{aligned}
\mathsf{top}_1(\mathsf{free}(x_1), x_2, \ldots, x_n) &\to \mathsf{top}_2(\mathsf{new}(x_1), \ldots, x_n)\\
&\ldots\\
\mathsf{top}_1(\mathsf{free}(x_1), x_2, \ldots, x_n) &\to \mathsf{top}_2(x_1, \ldots, \mathsf{new}(x_n))\\
&\ldots\\
\mathsf{top}_2(x_1, \mathsf{free}(x_2), \ldots, x_n) &\to \mathsf{top}_3(\mathsf{new}(x_1), \ldots, x_n)\\
&\ldots\\
\mathsf{top}_2(x_1, \mathsf{free}(x_2), \ldots, x_n) &\to \mathsf{top}_3(x_1, \ldots, \mathsf{new}(x_n))\\
&\ldots\\
\mathsf{top}_n(x_1, \ldots, x_{n-1}, \mathsf{free}(x_n)) &\to \mathsf{top}_1(\mathsf{new}(x_1), \ldots, x_n)\\
&\ldots\\
\mathsf{top}_n(x_1, \ldots, x_{n-1}, \mathsf{free}(x_n)) &\to \mathsf{top}_1(x_1, \ldots, \mathsf{new}(x_n))
\end{aligned}$$

In order to prove that this TRS satisfies the liveness property w.r.t. the term $p = \mathsf{old}(x)$, one can proceed exactly as in the case where $n = 2$. Thus, one first uses the transformation of Def. 17. For example, the top_1-rules of $LS(R, p)$ are

$$\begin{array}{rcl}
\mathsf{top}_1(\mathsf{free}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_2(\mathsf{check}(\mathsf{new}(x_1)), \ldots, x_n) \\
& \ldots & \\
\mathsf{top}_1(\mathsf{free}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_2(\mathsf{new}(x_1), \ldots, \mathsf{check}(x_n)) \\
& \ldots & \\
\mathsf{top}_1(\mathsf{free}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_2(\mathsf{check}(x_1), \ldots, \mathsf{new}(x_n)) \\
& \ldots & \\
\mathsf{top}_1(\mathsf{free}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_2(x_1, \ldots, \mathsf{check}(\mathsf{new}(x_n)))
\end{array}$$

Now according to Step (a), we remove the rule $\mathsf{old}(\mathsf{serve}) \to \mathsf{free}(\mathsf{serve})$ (Rule (67)) by counting the occurrences of old. Then we apply semantic labelling with the same model as before, where $\mathsf{top}_{j\,M}(x) = 0$ for all j. In the labelled system, the rules for top_1 are changed into

$$\begin{array}{rcl}
\mathsf{top}_{1_{i_1,\ldots,i_n}}(\mathsf{free}_{i_1}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_{2_{0,\ldots,i_n}}(\mathsf{check}_{i_1}(\mathsf{new}_{i_1}(x_1)), \ldots, x_n) \\
& \ldots & \\
\mathsf{top}_{1_{i_1,\ldots,i_n}}(\mathsf{free}_{i_1}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_{2_{i_1,\ldots,0}}(\mathsf{new}_{i_1}(x_1), \ldots, \mathsf{check}_{i_n}(x_n)) \\
& \ldots & \\
\mathsf{top}_{1_{i_1,\ldots,i_n}}(\mathsf{free}_{i_1}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_{2_{0,\ldots,i_n}}(\mathsf{check}_{i_1}(x_1), \ldots, \mathsf{new}_{i_n}(x_n)) \\
& \ldots & \\
\mathsf{top}_{1_{i_1,\ldots,i_n}}(\mathsf{free}_{i_1}(x_1), x_2, \ldots, x_n) & \to & \mathsf{top}_{2_{i_1,\ldots,0}}(x_1, \ldots, \mathsf{check}_{i_n}(\mathsf{new}_{i_n}(x_n)))
\end{array}$$

for all $i_1, \ldots, i_n \in \{0, 1\}$, and we obtain similar rules for the other symbols. Thus, the labelled system contains $2^n * n^3$ top rules. Now Lemma 21 is applied repeatedly as in the case where $n = 2$. First, $\mathsf{check}_0(\mathsf{old}_1(x)) \to \mathsf{old}_0(\mathsf{check}_1(x))$ (Rule (70_1)) is removed by counting old_1. Then Lemma 21 is used to delete all top rules where the label of the left-hand side differs from the label of the right-hand side. Next, by counting free_0, free_1, and new_1, all remaining top rules can be eliminated except those where the right-hand side contains new_1 and check_0. So the only remaining labelled top_1-rules are

$$\begin{array}{ll}
\mathsf{top}_{1_{1,0,\ldots,i_n}}(\mathsf{free}_1(x_1), x_2, \ldots, x_n) & \to \mathsf{top}_{2_{1,0,\ldots,i_n}}(\mathsf{new}_1(x_1), \mathsf{check}_0(x_2), \ldots, x_n) \\
& \ldots \\
\mathsf{top}_{1_{1,\ldots,0}}(\mathsf{free}_1(x_1), \ldots, x_n) & \to \mathsf{top}_{2_{1,\ldots,0}}(\mathsf{new}_1(x_1), \ldots, \mathsf{check}_0(x_n)) \\
& \ldots \\
\mathsf{top}_{1_{0,\ldots,1}}(\mathsf{free}_0(x_1), \ldots, x_n) & \to \mathsf{top}_{2_{0,\ldots,1}}(\mathsf{check}_0(x_1), \ldots, \mathsf{new}_1(x_n)) \\
& \ldots \\
\mathsf{top}_{1_{i_1,\ldots,0,1}}(\mathsf{free}_{i_1}(x_1), ..., x_{n-1}, x_n) & \to \mathsf{top}_{2_{i_1,\ldots,0,1}}(x_1, ..., \mathsf{check}_0(x_{n-1}), \mathsf{new}_1(x_n))
\end{array}$$

Termination of this TRS can be proved with dependency pairs. For all Top- or top-symbols we use an argument filtering which removes the argument positions labelled with 1. Moreover, the argument filtering maps check_0 to its argument.

Now the resulting constraints are satisfied by the recursive path order where all Top-symbols are equal, all top-symbols are equal, and also new_i, free_i, old_i, and check_1 are considered equal in the precedence.

More precisely, all dependency pairs with free_0 in their left-hand sides are strictly decreasing and all other dependency pairs are weakly decreasing. This is sufficient, since every cycle of dependency pairs contains a pair with free_0 in its left-hand side. Otherwise, there would be a cycle containing dependency pairs of the form $\mathsf{top}_{j_{i_1,\ldots,i_{j-1},1,i_{j+1},\ldots,i_n}}(x_1,\ldots,x_{j-1},\mathsf{free}_1(x_j),x_{j+1},\ldots,x_n) \to \ldots$ for all $1 \leq j \leq n$ in the cycle. Since the labels of the tuple symbols in a cycle do not change, this would imply $i_1 = \ldots = i_n = 1$. However, symbols with such a label are no longer present in the current TRS. Hence we proved the liveness property for the network with n waiting lines for arbitrary $n \geq 1$.

7 Conclusion

In this paper, we showed how to relate liveness and termination of term rewriting. We presented a sound and complete transformation such that liveness holds iff the transformed TRS is terminating. By a simpler sound transformation and by refining techniques for proving termination of TRSs we developed an approach to verify liveness properties mechanically. We discussed extensions of our approach and demonstrated its applicability on case studies of liveness in networks of processes.

References

1. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
3. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
4. A. Bouajjani. Languages, rewriting systems, and verification of infinite-state systems. In *Proc. ICALP '01*, volume 2076 of *Lecture Notes in Computer Science*, pages 24–39, 2001.
5. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.
6. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1,2):39–72, 2001.
7. J. Giesl and A. Middeldorp. Transforming context-sensitive rewrite systems. In *Proc. 10th RTA*, volume 1631 of *Lecture Notes in Computer Science*, pages 271–285, 1999.
8. J. Giesl and A. Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 2003. To appear. Preliminary extended version appeared in Technical Report AIB-2002-02, RWTH Aachen, Germany. Available from `http://aib.informatik.rwth-aachen.de`.
9. L. Lamport. A new solution to Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, 1974.
10. H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.
11. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.

Aachener Informatik-Berichte

This is a list of recent technical reports. To obtain copies of technical reports please consult http://aib.informatik.rwth-aachen.de/ or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

95-11 *	M. Staudt / K. von Thadden: Subsumption Checking in Knowledge Bases
95-12 *	G.V. Zemanek / H.W. Nissen / H. Hubert / M. Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology
95-13 *	M. Staudt / M. Jarke: Incremental Maintenance of Externally Materialized Views
95-14 *	P. Peters / P. Szczurko / M. Jeusfeld: Business Process Oriented Information Management: Conceptual Models at Work
95-15 *	S. Rams / M. Jarke: Proceedings of the Fifth Annual Workshop on Information Technologies & Systems
95-16 *	W. Hans / St. Winkler / F. Sáenz: Distributed Execution in Functional Logic Programming
96-1 *	Jahresbericht 1995
96-2	M. Hanus / Chr. Prehofer: Higher-Order Narrowing with Definitional Trees
96-3 *	W. Scheufele / G. Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates
96-4	K. Pohl: PRO-ART: Enabling Requirements Pre-Traceability
96-5	K. Pohl: Requirements Engineering: An Overview
96-6 *	M. Jarke / W. Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools
96-7	O. Chitil: The ς-Semantics: A Comprehensive Semantics for Functional Programs
96-8 *	S. Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics
96-9	M. Hanus (Ed.): Proceedings of the Poster Session of ALP'96 — Fifth International Conference on Algebraic and Logic Programming
96-10	R. Conradi / B. Westfechtel: Version Models for Software Configuration Management
96-11 *	C. Weise / D. Lenzkes: A Fast Decision Algorithm for Timed Refinement
96-12 *	R. Dömges / K. Pohl / M. Jarke / B. Lohmann / W. Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models
96-13 *	K. Pohl / R. Klamma / K. Weidenhaupt / R. Dömges / P. Haumer / M. Jarke: A Framework for Process-Integrated Tools

96-14 * R. Gallersdörfer / K. Klabunde / A. Stolz / M. Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996
96-15 * H. Schimpe / M. Staudt: VAREX: An Environment for Validating and Refining Rule Bases
96-16 * M. Jarke / M. Gebhardt, S. Jacobs, H. Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization
96-17 M. Jeusfeld / T. X. Bui: Decision Support Components on the Internet
96-18 M. Jeusfeld / M. Papazoglou: Information Brokering: Design, Search and Transformation
96-19 * P. Peters / M. Jarke: Simulating the impact of information flows in networked organizations
96-20 M. Jarke / P. Peters / M. Jeusfeld: Model-driven planning and design of cooperative information systems
96-21 * G. de Michelis / E. Dubois / M. Jarke / F. Matthes / J. Mylopoulos / K. Pohl / J. Schmidt / C. Woo / E. Yu: Cooperative information systems: a manifesto
96-22 * S. Jacobs / M. Gebhardt, S. Kethers, W. Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality
96-23 * M. Gebhardt / S. Jacobs: Conflict Management in Design
97-01 Jahresbericht 1996
97-02 J. Faassen: Using full parallel Boltzmann Machines for Optimization
97-03 A. Winter / A. Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems
97-04 M. Mohnen / S. Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler
97-05 * S. Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge
97-06 M. Nicola / M. Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries
97-07 P. Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen
97-08 D. Blostein / A. Schürr: Computing with Graphs and Graph Rewriting
97-09 C.-A. Krapp / B. Westfechtel: Feedback Handling in Dynamic Task Nets
97-10 M. Nicola / M. Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases
97-13 M. Mohnen: Optimising the Memory Management of Higher-Order Functional Programs
97-14 R. Baumann: Client/Server Distribution in a Structure-Oriented Database Management System
97-15 G. H. Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms
98-01 * Jahresbericht 1997

98-02 S. Gruner/ M. Nagel / A. Schürr: Fine-grained and Structure-oriented Integration Tools are Needed for Product Development Processes

98-03 S. Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr

98-04 * O. Kubitz: Mobile Robots in Dynamic Environments

98-05 M. Leucker / St. Tobies: Truth — A Verification Platform for Distributed Systems

98-07 M. Arnold / M. Erdmann / M. Glinz / P. Haumer / R. Knoll / B. Paech / K. Pohl / J. Ryser / R. Studer / K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects

98-08 * H. Aust: Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen

98-09 * Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien

98-10 * M. Nicola / M. Jarke: Performance Modeling of Distributed and Replicated Databases

98-11 * A. Schleicher / B. Westfechtel / D. Jäger: Modeling Dynamic Software Processes in UML

98-12 * W. Appelt / M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web

98-13 K. Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation

99-01 * Jahresbericht 1998

99-02 * F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version

99-03 * R. Gallersdörfer / M. Jarke / M. Nicola: The ADR Replication Manager

99-04 M. Alpuente / M. Hanus / S. Lucas / G. Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing

99-07 Th. Wilke: CTL+ is exponentially more succinct than CTL

99-08 O. Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures

2000-01 * Jahresbericht 1999

2000-02 Jens Vöge / Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games

2000-04 Andreas Becks / Stefan Sklorz / Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach

2000-05 Mareike Schoop: Cooperative Document Management

2000-06 Mareike Schoop / Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling

2000-07 * Markus Mohnen / Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages

2000-08 Thomas Arts / Thomas Noll: Verifying Generic Erlang Client-Server Implementations
2001-01 * Jahresbericht 2000
2001-02 Benedikt Bollig / Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03 Thierry Cachat: The power of one-letter rational languages
2001-04 Benedikt Bollig / Martin Leucker / Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05 Benedikt Bollig / Martin Leucker / Thomas Noll: Regular MSC Languages
2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07 Martin Grohe / Stefan Wöhrle: An Existential Locality Theorem
2001-08 Mareike Schoop / James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09 Thomas Arts / Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11 Klaus Indermark / Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 * Jahresbericht 2001
2002-02 Jürgen Giesl / Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03 Benedikt Bollig / Martin Leucker / Thomas Noll: Generalised Regular MSC Languages
2002-04 Jürgen Giesl / Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05 Horst Lichter / Thomas von der Maßen / Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09 Markus Mohnen: Interfaces with Default Implementations in Java
2002-10 Martin Leucker: Logics for Mazurkiewicz traces

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.