AAECC **12**, 39–72 (2001)

AAECC
Applicable Algebra in Engineering, Communication and Computing
© Springer-Verlag 2001

Verification of Erlang Processes by Dependency Pairs

Jürgen Giesl[1], Thomas Arts[2]

[1] LuFG Informatik II, RWTH Aachen, Ahornstrasse 55, 52074 Aachen, Germany
(e-mail: giesl@informatik.rwth-aachen.de)
[2] Computer Science Lab., Ericsson Utvecklings AB, Box 1505, 125 25 Älvsjö, Sweden
(e-mail: thomas@cslab.ericsson.se)

Received: October 6, 1999

Abstract. Erlang is a functional programming language developed by Ericsson Telecom, which is particularly well suited for implementing concurrent processes. In this paper we show how methods from the area of term rewriting are presently used at Ericsson. To verify properties of processes, such a property is transformed into a termination problem of a conditional term rewriting system (CTRS). Subsequently, this termination proof can be performed automatically using *dependency pairs*.

The paper illustrates how the dependency pair technique can be applied for termination proofs of *conditional* TRSs. Secondly, we present three refinements of this technique, viz. *narrowing*, *rewriting*, and *instantiating dependency pairs*. These refinements are not only of use in the industrial applications sketched in this paper, but they are generally applicable to arbitrary (C)TRSs. Thus, in this way dependency pairs can be used to prove termination of even more (C)TRSs automatically.

Keywords: Verification, Distributed processes, Rewriting, Termination.

1 Introduction

In a patent application [24], Ericsson developed a protocol for a query lookup in a distributed database. In several products of Ericsson, for example their newer telecommunication switches, this database plays a key role in the re-

This work was partially supported by the DFG under grant GI 274/4-1.

covery after a shutdown or crash of the system. Clearly, this critical part of the software should be trustworthy. This paper originates from an attempt to verify this protocol's implementation written in Erlang. To save the amount of work and to increase reliability, the aim was to perform as much as possible of this verification automatically. Model checking techniques were not applicable, since the properties to be proved require the consideration of the infinite state space of the processes. A user guided approach based on theorem proving by a specialized proof checking tool was successful, but very labour intensive [1]. We describe two of the properties which had to be verified in Sect. 2 and Sect. 7, respectively, and we show that they can be represented as non-trivial termination problems of CTRSs.

In general, proving termination of CTRSs is considerably more difficult than showing termination of unconditional TRSs. Therefore, standard techniques (see e.g. [14, 18, 31]) fail with the termination proofs required for the protocol verification described above. Moreover, due to the complexity and the safety requirements arising with practical applications in industry, a high degree of automation is desirable for the termination proofs required. These reasons motivate why we chose to apply the *dependency pair* technique [2, 3, 5, 8] (i.e., the currently most powerful termination proof method that is amenable to automation). However, it turned out that (without further extensions) even the dependency pair technique could not perform the required termination proofs automatically.

In Sect. 3 we show that termination problems of CTRSs can be reduced to termination problems of unconditional TRSs. After recapitulating the basic notions of dependency pairs in Sect. 4, we present three important extensions, viz. *narrowing* (Sect. 5), *rewriting* (Sect. 6), and *instantiating* dependency pairs (Sect. 7), which are particularly useful in the context of CTRSs. With these refinements, the dependency pair approach could solve the termination problems automatically.

2 A Process Verification Problem

We have to prove properties of processes in a network. A process P_n receives messages from a process P_{n-1} that consist of a list of data items and an integer $\mathtt{M}$. For every item in the list, process P_n computes a new list of data items. For example, the data items could be telephone numbers and the process could generate a list of calls to that number on a certain date. The resulting list may have arbitrary length, including zero. The integer $\mathtt{M}$ in the message indicates how many items of the newly computed list should be sent to the next process P_{n+1}. The restriction on the number of items that may be sent is imposed for practical optimization reasons.

Of course, process P_n may have computed more than $\mathtt{M}$ new items and in that case, it stores the remaining answers in an accumulator (implemented by

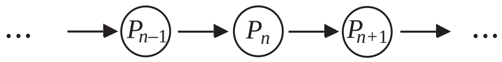

Fig. 1. Process P_n in a network

an extra argument `Store` of the process). However, whenever it has sent the first `M` items to the next process P_{n+1}, process P_n may receive a new message from P_{n-1}. To respond to the new message, it first checks whether its store already contains at least `M` items. In this case, it sends the first `M` items from its store to P_{n+1} and depending on the incoming message, probably some new items are computed afterwards. Otherwise, if the store contains fewer than `M` items, then process P_{n+1} has to wait until the new items are computed. After this computation, the first `M` items from the newly obtained item list and the store are sent to P_{n+1}. Again, those items that exceed the limit `M` are stored in the process accumulator. Finally, in order to empty the store, process P_{n-1} repeatedly sends the empty list to process P_n. In the end, so is the claim, process P_n will send the empty list as well.

We describe how we are able to formally verify this claim with a high degree of automation. The Erlang code executed by the processes is given below (to save space, the code for obvious library functions like `app` and `leq` is not presented).

```
process(NextPid,Store) ->
 receive
    {Items,M} ->
       case leq(M,length(Store)) of
          true ->
             {ToSend,ToStore} = split(M,Store),
             NextPid!{ToSend,M},
             process(NextPid,app(map_f(self(),Items),
                                          ToStore));
          false ->
             {ToSend,ToStore} =
                split(M,app(map_f(self(),Items),
                                  Store)),
             NextPid!{ToSend,M},
             process(NextPid,ToStore)
       end
 end.

map_f(Pid,nil) -> nil;
map_f(Pid,cons(H,T)) -> app(f(Pid,H),map_f(Pid,T)).
```

For a list `L`, `split(M,L)` returns a pair of lists {L$_1$,L$_2$} where L$_1$ contains the first `M` elements (or `L` if its length is shorter than `M`) and L$_2$ contains the rest of `L`. The command ‘`!`’ denotes the sending of data and `NextPid!{ToSend,M}` stands for sending the items `ToSend` and the integer `M` to the process with the identifier `NextPid`. A process can obtain its own identifier by calling the function `self()`. For every item in the list `Items`, the function `map_f(Pid, Items)` computes new data items by means of the function `f(Pid,Item)`. So the actual computation that `f` performs depends on the process identifier `Pid`. Hence, to compute new data items for the incoming `Items`, a process P_n has to pass its own identifier to the function `map_f`, i.e., it calls `map_f(self(),Items)`.

Note that a process itself is not a terminating function: in fact, it has been designed to be non-terminating. Our aim is not to prove its termination, but to verify a certain property, which can be expressed in terms of termination. As part of the correctness proof of the software, we have to prove that if a process P_n continuously receives the message {`nil,M`} for any integer `M`, then eventually the process will send the message {`nil,M`} as well. This property must hold independent of the value of the store and of the way in which new data items are generated from given ones. Therefore, `f` has been left unspecified, i.e., `f` may be any terminating function which returns a list of arbitrary length.

The framework of term rewriting [10, 17] is very useful for this verification. We prove the desired property by constructing a CTRS containing a binary function process whose arguments represent the stored data items `Store` and the integer `M` sent in the messages. In this example, we may abstract from the process communication. Thus, the Erlang function `self()` becomes a constant and we drop the send command (`!`) and the argument `NextPid` in the CTRS. Since we assume that the process constantly receives the message {`nil,M`}, we hard-code it into the CTRS. Thus, the variable `Items` is replaced by nil. As we still want to reason about the variable `M`, we added it to the arguments of the process. To model the function `split` (which returns a *pair* of lists) in the CTRS, we use separate functions fstsplit and sndsplit for the two components of `split`'s result. Thus, fstsplit$(m, store)$ results in the first m elements of the store and sndsplit$(m, store)$ results in all but the first m elements of the store. Now the idea is to force the function process to terminate if `ToSend` is the empty list nil. So we only continue the computation if application of the function empty to the result of fstsplit yields false. Thus, if all evaluations w.r.t. this CTRS terminate, then the original process eventually outputs the demanded value. As usual, the semantics of a rule '$s_1 \to^* t_1, s_2 \to^* t_2 \mid l \to r$' is that a redex $l\sigma$ may only be reduced to $r\sigma$ if $s_1\sigma$ reduces to $t_1\sigma$ and $s_2\sigma$ reduces to $t_2\sigma$ (i.e., the vertical bar | separates the conditions from the actual rule).

$$\begin{aligned}
&\mathsf{leq}(m, \mathsf{length}(store)) \to^* \mathsf{true},\\
&\mathsf{empty}(\mathsf{fstsplit}(m, store)) \to^* \mathsf{false} \quad \mid\\
&\quad \mathsf{process}(store, m)\\
&\quad \to \mathsf{process}(\mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), \mathsf{sndsplit}(m, store)), m) \qquad (1)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{leq}(m, \mathsf{length}(store)) \to^* \mathsf{false},\\
&\mathsf{empty}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), store))) \to^* \mathsf{false} \mid\\
&\quad \mathsf{process}(store, m)\\
&\qquad \to \mathsf{process}(\mathsf{sndsplit}(m, \mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), store)), m)
\end{aligned} \tag{2}$$

The auxiliary Erlang functions as well as the functions for empty, fstsplit, and sndsplit are straightforwardly expressed by unconditional rewrite rules.

$$\begin{aligned}
\mathsf{fstsplit}(0, x) &\to \mathsf{nil}\\
\mathsf{fstsplit}(\mathsf{s}(n), \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{fstsplit}(\mathsf{s}(n), \mathsf{cons}(h, t)) &\to \mathsf{cons}(h, \mathsf{fstsplit}(n, t))\\
\mathsf{sndsplit}(0, x) &\to x\\
\mathsf{sndsplit}(\mathsf{s}(n), \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{sndsplit}(\mathsf{s}(n), \mathsf{cons}(h, t)) &\to \mathsf{sndsplit}(n, t)\\
\mathsf{empty}(\mathsf{nil}) &\to \mathsf{true}\\
\mathsf{empty}(\mathsf{cons}(h, t)) &\to \mathsf{false}\\
\mathsf{leq}(0, m) &\to \mathsf{true}\\
\mathsf{leq}(\mathsf{s}(n), 0) &\to \mathsf{false}\\
\mathsf{leq}(\mathsf{s}(n), \mathsf{s}(m)) &\to \mathsf{leq}(n, m)\\
\mathsf{length}(\mathsf{nil}) &\to 0\\
\mathsf{length}(\mathsf{cons}(h, t)) &\to \mathsf{s}(\mathsf{length}(t))\\
\mathsf{app}(\mathsf{nil}, x) &\to x\\
\mathsf{app}(\mathsf{cons}(h, t), x) &\to \mathsf{cons}(h, \mathsf{app}(t, x))\\
\mathsf{map\_f}(pid, \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{map\_f}(pid, \mathsf{cons}(h, t)) &\to \mathsf{app}(\mathsf{f}(pid, h), \mathsf{map\_f}(pid, t))
\end{aligned}$$

The rules for the Erlang function `f` are not specified, since we have to verify the desired property for *any* terminating function `f`. However, as Erlang has an eager (call-by-value) evaluation strategy, if a terminating Erlang function `f` is straightforwardly transformed into a (C)TRS (such as the above library functions), then any evaluation w.r.t. these rules is finite. Now to prove the desired property of the Erlang process, we have to show that the whole CTRS with all its extra rules for the auxiliary functions only permits finite evaluations.

The construction of the above CTRS is rather straightforward, but it presupposes an understanding of the program and the verification problem and therefore it can hardly be mechanized. But after obtaining the CTRS, the proof that any evaluation w.r.t. this CTRS is finite should be done automatically.

In this paper we describe an extension of the dependency pair technique which can perform such automatic proofs. Moreover, this extension is of general use for termination proofs of TRSs and CTRSs. Hence, our results significantly increase the class of systems where termination can be shown mechanically.

3 Termination of Conditional Term Rewriting Systems

A CTRS is a TRS where conditions $s_1 = t_1, \ldots, s_n = t_n$ may be added to rewrite rules $l \to r$. In this paper, we restrict ourselves to CTRSs where all variables in the conditions s_i, t_i also occur in l. Depending on the interpretation of the equality sign in the conditions, different rewrite relations can be associated with a CTRS, cf. e.g. [11, 12, 15, 16, 20, 22, 23, 26, 27, 29, 32]. In our verification example, we transformed the problem into an *oriented* CTRS [32], where the equality signs in conditions of rewrite rules are interpreted as reachability ($\to^*$). Thus, we denote rewrite rules by

$$s_1 \to^* t_1, \ldots, s_n \to^* t_n \;\mid\; l \to r. \tag{3}$$

In fact, we even have a *normal* CTRS, because all t_i are ground normal forms w.r.t. the TRS which results from dropping all conditions.

A reduction of $C[l\sigma]$ to $C[r\sigma]$ with rule (3) is only possible if $s_i\sigma$ reduces to $t_i\sigma$ for all $1 \leq i \leq n$. Formally, the rewrite relation $\to_{\mathcal{R}}$ of a CTRS $\mathcal{R}$ can be defined as $\to_{\mathcal{R}} = \bigcup_{j \geq 0} \to_{\mathcal{R}_j}$, where

$$\begin{aligned} \mathcal{R}_0 &= \emptyset \quad \text{and} \\ \mathcal{R}_{j+1} &= \bigcup_{\text{`}s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r\text{'} \in \mathcal{R}} \{l\sigma \to r\sigma \mid s_i\sigma \to^*_{\mathcal{R}_j} t_i\sigma \text{ for all } 1 \leq i \leq n\}, \end{aligned}$$

cf. e.g. [23, 29].

A CTRS $\mathcal{R}$ is *terminating* iff $\to_{\mathcal{R}}$ is well founded. But termination is not enough to ensure that every evaluation with a CTRS is finite. For example, assume that evaluation of the condition $\mathsf{leq}(m, \mathsf{length}(store))$ in our CTRS would require the reduction of $\mathsf{process}(store, m)$. Then evaluation of $\mathsf{process}(store, m)$ would yield an infinite computation. Nevertheless, $\mathsf{process}(store, m)$ could not be rewritten further and thus, the CTRS would be terminating. But in this case, the desired property would *not* hold for the original Erlang process, because this would correspond to a deadlock situation where no messages are sent at all.

For that reason, instead of *termination* one is often much more interested in *decreasing* CTRSs [15]. In this paper, we use a slightly modified notion of decreasingness, because in our evaluation strategy conditions are checked from left to right, cf. [33]. Thus, the i-th condition $s_i \to^* t_i$ is only checked if all previous conditions $s_j \to^* t_j$ for $1 \leq j < i$ hold.

Definition 1 (Left-Right Decreasing) *A CTRS $\mathcal{R}$ is* left-right decreasing *if there exists a well-founded relation $>$ containing the rewrite relation $\to_{\mathcal{R}}$ and the subterm relation $\rhd$ such that $l\sigma > s_i\sigma$ holds for all rules like* (3)*, all $i \in \{1, \ldots, n\}$, and all substitutions σ where $s_j\sigma \to^*_{\mathcal{R}} t_j\sigma$ for all $j \in \{1, \ldots, i-1\}$.*

This definition of left-right decreasingness exactly captures the finiteness of recursive evaluation of terms. (Obviously, decreasingness implies left-right decreasingness, but not vice versa.) Hence, now our aim is to prove that the CTRS corresponding to the Erlang process is left-right decreasing.

A standard approach for proving termination of a CTRS $\mathcal{R}$ is to verify termination of the TRS $\mathcal{R}'$ which results from dropping all conditions (and for decreasingness one has to impose some additional demands). But this approach fails for CTRSs where the conditions are necessary to ensure termination. This also happens in our example, because without the conditions $\mathsf{empty}(\ldots) \to^* \mathsf{false}$ the CTRS is no longer terminating (and thus, not left-right decreasing either).

A solution for this problem is to transform CTRSs into *unconditional* TRSs, cf. [13, 19, 28]. For unconditional rules, let $\mathrm{tr}(l \to r) = \{l \to r\}$. If α is a conditional rule, i.e., α = '$s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r$', we define

$$\begin{aligned}\mathrm{tr}(\alpha) \;&= \{l \to \mathsf{if}_{1,\alpha}(\mathbf{x}, s_1)\} \cup \{\mathsf{if}_{i,\alpha}(\mathbf{x}, t_i) \to \mathsf{if}_{i+1,\alpha}(\mathbf{x}, s_{i+1}) \mid 1 \le i < n\}\\ &\cup \{\mathsf{if}_{n,\alpha}(\mathbf{x}, t_n) \to r\}\end{aligned}$$

where $\mathbf{x}$ is the tuple of all variables in l and the if's are new function symbols. To ease readability, instead of $\mathsf{if}_{i,\alpha}$ we often just write if_m for some $m \in \mathbb{N}$ where if_m is a function symbol which has not yet been used before.

Let $\mathcal{R}^{\mathrm{tr}} = \bigcup_{\alpha \in \mathcal{R}} \mathrm{tr}(\alpha)$. For CTRSs without extra variables, $\mathcal{R}^{\mathrm{tr}}$ is indeed an (unconditional) TRS. (An extension to *deterministic* CTRSs [12] with extra variables is also possible.) The transformation of Rule (1) results in

$$\mathsf{process}(store, m) \to \mathsf{if}_1(store, m, \mathsf{leq}(m, \mathsf{length}(store))) \tag{4}$$

$$\mathsf{if}_1(store, m, \mathsf{true}) \to \mathsf{if}_2(store, m, \mathsf{empty}(\mathsf{fstsplit}(m, store))) \tag{5}$$

$$\begin{aligned}&\mathsf{if}_2(store, m, \mathsf{false})\\ &\quad \to \mathsf{process}(\mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), \mathsf{sndsplit}(m, store)), m).\end{aligned} \tag{6}$$

Now we aim to prove termination of $\mathcal{R}^{\mathrm{tr}}$ instead of $\mathcal{R}$'s left-right decreasingness.

In [19], this transformation is restricted to a limited class of convergent CTRSs. However, in the following we show that for our purpose this restriction is not necessary. In other words, termination of $\mathcal{R}^{\mathrm{tr}}$ indeed implies left-right decreasingness (and thus also termination) of $\mathcal{R}$. Thus, this transformation is a generally applicable technique to reduce the termination problem of CTRSs to a termination problem of unconditional TRSs. (A similar approach was presented in [28] for decreasingness proofs (instead of *left-right* decreasingness) by using a transformation where all conditions of a rule have to be checked in parallel.) We first prove that any reduction with $\mathcal{R}$ can be simulated by $\mathcal{R}^{\mathrm{tr}}$. So in particular, the equational theory of $\mathcal{R}$ is a subset of $\mathcal{R}^{\mathrm{tr}}$'s equational theory.

Lemma 2 *Let q, q' be terms without* if*'s. If $q \to_{\mathcal{R}}^{+} q'$, then $q \to_{\mathcal{R}^{\mathrm{tr}}}^{+} q'$.*

Proof. There must be a $j \in \mathbb{N}$ such that $q \to_{\mathcal{R}_j}^{+} q'$ (j is the *depth* of the reduction). We prove the theorem by induction on the depth and the length of the reduction $q \to_{\mathcal{R}}^{+} q'$ (i.e., we use a lexicographic induction relation).

The reduction has the form $q \to_{\mathcal{R}} p \to_{\mathcal{R}}^{*} q'$ and by the induction hypothesis we know $p \to_{\mathcal{R}^{\mathrm{tr}}}^{*} q'$. Thus, it suffices to prove $q \to_{\mathcal{R}^{\mathrm{tr}}}^{+} p$.

If the reduction $q \to_{\mathcal{R}} p$ is done with an unconditional rule of $\mathcal{R}$, then the conjecture is trivial. Otherwise, we must have $q = C[l\sigma]$, $p = C[r\sigma]$ for some context C and some rule like (3). As the depth of the reductions $s_i\sigma \to_{\mathcal{R}}^{*} t_i\sigma$ is less than the depth of the reduction $q \to_{\mathcal{R}}^{+} q'$, by the induction hypothesis we have $s_i\sigma \to_{\mathcal{R}^{\mathrm{tr}}}^{*} t_i\sigma$. This implies $q \to_{\mathcal{R}^{\mathrm{tr}}}^{+} p$. □

Now the desired result is a direct consequence of Lemma 2.

Corollary 3 (Left-Right Decreasingness of $\mathcal{R}$ and Termination of $\mathcal{R}^{\mathrm{tr}}$) *If $\mathcal{R}^{\mathrm{tr}}$ is terminating, then $\mathcal{R}$ is left-right decreasing (and thus, it is also terminating).*

Proof. It is well known that if $\to_{\mathcal{R}^{\mathrm{tr}}}$ is well founded, then $\to_{\mathcal{R}^{\mathrm{tr}}} \cup \rhd$ is well founded, too (this is a direct consequence of $\to_{\mathcal{R}^{\mathrm{tr}}}$ being closed under context). Hence, the transitive closure $(\to_{\mathcal{R}^{\mathrm{tr}}} \cup \rhd)^{+}$ is well founded, too. By Lemma 2, this relation satisfies all conditions imposed on the relation $>$ in Def. 1. Hence, $\mathcal{R}$ is left-right decreasing. □

The converse of this corollary does not hold. If $\mathcal{R}$ is the CTRS with $\mathsf{a} \to \mathsf{b}$, $\mathsf{f}(\mathsf{a}) \to \mathsf{b}$, and the conditional rule $\mathsf{f}(x) \to^{*} x \mid \mathsf{g}(x) \to \mathsf{g}(\mathsf{a})$, then $\mathsf{g}(\mathsf{a}) \to^{+} \mathsf{g}(\mathsf{a})$ holds in the transformed TRS $\mathcal{R}^{\mathrm{tr}}$, but not in the original CTRS. Thus, the transformed TRS $\mathcal{R}^{\mathrm{tr}}$ is not terminating although the original CTRS $\mathcal{R}$ is left-right decreasing.

However, independently, in the meanwhile this transformation has also been studied by Ohlebusch [30] and he could prove a (restricted) completeness result for this transformation, viz. that left-right decreasingness of $\mathcal{R}$ at least implies *innermost* termination of $\mathcal{R}^{\mathrm{tr}}$. (In [30], our notion of left-right decreasingness is called "quasi-decreasingness".)

In our example, the conditional rule (2) is transformed into three additional unconditional rules. But apart from the if-root symbol of the right-hand side, the first of these rules is identical to (4). Thus, we obtain two overlapping rules in the transformed TRS which correspond to the overlapping conditional rules (1) and (2). However, in the CTRS this critical pair is *infeasible* [15], i.e., the conditions of both rules exclude each other. Thus, our transformation of CTRSs into TRSs sometimes introduces unnecessary rules and overlap.

Therefore, whenever we construct a rule of the form $q \to \mathsf{if}_k(\mathbf{t})$ and there already exists a rule $q \to \mathsf{if}_n(\mathbf{t})$, then we identify if_k and if_n. This does not affect the soundness of our approach, because termination of a TRS where all occurrences of a symbol g are substituted by a symbol f with the same arity always implies termination of the original TRS.[1] Thus, we obtain the additional rules:

[1] This possibility to eliminate unnecessary overlap is an advantage of our transformation compared to the one of [28], where the transformed unconditional TRSs remain overlapping. In practice, proving termination of non-overlapping TRSs is significantly easier, since one may use techniques specifically tailored to *innermost* termination proofs, see below.

$$\begin{aligned}&\mathsf{if}_1(store, m, \mathsf{false}) \to \\ &\qquad \mathsf{if}_3(store, m, \mathsf{empty}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), store)))) \qquad (7)\\ &\mathsf{if}_3(store, m, \mathsf{false}) \to \\ &\qquad \mathsf{process}(\mathsf{sndsplit}(m, \mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), store)), m) \qquad (8)\end{aligned}$$

If termination of a CTRS depends on its conditions, then in general termination of the transformed TRS can only be shown if one examines which terms may follow each other in a reduction. However, in the classical approaches based on simplification orderings (cf. e.g. [14, 31]), such considerations do not take place. Hence, they fail in proving the termination of (4)–(8). For this reason, such transformations into unconditional TRSs have rarely been applied for termination (or decreasingness) proofs of CTRSs. However, we will demonstrate that with the *dependency pair* approach this transformation is very useful.

To verify our original goal, we now have to prove termination of the transformed TRS which consists of (4)–(8), the rules for all auxiliary (library) functions from Sect. 2, and the (unknown) rules for the unspecified function f. Note that if an auxiliary Erlang function is straightforwardly transformed into a TRS, then this TRS is non-overlapping. Thus, we assume that all possible rules for the unspecified function f are non-overlapping as well. Then it is sufficient just to prove *innermost* termination of the resulting TRS, since innermost termination of non-overlapping systems implies their termination, cf. e.g. [21]. In order to apply verification on a large scale, the aim is to perform such proofs automatically.

In the rest of the paper we present some extensions of the dependency pair technique that make this possible. The dependency pair technique (including these extensions) has been implemented in a tool written in Erlang which provides both a user friendly interface for manual applications of dependency pairs and the possibility to perform fully automatic termination proofs of TRSs using dependency pairs [9]. See [4] for a collection of benchmarks to demonstrate the power of the dependency pair approach.

4 Dependency Pairs

Dependency pairs allow the use of existing methods like simplification orderings for automated termination and innermost termination proofs where they were not applicable before. In this section we briefly recapitulate the basic concepts of this approach and we present the theorems that we need for the rest of the paper. For further details and explanations see [3, 5, 8].

In contrast to the standard approaches for termination proofs, which compare left and right-hand sides of rules, we only examine those subterms that are responsible for starting new reductions. For that purpose we concentrate on the subterms in the right-hand sides of rules that have a defined[2] root symbol, because these are the only terms a rewrite rule can ever be applied to.

More precisely, for every rule $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ (where f and g are defined symbols), we compare the argument tuples $s_1, \ldots, s_n$ and $t_1, \ldots, t_m$. To avoid the handling of tuples, for every defined symbol f we introduce a fresh *tuple* symbol F. To ease readability, we assume that the original signature consists of lower case function symbols only, whereas the tuple symbols are denoted by the corresponding upper case symbols. Now instead of the tuples $s_1, \ldots, s_n$ and $t_1, \ldots, t_m$ we compare the *terms* $F(s_1, \ldots, s_n)$ and $G(t_1, \ldots, t_m)$.

Definition 4 (Dependency Pair) *Let $\mathcal{R}$ be a TRS. If $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ is a rule of $\mathcal{R}$ and g is a defined symbol, then $\langle F(s_1, \ldots, s_n), G(t_1, \ldots, t_m)\rangle$ is a* dependency pair *of $\mathcal{R}$.*

For the rules (4)–(8), (besides others) we obtain the following dependency pairs.

$$\langle \mathsf{PROCESS}(store, m), \mathsf{IF}_1(store, m, \mathsf{leq}(m, \mathsf{length}(store)))\rangle \tag{9}$$

$$\langle \mathsf{IF}_1(store, m, \mathsf{true}), \mathsf{IF}_2(store, m, \mathsf{empty}(\mathsf{fstsplit}(m, store)))\rangle \tag{10}$$

$$\langle \mathsf{IF}_2(store, m, \mathsf{false}), \mathsf{PROCESS}(\mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), \mathsf{sndsplit}(m, store)), m)\rangle \tag{11}$$

$$\begin{aligned}&\langle \mathsf{IF}_1(store, m, \mathsf{false}),\\ &\qquad \mathsf{IF}_3(store, m, \mathsf{empty}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), store))))\rangle\end{aligned} \tag{12}$$

$$\langle \mathsf{IF}_3(store, m, \mathsf{false}), \mathsf{PROCESS}(\mathsf{sndsplit}(m, \mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), store)), m)\rangle \tag{13}$$

To trace newly introduced redexes in an innermost reduction, we consider special sequences of dependency pairs, so-called *innermost chains*. A sequence of dependency pairs $\langle s_1, t_1\rangle\,\langle s_2, t_2\rangle \ldots$ is an innermost chain if there exists a substitution σ such that for all consecutive pairs $\langle s_j, t_j\rangle$ and $\langle s_{j+1}, t_{j+1}\rangle$ in the sequence we have $t_j\sigma \overset{\mathsf{i}}{\to}{}^*_{\mathcal{R}}\, s_{j+1}\sigma$. Here, "$\overset{\mathsf{i}}{\to}$" denotes innermost reductions (i.e., rewrite steps where only innermost redexes are contracted). In this way, the right-hand side of every dependency pair can be seen as the newly introduced redex that should be traced and the reductions $t_j\sigma \overset{\mathsf{i}}{\to}{}^*_{\mathcal{R}}\, s_{j+1}\sigma$ are necessary to normalize the arguments of the redex that is traced. Note that when regarding innermost reductions, arguments of a redex should be in normal form before the redex is contracted. Thus, we may restrict ourselves to substitutions σ where all $s_j\sigma$ are in normal form.

Definition 5 (Innermost $\mathcal{R}$-chains) *Let $\mathcal{R}$ be a TRS. A sequence of dependency pairs $\langle s_1, t_1\rangle\,\langle s_2, t_2\rangle \ldots$ is called an* innermost $\mathcal{R}$-chain *if there exists a substitution σ, such that all $s_j\sigma$ are in normal form and $t_j\sigma \overset{\mathsf{i}}{\to}{}^*_{\mathcal{R}}\, s_{j+1}\sigma$ holds for every two consecutive pairs $\langle s_j, t_j\rangle$ and $\langle s_{j+1}, t_{j+1}\rangle$ in the sequence.*

[2] Root symbols of left-hand sides are *defined* and all other functions are *constructors*.

We always assume that different (occurrences of) dependency pairs have disjoint variables and we always regard substitutions whose domains may be infinite. In [3] we showed that the absence of infinite innermost chains is a (sufficient and necessary) criterion for innermost termination.

Theorem 6 (Innermost Termination Criterion) *A TRS $\mathcal{R}$ is innermost terminating iff there exists no infinite innermost $\mathcal{R}$-chain.*

To improve this criterion we introduced the following graph which contains arcs between all those dependency pairs which may follow each other in innermost chains.

Definition 7 (Innermost Dependency Graph) *The* innermost dependency graph *of a TRS $\mathcal{R}$ is the directed graph whose nodes are the dependency pairs and there is an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ if $\langle s, t\rangle\ \langle v, w\rangle$ is an innermost $\mathcal{R}$-chain.*

In our example, (besides others) there are arcs from (9) to (10) and (12), from (10) to (11), from (12) to (13), and from both (11) and (13) to (9). The subgraph of the innermost dependency graph containing the nodes (9)–(13) is depicted in Fig. 2.

Since the innermost dependency graph is in general not computable, we use an estimation of this graph for automation purposes (cf. [3, 5, 8]). The estimation is such that all arcs in the original graph are also present in the estimated graph. Let CAP(t) result from t by replacing all subterms with defined root symbols by different fresh variables. The *estimated innermost dependency graph* is the directed graph whose nodes are the dependency pairs and there is an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ iff CAP(t) and v are unifiable by a mgu μ where $s\mu$ and $v\mu$ are normal forms. It is not difficult to see that whenever $\langle s, t\rangle\ \langle v, w\rangle$ is an innermost chain, then there is also an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ in the estimated innermost dependency graph. Thus, this estimated graph is indeed a supergraph of the (real) innermost dependency graph.

A non-empty set $\mathcal{P}$ of dependency pairs is called a *cycle* iff for all $\langle s, t\rangle$, $\langle v, w\rangle \in \mathcal{P}$, there is a path from $\langle s, t\rangle$ to $\langle v, w\rangle$ in the innermost dependency graph, which only traverses pairs from $\mathcal{P}$. Obviously, every cycle in this graph is also a cycle in the *estimated* innermost dependency graph.

In our example, the dependency pairs (9)–(13) form the cycles $\mathcal{P}_1 = \{(9), (10), (11)\}$, $\mathcal{P}_2 = \{(9), (12), (13)\}$, and $\mathcal{P}_3 = \{(9), (10), (11), (12),$

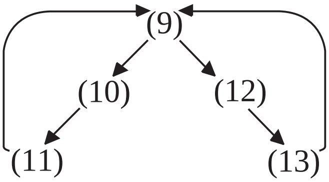

Fig. 2. Subgraph of the innermost dependency graph in our example

(13)}. However, (9)–(13) are not on a cycle with any *other* dependency pair (e.g., dependency pairs from the rules of the auxiliary library functions or the unspecified function f, since we assume that f does not call $\mathsf{process}$). This leads to the following refined criterion.

Theorem 8 (Modular Innermost Termination Criterion) *A finite TRS $\mathcal{R}$ is innermost terminating iff for each cycle $\mathcal{P}$ in the innermost dependency graph there exists no infinite innermost $\mathcal{R}$-chain of dependency pairs from $\mathcal{P}$.*

Note that for the soundness of this theorem one indeed has to regard *all* cycles, not just the minimal ones (i.e., not just those cycles which contain no other cycles as proper subsets). For example, the TRS with the rules $\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})$, $\mathsf{f}(\mathsf{1}) \to \mathsf{g}(\mathsf{0})$, and $\mathsf{g}(x) \to \mathsf{f}(x)$ has three dependency pairs

$$\langle \mathsf{F}(\mathsf{0}), \mathsf{G}(\mathsf{1})\rangle, \tag{14}$$

$$\langle \mathsf{F}(\mathsf{1}), \mathsf{G}(\mathsf{0})\rangle, \tag{15}$$

$$\langle \mathsf{G}(x), \mathsf{F}(x)\rangle \tag{16}$$

and three cycles $\mathcal{P}_1 = \{(14), (16)\}$, $\mathcal{P}_2 = \{(15), (16)\}$, and $\mathcal{P}_3 = \{(14), (15), (16)\}$. There is no infinite innermost chain from any of the minimal cycles $\mathcal{P}_1$ or $\mathcal{P}_2$. Nevertheless, the TRS is not innermost terminating, and indeed there is an infinite innermost chain from the non-minimal cycle $\mathcal{P}_3$.

In our definition, a cycle is a *set* of dependency pairs. Thus, a cycle never contains multiple occurrences of the same dependency pair and for a finite TRS there only exist finitely many cycles $\mathcal{P}$. The *automation* of the dependency pair technique is based on the generation of inequalities. For every cycle $\mathcal{P}$ (in the estimated graph) we search for a quasi-ordering $\geq_{\mathcal{P}}$ such that for any sequence of dependency pairs $\langle s_1, t_1\rangle\langle s_2, t_2\rangle\langle s_3, t_3\rangle \ldots$ from $\mathcal{P}$ and for any substitution σ with $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ (for all j) we have

$$s_1\sigma \geq_{\mathcal{P}} t_1\sigma \geq_{\mathcal{P}} s_2\sigma \geq_{\mathcal{P}} t_2\sigma \geq_{\mathcal{P}} s_3\sigma \geq_{\mathcal{P}} t_3\sigma \geq_{\mathcal{P}} \ldots$$

Moreover, for at least one $\langle s, t\rangle$ in $\mathcal{P}$ we demand the *strict* inequality $s\sigma >_{\mathcal{P}} t\sigma$. Here, $>_{\mathcal{P}}$ must be a well-founded ordering *compatible* with $\geq_{\mathcal{P}}$ (i.e., we have "$>_{\mathcal{P}} \circ \geq_{\mathcal{P}} \subseteq >_{\mathcal{P}}$ or $\geq_{\mathcal{P}} \circ >_{\mathcal{P}} \subseteq >_{\mathcal{P}}$"). Then there exists no innermost chain of dependency pairs from $\mathcal{P}$ which traverses all dependency pairs in $\mathcal{P}$ infinitely many times.

In the following we require that both $\geq_{\mathcal{P}}$ and $>_{\mathcal{P}}$ must be *closed under substitution*. Then $s_j \geq_{\mathcal{P}} t_j$ and $s_j >_{\mathcal{P}} t_j$ ensure $s_j\sigma \geq_{\mathcal{P}} t_j\sigma$ and $s_j\sigma >_{\mathcal{P}} t_j\sigma$, respectively, for all substitutions σ.

We also restrict ourselves to *weakly monotonic* quasi-orderings $\geq_{\mathcal{P}}$. (A quasi-ordering $\geq_{\mathcal{P}}$ is *weakly monotonic* if $s \geq_{\mathcal{P}} t$ implies $f(\ldots s \ldots) \geq_{\mathcal{P}} f(\ldots t \ldots)$.) Then to guarantee $t_j\sigma \geq_{\mathcal{P}} s_{j+1}\sigma$ whenever $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ holds, it is sufficient to demand $l \geq_{\mathcal{P}} r$ for all rules $l \to r$ of the TRS that may be used in this reduction. As we restrict ourselves to *normal* substitutions σ, not all rules are usable in a reduction of $t\sigma$. In general, if t contains a defined symbol f, then all f-rules are *usable* and moreover, all rules that are *usable* for right-hand sides of f-rules are also *usable* for t.

Definition 9 (Usable Rules) *Let $\mathcal{R}$ be a TRS. For any symbol f let $Rls_{\mathcal{R}}(f) = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) = f\}$. For any term we define the* usable rules:

- $\mathcal{U}_{\mathcal{R}}(x) = \emptyset$,
- $\mathcal{U}_{\mathcal{R}}(f(t_1, \ldots, t_n)) = Rls_{\mathcal{R}}(f) \;\cup\; \bigcup_{l \to r \in Rls_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}'}(r) \;\cup\; \bigcup_{j=1}^{n} \mathcal{U}_{\mathcal{R}'}(t_j)$,

where $\mathcal{R}' = \mathcal{R} \setminus Rls_{\mathcal{R}}(f)$. Moreover, for any set $\mathcal{P}$ of dependency pairs we define $\mathcal{U}_{\mathcal{R}}(\mathcal{P}) = \bigcup_{\langle s,t\rangle \in \mathcal{P}} \mathcal{U}_{\mathcal{R}}(t)$.

Note that this is indeed a recursive definition (since $\mathcal{R}$ is decreasing to $\mathcal{R}'$ in the second equation defining $\mathcal{U}_{\mathcal{R}}$).

Now we obtain the following theorem for automated[3] innermost termination proofs.

Theorem 10 (Innermost Termination Proofs) *A finite TRS is innermost terminating if for each cycle $\mathcal{P}$ there is a weakly monotonic quasi-ordering $\geq_{\mathcal{P}}$ and a well-founded ordering $>_{\mathcal{P}}$ compatible with $\geq_{\mathcal{P}}$, where both $\geq_{\mathcal{P}}$ and $>_{\mathcal{P}}$ are closed under substitution, such that*

- *$l \geq_{\mathcal{P}} r$ for all rules $l \to r \in \mathcal{U}_{\mathcal{R}}(\mathcal{P})$,*
- *$s \geq_{\mathcal{P}} t$ for all dependency pairs $\langle s, t\rangle$ from $\mathcal{P}$, and*
- *$s >_{\mathcal{P}} t$ for at least one dependency pair $\langle s, t\rangle$ from $\mathcal{P}$.*

We already demonstrated that for Thm. 8 (and hence, also for Thm. 10) considering just the minimal cycles would be unsound. In fact, for Thm. 10 it would also be unsound just to consider *maximal* cycles (i.e., those cycles which are not contained in any other cycle). The problem is that it is not sufficient if just one dependency pair of each maximal cycle is strictly decreasing. There must be a strictly decreasing dependency pair for every subcycle as well. As a counterexample regard the TRS $\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{s}(x))$, $\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(x)$. Its (only) maximal cycle is $\{\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{s}(x))\rangle, \langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(x)\rangle\}$. But the constraints $\mathsf{F}(\mathsf{s}(x)) \geq \mathsf{F}(\mathsf{s}(x))$ and $\mathsf{F}(\mathsf{s}(x)) > \mathsf{F}(x)$ for this cycle are easily fulfilled although this TRS is clearly not innermost terminating. Thus, it is crucial to consider *all* cycles $\mathcal{P}$ for Thm. 10.

In Sect. 2 we presented the rules for the auxiliary functions in our process example. Proving absence of infinite innermost chains for the cycles of their dependency pairs is very straightforward using Thm. 10. So all library functions of our TRS are innermost terminating. Moreover, as we assumed f to be a terminating function, its cycles do not lead to infinite innermost chains either.

[3] Additional refinements for the automation can be found in [3, 8].

Recall that (9)–(13) are not on cycles together with the remaining dependency pairs. Thus, what is left for verifying the desired property is proving absence of infinite innermost chains for the cycles $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$, where all rules of the whole TRS are possible candidates for being usable rules (also the rules for the unspecified function f).

Thm. 10 demands $s \geq_{\mathcal{P}} t$ resp. $s >_{\mathcal{P}} t$ for dependency pairs $\langle s, t\rangle$ on cycles. However for (9)–(13), these inequalities are not satisfied by any quasi-simplification ordering.[4] Thus, the automated proof fails here. Moreover, it is unclear which inequalities we have to add for the usable rules, since the rules for f are not given. Therefore, we have to extend the dependency pair technique.

5 Narrowing Dependency Pairs

To prove the absence of infinite innermost chains, for a dependency pair $\langle v, w\rangle$ it would be sufficient to demand $v\sigma \geq_{\mathcal{P}} w\sigma$ resp. $v\sigma >_{\mathcal{P}} w\sigma$ just for those instantiations σ where an instantiated right component $t\sigma$ of a previous dependency pair $\langle s, t\rangle$ reduces to $v\sigma$. For example, (11) only has to be regarded for instantiations σ where the instantiated right component $\mathsf{IF}_2(store, m, \mathsf{empty}(\mathsf{fstsplit}\,(m, store)))\sigma$ of (10) reduces to the instantiated left component $\mathsf{IF}_2(store, m, \mathsf{false})\sigma$ of (11). In fact, this can only happen if $store$ is not empty, i.e., if $store$ reduces to the form $\mathsf{cons}(h, t)$. However, this observation has not been used in the inequalities of Thm. 10 and hence, we could not find an ordering for them. Thus, the idea is to perform the computation of empty on the level of the dependency pair. For that purpose the well-known concept of *narrowing* is extended to pairs of terms.

Definition 11 *Let $\mathcal{R}$ be a TRS. If a term t $\mathcal{R}$-narrows to a term t' via the substitution μ, then the pair of terms $\langle s, t\rangle$ $\mathcal{R}$-narrows to the pair $\langle s\mu, t'\rangle$.*

In the following, we will usually speak of ‘narrowing’ instead of ‘$\mathcal{R}$-narrowing’ if the TRS $\mathcal{R}$ is clear from the context. For example, the narrowings of the dependency pair (10) are

$$\langle \mathsf{IF}_1(x, 0, \mathsf{true}), \mathsf{IF}_2(x, 0, \mathsf{empty}(\mathsf{nil}))\rangle \tag{10a}$$

$$\langle \mathsf{IF}_1(\mathsf{nil}, \mathsf{s}(n), \mathsf{true}), \mathsf{IF}_2(\mathsf{nil}, \mathsf{s}(n), \mathsf{empty}(\mathsf{nil}))\rangle \tag{10b}$$

$$\langle \mathsf{IF}_1(\mathsf{cons}(h, t), \mathsf{s}(n), \mathsf{true}), \mathsf{IF}_2(\mathsf{cons}(h, t), \mathsf{s}(n), \mathsf{empty}(\mathsf{cons}(h, \mathsf{fstsplit}(n, t))))\rangle. \tag{10c}$$

Thus, if a dependency pair $\langle s, t\rangle$ is followed by some dependency pairs $\langle v, w\rangle$ in an innermost chain and if t is not already unifiable with v (i.e., at least one rule is needed to reduce $t\sigma$ to $v\sigma$), then in order to ‘approximate’ the possible further $\mathcal{R}$-reductions of $t\sigma$ we may replace $\langle s, t\rangle$ by *all* its $\mathcal{R}$-narrowings.

[4] Essentially, the reason is that the left-hand side of dependency pair (9) is embedded in the right-hand sides of the pairs (11) and (13).

Hence, we can replace the dependency pair (10) by the new pairs (10a)–(10c), which already contain one 'hidden' step of the next $\mathcal{R}$-reduction.

This enables us to extract necessary information from the last arguments of if's, i.e., from the former conditions of the CTRS. Thus, the narrowing refinement is the main reason why the transformation of CTRSs into TRSs is useful when analyzing the termination behaviour with dependency pairs. The number of narrowings for a pair is finite (up to variable renaming) and it can easily be computed automatically.

Note however that narrowing may indeed only be applied for dependency pairs whose right-hand side does not unify with any left-hand side of a dependency pair (after variable renaming). As an example regard the following TRS.

$$\begin{aligned} \mathsf{g}(\mathsf{f}(\mathsf{a})) &\to \mathsf{h}(\mathsf{a}) \\ \mathsf{f}(\mathsf{b}) &\to \mathsf{c} \\ \mathsf{h}(x) &\to \mathsf{g}(\mathsf{f}(x)) \end{aligned}$$

This TRS is not innermost terminating as we have the infinite innermost reduction $\mathsf{g}(\mathsf{f}(\mathsf{a})) \overset{\mathsf{i}}{\to} \mathsf{h}(\mathsf{a}) \overset{\mathsf{i}}{\to} \mathsf{g}(\mathsf{f}(\mathsf{a})) \overset{\mathsf{i}}{\to} \ldots$ The only dependency pairs on a cycle are $\langle \mathsf{G}(\mathsf{f}(\mathsf{a})), \mathsf{H}(\mathsf{a})\rangle$ and $\langle \mathsf{H}(x), \mathsf{G}(\mathsf{f}(x))\rangle$. But if the latter dependency pair is narrowed to $\langle \mathsf{H}(\mathsf{b}), \mathsf{G}(\mathsf{c})\rangle$, then there is no cycle any more in the innermost dependency graph and hence, we would falsely conclude innermost termination. This example also demonstrates why this requirement is still necessary even if we would restrict ourselves to non-overlapping systems.

Before showing how narrowing helps in solving the inequalities of the process example, we first prove the soundness of our technique.

Theorem 12 (Narrowing Pairs) *Let $\mathcal{P}$ be a set of pairs of terms and let $\langle s, t\rangle \in \mathcal{P}$ such that $Var(t) \subseteq Var(s)$ and such that for all (renamings of) $\langle v, w\rangle \in \mathcal{P}$, the terms t and v are not unifiable. Let $\mathcal{P}'$ result from $\mathcal{P}$ by replacing $\langle s, t\rangle$ by all its narrowings. If there exists no infinite innermost chain of pairs from $\mathcal{P}'$, then there exists no infinite innermost chain of pairs from $\mathcal{P}$ either.*

Proof. Suppose there is an innermost $\mathcal{R}$-chain

$$\ldots\ \langle v_1, w_1\rangle\ \langle s, t\rangle\ \langle v_2, w_2\rangle\ \ldots$$

of pairs from $\mathcal{P}$. It suffices to prove that then there exists a narrowing $\langle s', t'\rangle$ of $\langle s, t\rangle$ such that $\ldots\ \langle v_1, w_1\rangle\ \langle s', t'\rangle\ \langle v_2, w_2\rangle\ \ldots$ is an innermost $\mathcal{R}$-chain as well. Here, $\langle s, t\rangle$ resp. $\langle s', t'\rangle$ may also be the first pair in the chain (i.e., $\langle v_1, w_1\rangle$ may be missing). If this has been proved, then all occurrences of $\langle s, t\rangle$ in an infinite innermost chain may be replaced by pairs from $\mathcal{P}'$.

For the above innermost chain, there must be a substitution σ such that all instantiated left-hand sides of the pairs are normal forms and every instantiated right-hand side reduces innermost to the instantiated left-hand side of the next pair in the innermost chain. Note that $t\sigma$ cannot be equal to $v_2\sigma$, as otherwise σ would be a unifier of t and v_2. Hence, we have $t\sigma \overset{\mathsf{i}}{\to}_{\mathcal{R}} q \overset{\mathsf{i}}{\to}^*_{\mathcal{R}} v_2\sigma$ for some term q.

The reduction $t\sigma \overset{\mathsf{i}}{\to}_{\mathcal{R}} q$ cannot take place 'in σ', because all variables of t are contained in s and hence, then $s\sigma$ would not be a normal form. Thus, t contains some subterm $f(\mathbf{u})$ such that a rule $l \to r$ has been applied to $f(\mathbf{u})\sigma$. In other words, l matches $f(\mathbf{u})\sigma$ (i.e. $l\rho = f(\mathbf{u})\sigma$). So the reduction has the following form:

$$t\sigma = t\sigma[f(\mathbf{u})\sigma]_\pi = t\sigma[l\rho]_\pi \overset{\mathsf{i}}{\to}_{\mathcal{R}} t\sigma[r\rho]_\pi = q.$$

As in the usual definition of narrowing, we assume that the variables of $l \to r$ have been renamed to fresh ones. Therefore we can extend σ to 'behave' like ρ on the variables of l and r (but it still remains the same on the variables of all pairs in the innermost chain). Now σ is a unifier of l and $f(\mathbf{u})$ and hence, there also exists a most general unifier μ. By the definition of most general unifiers, then there must be a substitution τ such that $\sigma = \mu\tau$.

Let t' be the term $t\mu[r\mu]_\pi$ and let s' be $s\mu$. Then $\langle s, t\rangle$ narrows to $\langle s', t'\rangle$. As we may assume s' and t' to be variable disjoint from all other pairs, we may extend σ to behave like τ on the variables of s' and t'. Then we have

$$w_1\sigma \overset{\mathsf{i}}{\to}^*_{\mathcal{R}} s\sigma = s\mu\tau = s'\tau = s'\sigma \text{ and}$$

$$t'\sigma = t'\tau = t\mu\tau[r\mu\tau]_\pi = t\sigma[r\sigma]_\pi = t\sigma[r\rho]_\pi = q \overset{\mathsf{i}}{\to}^*_{\mathcal{R}} v_2\sigma.$$

Hence, $\ldots \langle v_1, w_1\rangle\ \langle s', t'\rangle\ \langle v_2, w_2\rangle \ldots$ is also an innermost $\mathcal{R}$-chain. □

So we may always replace a dependency pair by all its narrowings. However, while this refinement is sound, in general it destroys the necessity of our innermost termination criterion in Thm. 8. For example, the TRS with the rules $\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{g}(\mathsf{h}(x)))$, $\mathsf{g}(\mathsf{h}(x)) \to \mathsf{g}(x)$, $\mathsf{g}(\mathsf{0}) \to \mathsf{s}(\mathsf{0})$, $\mathsf{h}(\mathsf{0}) \to \mathsf{1}$ is innermost terminating. But if the dependency pair $\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{g}(\mathsf{h}(x)))\rangle$ is replaced by its narrowings $\langle \mathsf{F}(\mathsf{s}(\mathsf{0})), \mathsf{F}(\mathsf{g}(\mathsf{1}))\rangle$ and $\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{g}(x))\rangle$, then $\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{g}(x))\rangle$ forms an infinite innermost chain (using the instantiation $\{x/\mathsf{0}\}$).

Nevertheless, in the application domain of process verification, we can restrict ourselves to TRSs with the unique normal form property.[5] In fact, the TRSs resulting from the translation of Erlang functions are always non-overlapping. As non-overlapping innermost terminating TRSs are confluent, they also satisfy the unique normal form property. Hence, the requirement of the unique normal form property in the following theorem could also be replaced by non-overlappingness.

[5] A TRS is said to have the unique normal form property iff for every term t, whenever $s_1 {}^*\!\!\leftarrow t \to^* s_2$ with s_1 and s_2 in normal form, then we have $s_1 = s_2$.

The theorem shows that for such TRSs, narrowing dependency pairs indeed is a completeness preserving technique. More precisely, whenever innermost termination can be proved with the pairs $\mathscr{P}$, then it can also be proved with the pairs $\mathscr{P}'$.

Theorem 13 (Narrowing Pairs Preserves Completeness) *Let $\mathscr{R}$ be an innermost terminating TRS with the unique normal form property and let $\mathscr{P}$, $\mathscr{P}'$ be as in Thm. 12. If there exists no infinite innermost $\mathscr{R}$-chain of pairs from $\mathscr{P}$, then there exists no infinite innermost $\mathscr{R}$-chain of pairs from $\mathscr{P}'$ either.*

Proof. We show that every innermost $\mathscr{R}$-chain $\ldots \langle v_1, w_1\rangle\ \langle s', t'\rangle\ \langle v_2, w_2\rangle \ldots$ from $\mathscr{P}'$ can be transformed into an innermost chain from $\mathscr{P}$ of same length. There must be a substitution σ such that for all pairs the instantiated left-hand side is a normal form and the instantiated right-hand side reduces to the instantiated left-hand side of the next pair in the innermost chain. So in particular we have

$$w_1\sigma \overset{\mathsf{i}}{\to}{}^{*}_{\mathscr{R}} s'\sigma \quad \text{and} \quad t'\sigma \overset{\mathsf{i}}{\to}{}^{*}_{\mathscr{R}} v_2\sigma.$$

We know that $\langle s, t\rangle$ narrows to $\langle s', t'\rangle$ via a substitution μ. As the variables in $\langle s, t\rangle$ are disjoint from all other variables, we may extend σ to ‘behave’ like $\mu\sigma$ on the variables of s and t. Then we have $s\sigma = s\mu\sigma = s'\sigma$ and hence, $w_1\sigma \overset{\mathsf{i}}{\to}{}^{*}_{\mathscr{R}} s\sigma$.

Moreover, by the definition of narrowing, $t\mu \to_{\mathscr{R}} t'$. This implies $t\mu\sigma \to_{\mathscr{R}} t'\sigma$ and as $t\sigma = t\mu\sigma$, we have $t\sigma \to_{\mathscr{R}} t'\sigma \overset{\mathsf{i}}{\to}{}^{*}_{\mathscr{R}} v_2\sigma$ where $v_2\sigma$ is a normal form. As $\mathscr{R}$ is innermost terminating and every term has a unique normal form, repeated application of *innermost* reduction steps to $t\sigma$ also yields the normal form $v_2\sigma$, i.e., $t\sigma \overset{\mathsf{i}}{\to}{}^{*}_{\mathscr{R}} v_2\sigma$. Thus, $\ldots\ \langle v_1, w_1\rangle\ \ \langle s, t\rangle\ \ \langle v_2, w_2\rangle\ \ldots$ is also an innermost $\mathscr{R}$-chain. □

Hence, *independent* of the technique used to check the absence of infinite innermost chains, for TRSs with the unique normal form property, narrowing dependency pairs preserves the success of the innermost termination proof. So we may narrow dependency pairs without the risk that the new pairs we obtain form an infinite innermost chain, whereas the original system is innermost terminating. Thus, in Thm. 6 and 8 when replacing the dependency pairs of $\mathscr{R}$ by their narrowings, one still obtains a sufficient and necessary criterion for innermost termination.

Moreover, narrowing can of course be repeated an *arbitrary* number of times. Thus, after replacing (10) by (10a)–(10c), we may subsequently replace (10a) and (10b) by their respective narrowings.

$$\langle \mathsf{IF}_1(x, \mathsf{0}, \mathsf{true}), \mathsf{IF}_2(x, \mathsf{0}, \mathsf{true})\rangle \tag{10aa}$$

$$\langle \mathsf{IF}_1(\mathsf{nil}, \mathsf{s}(n), \mathsf{true}), \mathsf{IF}_2(\mathsf{nil}, \mathsf{s}(n), \mathsf{true})\rangle \tag{10ba}$$

This excludes them from being on a cycle in the estimated innermost dependency graph. Thus, now instead of the dependency pairs (9)–(13) we consider (9), (10c), (11), (12), and (13). A further narrowing of (10c) is not necessary for our purposes (but according to Thm. 13 it would not harm either). The right component of the dependency pair (11) unifies with the left component of (9) and therefore, (11) must not be narrowed. Instead we narrow (9).

$$\langle \mathsf{PROCESS}(\mathsf{nil}, m), \mathsf{IF}_1(\mathsf{nil}, m, \mathsf{leq}(m, \mathsf{0}))\rangle \tag{9a}$$
$$\langle \mathsf{PROCESS}(\mathsf{cons}(h,t), m), \mathsf{IF}_1(\mathsf{cons}(h,t), m, \mathsf{leq}(m, \mathsf{s}(\mathsf{length}(t))))\rangle \tag{9b}$$
$$\langle \mathsf{PROCESS}(store, \mathsf{0}), \mathsf{IF}_1(store, \mathsf{0}, \mathsf{true})\rangle \tag{9c}$$

By narrowing (10) to (10c), we determined that we only have to regard instantiations where $store$ has the form $\mathsf{cons}(h,t)$ and m has the form $\mathsf{s}(n)$. Thus, (9a) and (9c) do not occur on a cycle and therefore, (9) can be replaced by (9b) only.

As (11)'s right component does not unify with left components any longer, we may now narrow (11) as well. By repeated narrowing steps and by dropping those pairs which do not occur on cycles, (11) can be replaced by

$$\langle \mathsf{IF}_2(\mathsf{cons}(h,t), \mathsf{s}(n), \mathsf{false}), \mathsf{PROCESS}(\mathsf{sndsplit}(n,t), \mathsf{s}(n))\rangle \tag{11aac}$$
$$\langle \mathsf{IF}_2(\mathsf{cons}(h,t), \mathsf{s}(n), \mathsf{false}), \mathsf{PROCESS}(\mathsf{app}(\mathsf{nil}, \mathsf{sndsplit}(n,t)), \mathsf{s}(n))\rangle \tag{11ad}$$
$$\begin{aligned}&\langle \mathsf{IF}_2(\mathsf{cons}(h,t), \mathsf{s}(n), \mathsf{false}),\\ &\qquad \mathsf{PROCESS}(\mathsf{app}(\mathsf{map\_f}(\mathsf{self}, \mathsf{nil}), \mathsf{sndsplit}(n,t)), \mathsf{s}(n))\rangle\end{aligned} \tag{11d}$$

Now for the cycle $\mathscr{P}_1$, it is (for example) sufficient to demand that (11aac), (11ad), and (11d) are strictly decreasing and that (9b), (10c), and all usable rules are weakly decreasing. Similar narrowings can also be applied for the pairs (12) and (13) which results in analogous inequalities for the cycles $\mathscr{P}_2$ and $\mathscr{P}_3$.

Most standard orderings amenable to automation are *strongly* monotonic path orderings (cf. e.g. [14, 31]), whereas here we only need *weak* monotonicity. Hence, before synthesizing a suitable ordering, some of the arguments of function symbols may be eliminated, cf. [8]. For example, in our inequalities one may eliminate the third argument of IF_2. Then every term $\mathsf{IF}_2(t_1, t_2, t_3)$ in the inequalities is replaced by $\mathsf{IF}'_2(t_1, t_2)$ (where IF'_2 is a new binary function symbol). By comparing the terms resulting from this replacement instead of the original terms, we can take advantage of the fact that IF_2 does not have to be strongly monotonic in its third argument. Similarly, in our example we will also eliminate the third arguments of IF_1 and IF_3 and the first argument of $\mathsf{sndsplit}$. Note that there are only finitely many (and only few) possibilities to eliminate arguments of function symbols. Therefore all these possibilities can be checked automatically. In this way, the recursive path ordering (rpo) [14] satisfies the inequalities for (11aac), (9b), (10c), for the dependency pairs resulting from (12) and (13), and for all (known) usable rules. However, the inequalities resulting from (11ad) and (11d)

$$\mathsf{IF}_2'(\mathsf{cons}(h,t),\mathsf{s}(n)) > \mathsf{PROCESS}(\mathsf{app}(\mathsf{nil},\mathsf{sndsplit}'(t)),\mathsf{s}(n))$$
$$\mathsf{IF}_2'(\mathsf{cons}(h,t),\mathsf{s}(n)) > \mathsf{PROCESS}(\mathsf{app}(\mathsf{map\_f}(\mathsf{self},\mathsf{nil}),\mathsf{sndsplit}'(t)),\mathsf{s}(n))$$

are not satisfied because of the app-terms on the right-hand sides (as the app-rules force app to be greater than cons in the precedence of the rpo). Moreover, the map_f-term in the inequalities requires us to consider the usable rules corresponding to the (unspecified) Erlang function `f` as well.

To get rid of these terms, one would like to perform narrowing on map_f and app. However, in general narrowing only *some* subterms of right components is unsound.[6] Instead, we always have to replace a pair by *all* its narrowings. But then narrowing (11ad) and (11d) provides no solution here, since narrowing the sndsplit-subterm results in pairs containing problematic app- and map_f-terms again. In the next section we describe a technique which solves the above problem.

6 Rewriting Dependency Pairs

While performing only some *narrowing* steps is unsound, for non-overlapping TRSs it is at least sound to perform only one of the possible *rewrite* steps. So if $t \to r$, then we may replace a dependency pair $\langle s, t\rangle$ by $\langle s, r\rangle$.

Note that this technique is only applicable to *dependency pairs*, but not to *rules* of the TRS. Indeed, by reducing the right-hand side of a rule, a non (innermost) terminating TRS can be transformed into a terminating one, even if the TRS is non-overlapping. As an example regard the TRS with the rules $\mathsf{0} \to \mathsf{f}(\mathsf{0}), \mathsf{f}(x) \to \mathsf{1}$ which is clearly not innermost terminating. However, if the right-hand side of the first rule is rewritten to $\mathsf{1}$, then the resulting TRS is terminating. The following theorem proves that our refinement of the dependency pair approach is sound.

Theorem 14 (Rewriting Pairs) *Let $\mathcal{R}$ be non-overlapping and let $\mathcal{P}$ be a set of pairs of terms. Let $\langle s, t\rangle \in \mathcal{P}$, let $t \to_{\mathcal{R}} r$ and let $\mathcal{P}'$ result from $\mathcal{P}$ by replacing $\langle s, t\rangle$ with $\langle s, r\rangle$. If there exists no infinite innermost chain of pairs from $\mathcal{P}'$, then there exists no infinite innermost chain from $\mathcal{P}$ either.*

Proof. By replacing all (renamed) occurrences of $\langle s, t\rangle$ with the corresponding renamed occurrences of $\langle s, r\rangle$, every innermost chain $\ldots \langle s, t\rangle\, \langle v, w\rangle \ldots$ from $\mathcal{P}$ can be translated into an innermost chain from $\mathcal{P}'$ of same length. The reason

[6] As an example regard the TRS $\mathsf{f}(\mathsf{0},\mathsf{1}) \to \mathsf{s}(\mathsf{1}), \mathsf{f}(x,\mathsf{0}) \to \mathsf{1}, \mathsf{a} \to \mathsf{0}$, and $\mathsf{g}(\mathsf{s}(y)) \to \mathsf{g}(\mathsf{f}(\mathsf{a},y))$. If we would replace the dependency pair $\langle \mathsf{G}(\mathsf{s}(y)), \mathsf{G}(\mathsf{f}(\mathsf{a},y))\rangle$ by only one of its narrowings, viz. $\langle \mathsf{G}(\mathsf{s}(\mathsf{0})), \mathsf{G}(\mathsf{1})\rangle$, then one could falsely prove innermost termination, although the term $\mathsf{g}(\mathsf{s}(\mathsf{1}))$ starts an infinite innermost reduction.

is that there must be a substitution σ with $t\sigma \overset{\mathsf{i}}{\to}{}^{*}_{\mathcal{R}} v\sigma$ where $v\sigma$ is a normal form. So $t\sigma$ is weakly innermost terminating[7] and as $\mathcal{R}$ is non-overlapping, by [22, Thm. 3.2.11 (1a) and (4a)] $t\sigma$ is confluent and terminating. With $t \to_{\mathcal{R}} r$, we obtain $t\sigma \to_{\mathcal{R}} r\sigma$. Hence, $r\sigma$ is terminating as well and thus, it also reduces innermost to some normal form q. Now confluence of $t\sigma$ implies $q = v\sigma$. Therefore, $\ldots \langle s, r\rangle\ \langle v, w\rangle \ldots$ is an innermost chain, too. □

The above theorem enables us to perform a rewrite step in the right-hand side of a dependency pair and to continue with this dependency pair instead of the original one. Note that a weakening of Thm. 14 by just demanding innermost confluence instead of non-overlappingness of $\mathcal{R}$ is not possible; not even if we only allow *innermost* reductions in the right-hand side of a dependency pair. As a counterexample consider $\mathsf{h}(\mathsf{f}(x)) \to \mathsf{h}(\mathsf{g}(\mathsf{s}(x)))$, $\mathsf{h}(\mathsf{g}(\mathsf{a})) \to \mathsf{h}(\mathsf{f}(\mathsf{a}))$, $\mathsf{g}(\mathsf{s}(x)) \to \mathsf{b}$, $\mathsf{s}(\mathsf{a}) \to \mathsf{a}$. This TRS is innermost confluent, but not innermost terminating (since $\mathsf{h}(\mathsf{f}(\mathsf{a}))$ starts a cycling reduction). Thus, the set $\mathcal{P}$ of all dependency pairs forms an infinite innermost chain. But if we perform an innermost rewrite step on the dependency pair $\langle \mathsf{H}(\mathsf{f}(x)), \mathsf{H}(\mathsf{g}(\mathsf{s}(x)))\rangle$, then it is replaced by $\langle \mathsf{H}(\mathsf{f}(x)), \mathsf{H}(\mathsf{b})\rangle$. Now the resulting set of pairs has no infinite innermost chains any more, and thus, we could falsely conclude innermost termination.

However, the demand that the TRS should be non-overlapping may be weakened by demanding that it is *innermost normal form preserving*, i.e., for any term t, whenever $s \overset{\mathsf{i}}{\leftarrow}{}^{*} t \to r$ holds for a normal form s, then $r \overset{\mathsf{i}}{\to}{}^{*} s$. Non-overlapping TRSs are innermost normal form preserving, but not vice versa (consider $\mathsf{a} \to \mathsf{a}$, $\mathsf{a} \to \mathsf{b}$). In practice, however, the above version of Thm. 14 is most important, since it is usually much easier to show that a TRS is non-overlapping than that it is innermost normal form preserving.

The converse of Thm. 14 holds as well if $\mathcal{P}$ is obtained from the dependency pairs by repeated narrowing and rewriting steps. So similar to *narrowing*, *rewriting* dependency pairs also preserves the necessity of our criterion.

Theorem 15 (Rewriting Pairs Preserves Completeness) *Let $\mathcal{R}$ be an innermost terminating TRS with the unique normal form property and let $\mathcal{P}$, $\mathcal{P}'$ be as in Thm. 14. If there exists no infinite innermost $\mathcal{R}$-chain of pairs from $\mathcal{P}$, then there exists no infinite innermost $\mathcal{R}$-chain of pairs from $\mathcal{P}'$ either.*

Proof. In an innermost chain $\ldots \langle s, r\rangle\ \langle v, w\rangle \ldots$ from $\mathcal{P}'$, replacing all (renamed) occurrences of $\langle s, r\rangle$ by corresponding renamings of $\langle s, t\rangle$ yields an innermost chain from $\mathcal{P}$ of same length. The reason is that there must be a σ with $r\sigma \overset{\mathsf{i}}{\to}{}^{*}_{\mathcal{R}} v\sigma$. As $\mathcal{R}$ is innermost terminating, there must be a normal form q

[7] We call a *term* t (innermost) terminating if all (innermost) reductions starting in t are finite. Analogously, t is weakly (innermost) terminating if there exists a finite (innermost) reduction starting in t.

which is reachable from $t\sigma$ by innermost reduction steps, i.e., $t\sigma \overset{\mathsf{i}}{\to}^*_{\mathcal{R}} q$. Thus, $t\sigma \to_{\mathcal{R}} r\sigma \overset{\mathsf{i}}{\to}^*_{\mathcal{R}} v\sigma$ implies $q = v\sigma$ by the unique normal form property of $\mathcal{R}$, and hence, $t\sigma \overset{\mathsf{i}}{\to}^*_{\mathcal{R}} v\sigma$. □

In our example we may now eliminate app and map_f by rewriting the pairs (11ad) and (11d). Even better, before narrowing, we could first rewrite (11), (12), and (13). Moreover, we could simplify (10c) by rewriting it as well. Thus, the resulting pairs on the cycles we are interested in are:

$$\langle \mathsf{PROCESS}(\mathsf{cons}(h,t),m), \mathsf{IF}_1(\mathsf{cons}(h,t),m,\mathsf{leq}(m,\mathsf{s}(\mathsf{length}(t))))\rangle \tag{9b}$$
$$\langle \mathsf{IF}_1(\mathsf{cons}(h,t),\mathsf{s}(n),\mathsf{true}), \mathsf{IF}_2(\mathsf{cons}(h,t),\mathsf{s}(n),\mathsf{false})\rangle \tag{10c$'$}$$
$$\langle \mathsf{IF}_2(store,m,\mathsf{false}), \mathsf{PROCESS}(\mathsf{sndsplit}(m,store),m)\rangle \tag{11$'$}$$
$$\langle \mathsf{IF}_1(store,m,\mathsf{false}), \mathsf{IF}_3(store,m,\mathsf{empty}(\mathsf{fstsplit}(m,store)))\rangle \tag{12$'$}$$
$$\langle \mathsf{IF}_3(store,m,\mathsf{false}), \mathsf{PROCESS}(\mathsf{sndsplit}(m,store),m)\rangle \tag{13$'$}$$

Analogous to Sect. 5, now we narrow (11′), (12′), (13′), perform a rewrite step for one of (12′)'s narrowings, and delete those resulting pairs which are not on any cycle. In this way, (11′), (12′), (13′) are replaced by

$$\langle \mathsf{IF}_2(\mathsf{cons}(h,t),\mathsf{s}(n),\mathsf{false}), \mathsf{PROCESS}(\mathsf{sndsplit}(n,t),\mathsf{s}(n))\rangle \tag{11$''$}$$
$$\langle \mathsf{IF}_1(\mathsf{cons}(h,t),\mathsf{s}(n),\mathsf{false}), \mathsf{IF}_3(\mathsf{cons}(h,t),\mathsf{s}(n),\mathsf{false})\rangle \tag{12$''$}$$
$$\langle \mathsf{IF}_3(\mathsf{cons}(h,t),\mathsf{s}(n),\mathsf{false}), \mathsf{PROCESS}(\mathsf{sndsplit}(n,t),\mathsf{s}(n))\rangle \tag{13$''$}$$

By eliminating the first argument of sndsplit and the third arguments of IF_1, IF_2, and IF_3 (cf. Sect. 5), we obtain the following inequalities. Note that according to Thm. 10, these inequalities prove the absence of infinite innermost chains for all three cycles built from (9b), (10c′), and (11″)–(13″), since for each of these cycles (at least) one of its dependency pairs is strictly decreasing.

$$\begin{aligned}
\mathsf{PROCESS}(\mathsf{cons}(h,t),m) &\geq \mathsf{IF}'_1(\mathsf{cons}(h,t),m)\\
\mathsf{IF}'_1(\mathsf{cons}(h,t),\mathsf{s}(n)) &\geq \mathsf{IF}'_2(\mathsf{cons}(h,t),\mathsf{s}(n))\\
\mathsf{IF}'_1(\mathsf{cons}(h,t),\mathsf{s}(n)) &\geq \mathsf{IF}'_3(\mathsf{cons}(h,t),\mathsf{s}(n))\\
\mathsf{IF}'_2(\mathsf{cons}(h,t),\mathsf{s}(n)) &> \mathsf{PROCESS}(\mathsf{sndsplit}'(t),\mathsf{s}(n))\\
\mathsf{IF}'_3(\mathsf{cons}(h,t),\mathsf{s}(n)) &> \mathsf{PROCESS}(\mathsf{sndsplit}'(t),\mathsf{s}(n))\\
\mathsf{sndsplit}'(x) &\geq x\\
\mathsf{sndsplit}'(\mathsf{nil}) &\geq \mathsf{nil}\\
\mathsf{sndsplit}'(\mathsf{cons}(h,t)) &\geq \mathsf{sndsplit}'(t)\\
l &\geq r \quad \text{for all rules } l \to r \text{ with } root(l) \in \{\mathsf{leq}, \mathsf{length}\}
\end{aligned}$$

Now these inequalities are satisfied by the rpo. The sndsplit′-, leq-, and length-inequalities are the only ones which correspond to the usable rules, since the rules for map_f and f are no longer usable. Hence, the TRS of Sect. 3 is innermost terminating. In this way, left-right decreasingness of the CTRS from Sect. 2 could be proved automatically. Therefore, the desired property holds for the original Erlang process.

7 Verifying Networks of Processes

In many applications, one is not only interested in verifying certain properties of a single process in a network, but instead one wants to verify a property of the whole network of processes. If these processes work asynchronously, then the exact order of the messages passed through the network is often indeterministic. Modelling this kind of behaviour usually results in TRSs which are *overlapping* (and in fact, not confluent).

In this section we extend the well-known result that innermost termination of non-overlapping TRSs implies their termination to the class of overlapping TRSs which result from describing process networks in our framework. Then we show that our techniques of narrowing and rewriting dependency pairs can also be applied to overlapping TRSs. Moreover, we introduce a third technique to modify dependency pairs, viz. *instantiating* dependency pairs, which is particularly useful when dealing with non-confluent TRSs. With these extensions, we show how an important property for a network of Erlang processes could be successfully verified.

In this verification problem, we have a ring of three asynchronous processes (similar to the process described in Sect. 2). The aim is to prove that if the first process disregards its input (i.e., it performs as if it repeatedly gets the empty list as input), then eventually, the third process will also send the empty list. Of course, if one can prove this for a ring of three processes, then a similar proof for any other number of processes works analogously.

To model this situation, we use a CTRS similar to the one of Sect. 2. However, as we have to regard all three processes simultaneously, we need a new defined symbol ring to describe the current state of the whole network. The term

$$\mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m)$$

describes a situation where the stores of the processes 1, 2, and 3 have the values st_1, st_2, and st_3, respectively. The variable in_2 is a list of lists containing all messages which have been sent from Process 1 to Process 2, but which have not yet been received by Process 2. Similarly, in_3 is the list of those messages sent from Process 2 to Process 3, which have not yet been received by Process 3. The messages sent from Process 3 to Process 1 are ignored, because in our verification problem we assume that Process 1 receives no new input any more. Again, m is the (maximum) length of messages allowed.

In order to prove the desired conjecture, we force the reduction to terminate as soon as all processes in the ring can only send the empty message. In addition to the auxiliary functions of Sect. 2 we now also need the functions head and tail which are defined by the following rules.

$$\mathsf{head}(\mathsf{cons}(h, t)) \to h \qquad \mathsf{tail}(\mathsf{cons}(h, t)) \to t$$

The CTRS to describe the behaviour of the three processes in the ring is the following one.

$$\begin{array}{l}
\mathsf{empty}(\mathsf{fstsplit}(m, st_1)) \to^* \mathsf{false} \quad | \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m) \to \\
\quad \mathsf{ring}(\mathsf{sndsplit}(m, st_1), \mathsf{cons}(\mathsf{fstsplit}(m, st_1), in_2), st_2, in_3, st_3, m)
\end{array} \tag{17}$$

$$\begin{array}{l}
\mathsf{leq}(m, \mathsf{length}(st_2)) \to^* \mathsf{true}, \\
\mathsf{empty}(\mathsf{fstsplit}(m, st_2)) \to^* \mathsf{false} \quad | \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m) \to \\
\quad \mathsf{ring}(st_1, in_2, \mathsf{sndsplit}(m, st_2), \mathsf{cons}(\mathsf{fstsplit}(m, st_2), in_3), st_3, m)
\end{array} \tag{18}$$

$$\begin{array}{l}
\mathsf{leq}(m, \mathsf{length}(st_2)) \to^* \mathsf{false}, \\
\mathsf{empty}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(2, \mathsf{head}(in_2)), st_2))) \to^* \mathsf{false} \quad | \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m) \to \\
\quad \mathsf{ring}(st_1, \mathsf{tail}(in_2), \mathsf{sndsplit}(m, \mathsf{app}(\mathsf{map\_f}(2, \mathsf{head}(in_2)), st_2)), \\
\qquad\qquad \mathsf{cons}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(2, \mathsf{head}(in_2)), st_2)), in_3), st_3, m)
\end{array} \tag{19}$$

$$\begin{array}{l}
\mathsf{empty}(\mathsf{map\_f}(2, \mathsf{head}(in_2))) \to^* \mathsf{true} \quad | \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m) \to \mathsf{ring}(st_1, \mathsf{tail}(in_2), st_2, in_3, st_3, m)
\end{array} \tag{20}$$

$$\begin{array}{l}
\mathsf{leq}(m, \mathsf{length}(st_3)) \to^* \mathsf{true}, \\
\mathsf{empty}(\mathsf{fstsplit}(m, st_3)) \to^* \mathsf{false} \quad | \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m) \to \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, \mathsf{sndsplit}(m, st_3), m)
\end{array} \tag{21}$$

$$\begin{array}{l}
\mathsf{leq}(m, \mathsf{length}(st_3)) \to^* \mathsf{false}, \\
\mathsf{empty}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(3, \mathsf{head}(in_3)), st_3))) \to^* \mathsf{false} \quad | \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m) \to \\
\quad \mathsf{ring}(st_1, in_2, st_2, \mathsf{tail}(in_3), \mathsf{sndsplit}(m, \mathsf{app}(\mathsf{map\_f}(3, \mathsf{head}(in_3)), st_3)), m)
\end{array} \tag{22}$$

$$\begin{array}{l}
\mathsf{empty}(\mathsf{map\_f}(3, \mathsf{head}(in_3))) \to^* \mathsf{true} \quad | \\
\quad \mathsf{ring}(st_1, in_2, st_2, in_3, st_3, m) \to \mathsf{ring}(st_1, in_2, st_2, \mathsf{tail}(in_3), st_3, m)
\end{array} \tag{23}$$

Rule (17) describes how Process 1 sends a message consisting of the first m items in its store st_1. To that end, $\mathsf{fstsplit}(m, st_1)$ is added to those other items in_2 which were already sent as an input to Process 2, but which have not yet been received by this next process. These first m items are taken out of the store st_1, i.e., its new value is $\mathsf{sndsplit}(m, st_1)$.

The rules (18) and (19) describe the case where Process 2 sends a message. If its store already contains at least m items, then Rule (18) applies and the first m items $\mathsf{fstsplit}(m, st_2)$ are directly sent to Process 3, after which these items are removed from its store. Otherwise, if st_2 contains less than m items, then Rule (19) is used to receive one of the incoming messages from in_2, i.e., in_2 is replaced by $\mathsf{tail}(in_2)$. For these received items $\mathsf{head}(in_2)$, the process computes new items $\mathsf{map\_f}(2, \mathsf{head}(in_2))$ and appends these newly computed items to its store. Afterwards it sends the first m items of the new extended store to Process 3.

Finally, Rule (20) deletes those messages from in_2 that Process 2 would not generate any new items from (i.e., where $\mathsf{map\_f}(2, \mathsf{head}(in_2))$ is empty). This rule is required in order to allow Process 2 to continue receiving messages from $\mathsf{tail}(in_2)$, even if $\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(2, \mathsf{head}(in_2)), st_2))$ is empty.

Similarly, Rules (21) and (22) describe the sending of messages by Process 3. The only difference is that messages sent by Process 3 are not delivered to Process 1 again, but they are ignored. Analogous to Rule (20), Rule (23) is used to remove those messages from in_3 for which Process 3 does not compute new items. The ring-term will be irreducible as soon as none of the processes can send a non-empty message any longer.

To prove the desired conjecture, we have to show that this CTRS is left-right decreasing. Note that this CTRS indeed models an *asynchronous* behaviour of the processes. The reason is that we do not determine in which order the processes send messages to the next process in the ring. Consequently, the translation of this CTRS yields a non-confluent unconditional TRS. In the following TRS, "…" abbreviates the arguments "$st_1, in_2, st_2, in_3, st_3, m$".

$$\mathsf{ring}(\ldots) \to \mathsf{if}_1(\ldots, \mathsf{empty}(\mathsf{fstsplit}(m, st_1))) \tag{24}$$

$$\mathsf{if}_1(\ldots,\mathsf{false}) \to \mathsf{ring}(\mathsf{sndsplit}(m, st_1), \mathsf{cons}(\mathsf{fstsplit}(m, st_1), in_2), st_2, in_3, st_3, m) \tag{25}$$

$$\mathsf{ring}(\ldots) \to \mathsf{if}_2(\ldots, \mathsf{leq}(m, \mathsf{length}(st_2))) \tag{26}$$

$$\mathsf{if}_2(\ldots, \mathsf{true}) \to \mathsf{if}_3(\ldots, \mathsf{empty}(\mathsf{fstsplit}(m, st_2))) \tag{27}$$

$$\mathsf{if}_3(\ldots, \mathsf{false}) \to \mathsf{ring}(st_1, in_2, \mathsf{sndsplit}(m, st_2), \mathsf{cons}(\mathsf{fstsplit}(m, st_2), in_3), st_3, m) \tag{28}$$

$$\mathsf{if}_2(\ldots, \mathsf{false}) \to \mathsf{if}_4(\ldots, \mathsf{empty}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(2, \mathsf{head}(in_2)), st_2)))) \tag{29}$$

$$\begin{aligned}\mathsf{if}_4(\ldots, \mathsf{false}) \to\ & \mathsf{ring}(st_1, \mathsf{tail}(in_2), \mathsf{sndsplit}(m, \mathsf{app}(\mathsf{map\_f}(2, \mathsf{head}(in_2)), st_2)),\\ & \quad \mathsf{cons}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(2, \mathsf{head}(in_2)), st_2)), in_3), st_3, m)\end{aligned} \tag{30}$$

$$\mathsf{ring}(\ldots) \to \mathsf{if}_5(\ldots, \mathsf{empty}(\mathsf{map\_f}(2, \mathsf{head}(in_2)))) \tag{31}$$

$$\mathsf{if}_5(\ldots, \mathsf{true}) \to \mathsf{ring}(st_1, \mathsf{tail}(in_2), st_2, in_3, st_3, m) \tag{32}$$

$$\mathsf{ring}(\ldots) \to \mathsf{if}_6(\ldots, \mathsf{leq}(m, \mathsf{length}(st_3))) \tag{33}$$

$$\mathsf{if}_6(\ldots, \mathsf{true}) \to \mathsf{if}_7(\ldots, \mathsf{empty}(\mathsf{fstsplit}(m, st_3))) \tag{34}$$

$$\mathsf{if}_7(\ldots, \mathsf{false}) \to \mathsf{ring}(st_1, in_2, st_2, in_3, \mathsf{sndsplit}(m, st_3), m) \tag{35}$$

$$\mathsf{if}_6(\ldots, \mathsf{false}) \to \mathsf{if}_8(\ldots, \mathsf{empty}(\mathsf{fstsplit}(m, \mathsf{app}(\mathsf{map\_f}(3, \mathsf{head}(in_3)), st_3)))) \tag{36}$$

$$\begin{aligned}\mathsf{if}_8(\ldots, \mathsf{false}) \to\ & \mathsf{ring}(st_1, in_2, st_2, \mathsf{tail}(in_3),\\ & \quad \mathsf{sndsplit}(m, \mathsf{app}(\mathsf{map\_f}(3, \mathsf{head}(in_3)), st_3)), m)\end{aligned} \tag{37}$$

$$\mathsf{ring}(\ldots) \to \mathsf{if}_9(\ldots, \mathsf{empty}(\mathsf{map\_f}(3, \mathsf{head}(in_3)))) \tag{38}$$

$$\mathsf{if}_9(\ldots, \mathsf{true}) \to \mathsf{ring}(st_1, in_2, st_2, \mathsf{tail}(in_3), st_3, m) \tag{39}$$

According to Corollary 3 now it suffices to show that this TRS is terminating. Note that this TRS is obviously not simply terminating. For example, by adding the embedding rules $\mathsf{fstsplit}(m, st_1) \to st_1$, $\mathsf{sndsplit}(m, st_1) \to st_1$, $\mathsf{empty}(l) \to l$, and $\mathsf{cons}(h, t) \to t$ to the first two rules (24) and (25), one can obtain a cycling reduction of $\mathsf{ring}(\mathsf{false}, in_2, st_2, in_3, st_3, m)$ to itself.

In fact, to prove termination of this TRS using the dependency pair approach in combination with simplification orderings, we again need our refinements of narrowing and rewriting dependency pairs. However, recall that the refinements of the theorems 12–15 were restricted to *innermost* termination proofs. In the example of Sect. 3, the resulting TRS was non-overlapping and thus, innermost termination was enough to conclude its termination. However, now we have a TRS which is not confluent and hence, none of the existing results for proving termination by innermost termination is applicable.

Nevertheless, the following theorem shows that for TRSs like the one in our example, innermost termination still implies termination. Note that our TRS is a hierarchical combination of a non-overlapping TRS $\mathcal{R}_1$ (which defines the auxiliary functions) and an overlapping TRS $\mathcal{R}_2$ with the ring- and if-rules to describe the network verification problem. In fact, TRSs of this form occur frequently in the process verification domain, since the auxiliary Erlang functions always result in non-overlapping rules, whereas the description of an asynchronous process network often requires overlapping rules. The following theorem gives a syntactical characterization of these TRSs, and it shows that for such systems, innermost termination already implies termination. Hence, this theorem is an important result in order to facilitate their termination proofs.

Theorem 16 (Sufficiency of Innermost Termination) *Let $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, where $\mathcal{R}_1$ is non-overlapping, $\mathcal{R}_2$ is non-collapsing, and $\mathcal{R}_2$-rules do not form critical pairs with $\mathcal{R}_1$-rules. Let Σ contain all root symbols of left- and right-hand sides of $\mathcal{R}_2$-rules, i.e., $\Sigma = \{\mathrm{root}(l) | \ l \to r \in \mathcal{R}_2\} \cup \{\mathrm{root}(r) | \ l \to r \in \mathcal{R}_2\}$. If no $\mathcal{R}_1$-rule contains symbols from Σ and if no $\mathcal{R}_2$-rule contains symbols from Σ below the root level, then innermost termination of $\mathcal{R}$ implies termination of $\mathcal{R}$.*

Proof. For any ground term t, we write $t = C[\![t_1, \ldots, t_n]\!]$ provided that C is a non-empty context (i.e., $C \neq \Box$) which does not contain symbols from Σ below the root level and provided that $\mathrm{root}(t_i) \in \Sigma$ for all $1 \leq i \leq n$. Now it is easy to see that if $t = C[\![t_1, \ldots, t_n]\!]$ and $t \to_{\mathcal{R}} s$, then we have one of the following three possibilities:

(i) $s = C[\![t_1, \ldots, t_{i-1}, s_i, t_{i+1}, \ldots, t_n]\!]$ and $t_i \to_{\mathcal{R}} s_i$ for some $1 \leq i \leq n$
(in this case, we speak of a *bottom* rewrite step)

(ii) $s = C'[\![s_1, \ldots, s_m]\!]$, $C \to_{\mathcal{R}} C'$, and $\{s_1, \ldots, s_m\} \subseteq \{t_1, \ldots, t_n\}$
(in this case, we speak of a *top* rewrite step)

(iii) $s = t_i$ for some $1 \leq i \leq n$
(in this case, we have a *top collapsing* rewrite step).

The reason is that reducing a term t with $\mathrm{root}(t) \in \Sigma$ again yields a term whose root is from Σ and that symbols of Σ do not occur below the root level in any rule of $\mathcal{R}$. Thus, if the root of the redex is in C, then we really must have a step of the form (ii) or (iii).

Now assume that $\mathcal{R}$ is innermost terminating, but not terminating. Let t be a minimal ground term (w.r.t. the subterm relation) such that t starts an infinite $\mathcal{R}$-reduction. Again, we must have $t = C[\![t_1, \ldots, t_n]\!]$ for some context C. Due to the minimality of t, its subterms $t_1, \ldots, t_n$ are terminating. Thus, in the infinite reduction of t, there cannot be any top collapsing rewrite step and there can only be finitely many bottom rewrite steps. Hence, C starts an infinite $\mathcal{R}$-reduction as well.

In other words, if $\mathcal{R}$ is not terminating, then there exists a non-terminating context C which does not contain any Σ-symbol below the root level. To use standard notation, we will now denote this context C by q, since a context is just a term possibly containing '□' symbols.

First suppose that q does not contain any Σ-symbol at all. Then the only rules applicable in any reduction of q are from $\mathcal{R}_1$. However, $\mathcal{R}$'s innermost termination implies that all innermost reductions starting from q are finite. Thus, q is innermost terminating w.r.t. $\mathcal{R}_1$ and since $\mathcal{R}_1$ is non-overlapping, by [22, Thm. 3.2.11 (1a)] we know that q is also terminating, which yields a contradiction.

Thus, innermost termination of $\mathcal{R}$ in fact implies termination of $\mathcal{R}_1$ for all terms without symbols from Σ. Now suppose that the root of q is from Σ, i.e., q has the form $f_0(“\mathbf{s_0}”)$ with $f_0 \in \Sigma$ and “$\mathbf{s_0}$” are terms without symbols from Σ. Thus, the infinite $\mathcal{R}$-reduction of $f_0(“\mathbf{s_0}”)$ must have the following form.

$$f_0(“\mathbf{s_0}”) \to^*_{\mathcal{R}_1} f_0(“\mathbf{t_0}”) \to_{\mathcal{R}_2} f_1(“\mathbf{s_1}”) \to^*_{\mathcal{R}_1} f_1(“\mathbf{t_1}”) \to_{\mathcal{R}_2} f_2(“\mathbf{s_2}”) \to^*_{\mathcal{R}_1} \ldots$$

Here, we have $f_i \in \Sigma$ for all i, the terms “$\mathbf{s_i}$” and “$\mathbf{t_i}$” do not contain any symbols from Σ, and we have “$\mathbf{s_i}$” $\to^*_{\mathcal{R}_1}$ “$\mathbf{t_i}$”.

Hence, there must be substitutions σ_i and rules $f_i(“\mathbf{l_i}”) \to f_{i+1}(“\mathbf{r_i}”)$ in $\mathcal{R}_2$ such that $“\mathbf{l_i}”\sigma_i = “\mathbf{t_i}”$ and $“\mathbf{r_i}”\sigma_i = “\mathbf{s_{i+1}}”$. Let σ'_i be the substitution with $\sigma'_i(x) = (\sigma_i(x)) \downarrow_{\mathcal{R}_1}$. (For terms without symbols from Σ, the normal form w.r.t. $\mathcal{R}_1$ is well defined, since these terms are terminating and $\mathcal{R}_1$ is non-overlapping.) Since $\mathcal{R}_2$ does not form critical pairs with $\mathcal{R}_1$-rules, we have $“\mathbf{l_i}”\sigma'_i = (“\mathbf{l_i}”\sigma_i) \downarrow_{\mathcal{R}_1} = “\mathbf{t_i}” \downarrow_{\mathcal{R}_1} = “\mathbf{s_i}” \downarrow_{\mathcal{R}_1}$. Moreover, we have $(“\mathbf{r_i}”\sigma'_i) \downarrow_{\mathcal{R}_1} = “\mathbf{s_{i+1}}” \downarrow_{\mathcal{R}_1}$ by the convergence of $\mathcal{R}_1$ for terms without symbols from Σ. This implies

$$f_0(“\mathbf{s_0}”\!\downarrow_{\mathcal{R}_1}) \to_{\mathcal{R}_2} f_1(“\mathbf{r_0}”\sigma'_0) \to^*_{\mathcal{R}_1} f_1(“\mathbf{s_1}”\!\downarrow_{\mathcal{R}_1}) \to_{\mathcal{R}_2} f_2(“\mathbf{r_1}”\sigma'_1) \to^*_{\mathcal{R}_1} f_2(“\mathbf{s_2}”\!\downarrow_{\mathcal{R}_1}) \to_{\mathcal{R}_2} \ldots$$

Since $\mathcal{R}_1$ is terminating, we can use innermost steps to reduce each $“\mathbf{r_i}”\sigma'_i$ to its normal form $“\mathbf{s_{i+1}}” \downarrow_{\mathcal{R}_1}$. Moreover, all the $\mathcal{R}_2$-steps in the above reduction are innermost steps as well, since the arguments $“\mathbf{s_i}” \downarrow_{\mathcal{R}_1}$ are in normal form. Thus, the above reduction is an infinite *innermost* reduction, which yields a contradiction to the innermost termination of $\mathcal{R}$. □

Thus in our example, innermost termination of the transformed TRS indeed implies termination of the TRS and thus, it implies left-right decreasingness of the original CTRS. Hence, in this way the property of the process network can be proved.

As indicated, to perform this innermost termination proof, we again need our refinements of narrowing and rewriting dependency pairs. However, as this TRS is not confluent, for this purpose these techniques now have to be extended to overlapping TRSs.

It turns out that such an extension is indeed possible, because for the theorems 13–15 it is in fact sufficient to demand non-overlappingness (resp. the unique normal form property) just for the usable rules $\mathscr{U}(\mathscr{P})$ instead of the whole TRS $\mathscr{R}$. In our example, the usable rules of the RING-cycles only consist of the rules for the auxiliary functions, i.e., the rules (24)–(39) are not usable. As demonstrated in Sect. 2, these auxiliary rules are non-overlapping. Thus, the following extensions of the theorems 13–15 allow us to apply our new techniques for TRSs like the one above, too. In this way, conjectures about asynchronous networks of processes can now be verified by dependency pairs as well.

Theorem 17 (Completeness of Narrowing for Non-Confluent Systems) *Let $\mathscr{R}$ be an innermost terminating TRS, let $\mathscr{P}$, $\mathscr{P}'$ be as in Thm. 12 and let $\mathscr{U}(\mathscr{P})$ have the unique normal form property. If there exists no infinite innermost $\mathscr{R}$-chain of pairs from $\mathscr{P}$, then there exists no infinite innermost $\mathscr{R}$-chain of pairs from $\mathscr{P}'$ either.*

Proof. The proof is similar to the one of Thm. 13. The only difference is the proof that $t\sigma \to^*_{\mathscr{R}} v_2\sigma$ implies $t\sigma \stackrel{\mathsf{i}}{\to}{}^*_{\mathscr{R}} v_2\sigma$ for the normal form $v_2\sigma$. The reason is that innermost termination of $\mathscr{R}$ implies that there must exist some normal form q such that $t\sigma \stackrel{\mathsf{i}}{\to}{}^*_{\mathscr{R}} q$. Note that all rules used in any reduction of $t\sigma$ are contained in $\mathscr{U}(\mathscr{P})$. Thus, the unique normal form property of $\mathscr{U}(\mathscr{P})$ is enough to conclude $q = v_2\sigma$. □

Theorem 18 (Rewriting Pairs for Non-Confluent TRSs) *Let $\mathscr{R}$ be a TRS and let $\mathscr{P}$ be a set of pairs of terms such that $\mathscr{U}(\mathscr{P})$ is non-overlapping. Let $\langle s, t\rangle \in \mathscr{P}$, let $t \to_{\mathscr{R}} r$ and let $\mathscr{P}'$ result from $\mathscr{P}$ by replacing $\langle s, t\rangle$ with $\langle s, r\rangle$. If there exists no infinite innermost chain of pairs from $\mathscr{P}'$, then there exists no infinite innermost chain from $\mathscr{P}$ either.*

Proof. Again, the proof is similar to the proof of Thm. 14. The only extra observation needed is that $t\sigma \stackrel{\mathsf{i}}{\to}{}^*_{\mathscr{R}} v\sigma$ implies $t\sigma \stackrel{\mathsf{i}}{\to}{}^*_{\mathscr{U}(\mathscr{P})} v\sigma$, since all rules applicable in a reduction of $t\sigma$ are contained in $\mathscr{U}(\mathscr{P})$. Hence, by non-overlappingness of $\mathscr{U}(\mathscr{P})$ we can apply [22, Thm. 3.2.11 (1a) and (4a)] to conclude termination and confluence of $t\sigma$ w.r.t. $\mathcal{U}(\mathcal{P})$. But as all rules applicable in reductions of $t\sigma$ are already contained in $\mathcal{U}(\mathcal{P})$, this means that $t\sigma$ is terminating and confluent w.r.t. $\mathcal{R}$ as well. Thus, now the rest of the proof is identical to the one of Thm. 14. □

Theorem 19 (Completeness of Rewriting for Non-Confluent TRS) *Let $\mathcal{R}$ be an innermost terminating TRS, let $\mathcal{P}$, $\mathcal{P}'$ be as in Thm. 18, and let $\mathcal{U}(\mathcal{P})$ have the unique normal form property. If there exists no infinite innermost $\mathcal{R}$-chain of pairs from $\mathcal{P}$, then there exists no infinite innermost $\mathcal{R}$-chain of pairs from $\mathcal{P}'$ either.*

Proof. The changes to the proof of Thm. 15 are similar as in the proof of Thm. 17. We have $t\sigma \to_{\mathcal{R}}^{*} v\sigma$ for some normal form $v\sigma$ and innermost termination of $\mathcal{R}$ implies $t\sigma \overset{\mathsf{i}}{\to}_{\mathcal{R}}^{*} q$ for some normal form q. Again, all these reduction steps only use rules from $\mathcal{U}(\mathcal{P})$. Thus, $\mathcal{U}(\mathcal{P})$'s unique normal form property implies $v\sigma = q$. □

Note that with these refined theorems we can also handle TRSs where different, but equivalent if-symbols are not identified (cf. Sect. 3). However in practice, such an identification is still useful, since it simplifies the TRSs considerably.

In particular, due to the above extended theorems, now we may apply narrowing and rewriting to the dependency pairs resulting from the rules (24)–(39). The only dependency pair resulting from Rule (24) which is on a cycle is $\langle \mathsf{RING}(\ldots), \mathsf{IF}_1(\ldots)\rangle$. Narrowing and rewriting this dependency pair (and deleting those resulting pairs which are not on cycles) yields

$$\langle \mathsf{RING}(\mathsf{cons}(h,t),\ldots,\mathsf{s}(n)),\ \mathsf{IF}_1(\mathsf{cons}(h,t),\ldots,\mathsf{s}(n),\mathsf{false})\rangle. \tag{40}$$

Next we regard the dependency pair $\langle \mathsf{IF}_1(\ldots),\ \mathsf{RING}(\ldots)\rangle$ resulting from Rule (25). One would like to perform narrowing on this dependency pair. However, this is not possible since its right-hand side unifies with the left-hand sides of the dependency pairs resulting from the rules (26), (31), (33), and (38). In fact, this problem is typical when regarding overlapping TRSs.

Nevertheless, the only pair which may occur before $\langle \mathsf{IF}_1(\ldots),\ \mathsf{RING}(\ldots)\rangle$ in an innermost chain is (40). When regarding (40), one immediately sees that therefore one only has to regard instantiations of $\langle \mathsf{IF}_1(\ldots),\ \mathsf{RING}(\ldots)\rangle$ where st_1 is replaced by $\mathsf{cons}(h,t)$ and m is replaced by $\mathsf{s}(n)$.

Recall that when estimating the innermost dependency graph, for every dependency pair $\langle s,t\rangle$ we check for which (renamings of) dependency pairs $\langle v,w\rangle$, $\textsc{cap}(w)$ unifies with s (where their mgu must satisfy some additional normality condition). Here, $\textsc{cap}(w)$ results from replacing all subterms of w with defined root symbols by different fresh variables. Let $\mu_1,\ldots,\mu_k$ be all mgu's of s and terms of the form $\textsc{cap}(w)$. Then one may replace the dependency pair $\langle s, t\rangle$ by its instantiations $\langle s\mu_1, t\mu_1\rangle, \ldots, \langle s\mu_k, t\mu_k\rangle$, since (specializations of) these instantiations are the only ones that are needed in infinite innermost chains. This leads to the technique of *instantiating dependency pairs*.

Theorem 20 (Instantiating Pairs) *Let $\mathcal{P}$ be a set of pairs of terms with $\langle s, t\rangle \in \mathcal{P}$ and let $Var(w) \subseteq Var(v)$ for all $\langle v, w\rangle \in \mathcal{P}$. Let*

$$\mathcal{P}' = \mathcal{P} \setminus \{\langle s, t\rangle\} \cup \{\langle s\mu, t\mu\rangle \mid \mu = \mathrm{mgu}(\textsc{cap}(w), s),\ \langle v, w\rangle \in \mathcal{P}\},$$

where we again assume that different occurrences of pairs from $\mathcal{P}$ are variable disjoint. Then there exists no infinite innermost chain of pairs from $\mathcal{P}'$ iff there exists no infinite innermost chain of pairs from $\mathcal{P}$.

Proof. If $\ldots \langle v_1, w_1\rangle\ \langle s, t\rangle\ \langle v_2, w_2\rangle \ldots$ is an innermost chain, then there exists a substitution σ such that $w_1\sigma \overset{\mathrm{i}}{\to}{}^*_{\mathcal{R}} s\sigma$. Let w_1 have the form $C[p_1, \ldots, p_n]$, where the context C contains no defined symbols and all p_i have a defined root symbol. As reductions cannot take place in σ (since otherwise, $v_1\sigma$ would not be a normal form), we know that $s\sigma = C\sigma[q_1, \ldots, q_n]$ where $p_i\sigma \overset{\mathrm{i}}{\to}{}^*_{\mathcal{R}} q_i$.

We have $\textsc{cap}(w_1) = C[y_1, \ldots, y_n]$, where the y_i are fresh variables. Let σ' be the modification of σ such that $\sigma'(y_i) = q_i$. Then we obtain $\textsc{cap}(w_1)\sigma' = s\sigma = s\sigma'$, i.e., $\textsc{cap}(w_1)$ and s are unifiable. Let μ be the mgu of $\textsc{cap}(w_1)$ and s. Thus, there exists a substitution τ such that $\sigma' = \mu\tau$. As the variables of all (occurrences of all) pairs may be assumed disjoint, we may modify σ to behave like τ on the variables of $\langle s\mu, t\mu\rangle$. Then we have $w_1\sigma \overset{\mathrm{i}}{\to}{}^*_{\mathcal{R}} s\sigma = s\sigma' = s\mu\tau = (s\mu)\sigma$ and we also have $(t\mu)\sigma = t\mu\tau = t\sigma \overset{\mathrm{i}}{\to}{}^*_{\mathcal{R}} v_2\sigma$. Thus, $\ldots \langle v_1, w_1\rangle\ \langle s\mu, t\mu\rangle\ \langle v_2, w_2\rangle \ldots$ is an innermost chain, too.

In this way, one can replace all occurrences of $\langle s, t\rangle$ in innermost chains by pairs of $\mathcal{P}'$, except for the very first pair in the chain. However, if $\langle s, t\rangle\ \langle v_1, w_1\rangle\ \langle v_2, w_2\rangle \ldots$ is an infinite innermost chain, then $\langle v_1, w_1\rangle\ \langle v_2, w_2\rangle \ldots$ is an infinite innermost chain as well. Thus, by deleting the possibly remaining first occurrence of $\langle s, t\rangle$ in the end, every infinite innermost chain of $\mathcal{P}$ can indeed be transformed into an infinite innermost chain of $\mathcal{P}'$.

For the other direction, let $\ldots\ \langle s\mu, t\mu\rangle\ \ldots$ be an innermost chain. As different occurrences of dependency pairs may be assumed variable disjoint, we can extend every substitution σ to behave like $\mu\sigma$ on the variables of s. Hence, this direction of the theorem is immediately proved. □

It should be remarked that the technique of instantiating dependency pairs can also be used for termination instead of innermost termination proofs. When using dependency pairs for arbitrary termination proofs, one has to prove absence of infinite chains (instead of *innermost* chains), where $\langle s_1, t_1\rangle\ \langle s_2, t_2\rangle \ldots$ is an $\mathcal{R}$-chain if there exists a substitution σ such that $t_j\sigma \to^*_{\mathcal{R}} s_{j+1}\sigma$ for all consecutive pairs $\langle s_j, t_j \rangle$ and $\langle s_{j+1}, t_{j+1} \rangle$, cf. [2, 8]. Let $\textsc{ren}(t)$ result from renaming all occurrences of variables to fresh variables (in particular, different occurrences of the same variable are also renamed to different new variables). If $\mathscr{P}' = \mathscr{P} \setminus \{\langle s, t \rangle\} \cup \{\langle s\mu, t\mu \rangle \mid \mu = \text{mgu}(\textsc{ren}(\textsc{cap}(w)), s),\ \langle v, w \rangle \in \mathscr{P}\}$, then there exists no infinite chain of pairs from $\mathscr{P}'$ iff there exists no infinite chain of pairs from $\mathscr{P}$. The proof is very similar to the proof of Thm. 20. The only difference is that now we write w_1 as $C[p_1, \dots p_n]$ where C contains no defined symbols *or variables* and all p_i either have a defined root symbol *or they are variables*. Then we know that $s\sigma = C[q_1, \dots, q_n]$ with $p_i\sigma \to_{\mathscr{R}}^* q_i$ and $\textsc{ren}(\textsc{cap}(w_1)) = C[y_1, \dots, y_n]$ where the y_i are fresh variables. The rest of the proof is completely analogous.

In our example, the only right-hand side of a pair whose $\textsc{cap}$ unifies with the left-hand side $\mathsf{IF}_1(st_1, in_2, st_2, in_3, st_3, m, \mathsf{false})$ of the dependency pair from Rule (25) is $\mathsf{IF}_1(\mathsf{cons}(h, t), in_2, st_2, in_3, st_3, \mathsf{s}(n), \mathsf{false})$ from Pair (40). Thus, we can instantiate st_1 by $\mathsf{cons}(h, t)$ and m by $\mathsf{s}(n)$ in the dependency pair $\langle \mathsf{IF}_1(\dots),\ \mathsf{RING}(\dots) \rangle$ from Rule (25). Subsequent rewriting yields

$$\langle \mathsf{IF}_1(\mathsf{cons}(h, t), \dots, \mathsf{s}(n), \mathsf{false}),\ \mathsf{RING}(\mathsf{sndsplit}(n, t), \dots, \mathsf{s}(n)) \rangle. \tag{41}$$

The only dependency pair resulting from Rule (26) which is on a cycle is

$$\langle \mathsf{RING}(\dots),\ \mathsf{IF}_2(\dots, \mathsf{leq}(m, \mathsf{length}(st_2))) \rangle. \tag{42}$$

For the dependency pair $\langle \mathsf{IF}_2(\dots), \mathsf{IF}_3(\dots) \rangle$ from Rule (27) we proceed in a similar way as for the one from Rule (24) which yields

$$\langle \mathsf{IF}_2(\dots, \mathsf{cons}(h, t), \dots, \mathsf{s}(n), \mathsf{true}),\ \mathsf{IF}_3(\dots, \mathsf{cons}(h, t), \dots, \mathsf{s}(n), \mathsf{false}) \rangle. \tag{43}$$

Rule (28) gives rise to a dependency pair $\langle \mathsf{IF}_3(\dots), \mathsf{RING}(\dots) \rangle$. The only dependency pair which may precede this one in innermost chains is (43). Thus, by the instantiation technique, st_2 can be replaced by $\mathsf{cons}(h, t)$ and m can be replaced by $\mathsf{s}(n)$. Subsequent rewriting yields

$$\langle \mathsf{IF}_3(st_1, in_2, \mathsf{cons}(h, t), \dots),\ \mathsf{RING}(st_1, in_2, \mathsf{sndsplit}(n, t), \dots) \rangle. \tag{44}$$

The dependency pair $\langle \mathsf{IF}_2(\dots),\ \mathsf{IF}_4(\dots) \rangle$ from Rule (29) yields the following narrowing.

$$\langle \mathsf{IF}_2(st_1, \mathsf{cons}(h, t), \dots),\ \mathsf{IF}_4(st_1, \mathsf{cons}(h, t), \dots) \rangle \tag{45}$$

For the dependency pair resulting from Rule (30) we only have to regard the instantiation where in_2 is replaced by $\mathsf{cons}(h, t)$. Rewriting this pair yields

$$\langle \mathsf{IF}_4(st_1, \mathsf{cons}(h, t), \dots),\ \mathsf{RING}(st_1, t, \dots) \rangle. \tag{46}$$

Similarly, narrowing the dependency pair $\langle \mathsf{RING}(\ldots),\ \mathsf{IF}_5(\ldots)\rangle$ from Rule (31) yields

$$\langle \mathsf{RING}(st_1, \mathsf{cons}(h, t), \ldots),\ \mathsf{IF}_5(st_1, \mathsf{cons}(h, t), \ldots)\rangle. \qquad (47)$$

So the dependency pair $\langle \mathsf{IF}_5(\ldots),\ \mathsf{RING}(\ldots)\rangle$ from Rule (32) only has to be regarded for the instantiation of in_2 by $\mathsf{cons}(h, t)$ and thus, rewriting it results in

$$\langle \mathsf{IF}_5(st_1, \mathsf{cons}(h, t), \ldots),\ \mathsf{RING}(st_1, t, \ldots)\rangle. \qquad (48)$$

Finally, for the dependency pairs resulting from the rules (33)–(39) we proceed in an analogous way and we obtain seven pairs similar to (42)–(48). Now the resulting constraints from the dependency pair approach are satisfied by the lexicographic path ordering (lpo) [25] if one eliminates the last arguments of all IF-symbols and the first argument of sndsplit before (to benefit from the fact that these symbols do not have to be strongly monotonic in these arguments). In this way, all of the above dependency pairs are weakly decreasing and the ones with a RING-term as their right component are strictly decreasing. The precedence used for this lpo should make RING and the IF-symbols equally great, whereas the tuple symbols should be greater than all lower case symbols. Of course, here we assume that the rules for the function f are also weakly decreasing w.r.t. the lpo. The reason is that now we consider a problem where non-empty lists must be processed and thus, the f-rules are usable as well. Hence, as soon as the actual rules for the function f are determined, their weak decreasingness has to be checked.

Thus, in this section we have demonstrated that although asynchronous networks are described by non-confluent (C)TRSs, proving *innermost* termination is still sufficient for their termination proof. Subsequently, we have shown that our techniques of rewriting and narrowing dependency pairs can be extended to TRSs where just the usable rules (i.e., the rules for the auxiliary functions) satisfy non-overlappingness requirements. Finally, we have introduced a third technique for manipulating dependency pairs, viz. instantiation. In this way, now dependency pairs can also be used to prove statements about asynchronous networks of processes.

8 Conclusion

We have shown that the dependency pair approach can be successfully applied for process verification tasks in industry. While our work was motivated by specific process verification problems, in this paper we developed several techniques which are of general use in term rewriting.

First of all, we showed how dependency pairs can be utilized to prove that *conditional* term rewriting systems are decreasing and terminating. Moreover, we presented three refinements which considerably increase the class of systems where dependency pairs are successful. The first refinement of *narrowing* dependency pairs for innermost termination was already introduced in [8]. However, [8] did not contain an explicit proof of its soundness, and completeness of the technique for TRSs with unique normal forms is a new result. It ensures that application of the narrowing technique preserves the success of such an innermost termination proof. In fact, our narrowing refinement is the main reason why the approach of handling CTRSs by transforming them into TRSs is successful in combination with the dependency pair approach (whereas this transformation is usually not of much use for the standard termination proving techniques). To strengthen the power of dependency pairs we also introduced the novel technique of *rewriting* dependency pairs and proved its soundness and completeness for innermost termination of non-overlapping TRSs. Finally, the refinement of *instantiating* dependency pairs was presented and we showed how to lift the non-overlappingness restrictions for narrowing and rewriting dependency pairs in order to apply these techniques to non-confluent TRSs. We also developed a new syntactical characterization for a class of (possibly) non-confluent TRSs where innermost termination implies termination, which captures those rewrite systems describing asynchronous process networks. This paper is a substantially revised and extended version of [6] and [7].

Note that we have used the modularity results for the dependency pair technique [5] for both a split and conquer approach and for dealing with the incompleteness of our specification. For many reasons, in practice it is more rule than exception that a specification lacks some information, like the definition of the function f in our example. Usually, at a certain level of abstraction one stops specifying and, hence, for many built-in functions the specification is preferably hidden (e.g., one could add a date as a time stamp to every message where in many cases the computation of this date is not relevant). Thus, assuming some properties of the missing part of the specification and proving them for that part when it becomes available makes sense. In that context the modularity of the dependency pair technique is of great help.

Our techniques have shown to be successfully applicable in small, but real examples, where eventuality properties had to be proved. These experiences demonstrate that our approach is particularly useful for verifying properties of processes where a lot of data manipulation is involved and where communication plays a minor role. Typically, these are the properties that are hard to handle by model-checking. The examples in this paper represent such situations where model-checking cannot be used because of the arbitrary lengths of the stores. These problems have also been tackled by a specialized proof checker for Erlang [1]. Compared to dependency pairs, the proof checker approach is more generally applicable. But since in that approach the proofs had, up to a great extend, to be provided by hand, the dependency pair approach has the important advantage that it is much better suitable for automation.

References

1. Arts, T., Dam, M.: Verifying a distributed database lookup manager written in Erlang. In: Proc. FM ’99, Toulouse, France, LNCS, Vol. 1708, pp. 682–700. Berlin Heidelberg New York: Springer 1999
2. Arts, T., Giesl, J.: Automatically proving termination where simplification orderings fail. In: Proc. TAPSOFT ’97, Lille, France. LNCS, Vol. 1214, pp. 261–273. Berlin Heidelberg New York: Springer 1997
3. Arts, T., Giesl, J.: Proving innermost normalisation automatically. In: Proc. RTA-97, Sitges, Spain. LNCS, Vol. 1232, pp. 157–172. Berlin Heidelberg New York: Springer 1997
4. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Technical Report IBN 97/46, Darmstadt University of Technology, Germany. `http://www-i2.informatik.rwth-aachen.de/giesl/papers/ibn-97-46.ps`
5. Arts, T., Giesl, J.: Modularity of termination using dependency pairs. In: Proc. RTA-98, Tsukuba, Japan. LNCS, Vol. 1379, pp. 226–240. Berlin Heidelberg New York: Springer 1998
6. Arts, T., Giesl, J.: Verification of Erlang Processes. In: Proc. 4th International Workshop on Termination, Dagstuhl, Germany, 1999
7. Arts, T., Giesl, J.: Applying rewriting techniques to the verification of Erlang processes. In: Proc. CSL ’99, Madrid, Spain. LNCS, Vol. 1683, pp. 96–110. Berlin Heidelberg New York: Springer 1999
8. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. TCS **236**, 133–178 (2000). Preliminary extended version appeared in [4]
9. Arts, T.: System description: The dependency pair method. In: Proc. RTA-00, Norwich, UK. LNCS, Vol. 1833, pp. 261–264. Berlin Heidelberg New York: Springer 2000
10. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press 1998
11. Bergstra, J. A., Klop, J. W.: Conditional rewrite rules: confluence and termination. JCSS **32**, 323–362 (1986)
12. Bertling, H., Ganzinger, H.: Completion-time optimization of rewrite-time goal solving. In: Proc. RTA-89, Chapel Hill, USA. LNCS, Vol. 355, pp. 45–58. Berlin Heidelberg New York: Springer 1989
13. Dershowitz, N., Plaisted, D. A.: Equational programming. Machine Intelligence **11**, 21–56 (1987)
14. Dershowitz, N.: Termination of rewriting. JSC **3**, 69–116 (1987)
15. Dershowitz, N., Okada, M., Sivakumar, G.: Canonical conditional rewrite systems. In: Proc. CADE-9, Argonne, USA. LNCS, Vol. 310, pp. 538–549. Berlin Heidelberg New York: Springer 1988
16. Dershowitz, N., Okada, M.: A rationale for conditional equational programming. TCS **75**, 111–138 (1990)
17. Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In: Handbook of Theoretical Computer Science, Vol. B, pp. 243–320. Amsterdam: Elsevier 1990
18. Dershowitz, N., Hoot, C.: Natural termination. TCS **142**, 179–207 (1995)
19. Giovanetti, E., Moiso, C.: Notes on the eliminations of conditions. In: Proc. CTRS ’87, Orsay, France. LNCS, Vol. 308, pp. 91–97. Berlin Heidelberg New York: Springer 1987
20. Gramlich, B.: On termination and confluence of conditional rewrite systems. In: Proc. CTRS ’94, Jerusalem, Israel. LNCS, Vol. 968, pp. 166–185. Berlin Heidelberg New York: Springer 1994
21. Gramlich, B.: Abstract relations between restricted termination and confluence properties of rewrite systems. Fundamenta Informaticae **24**, 3–23 (1995)
22. Gramlich, B.: Termination and confluence properties of structured rewrite systems. PhD Thesis. Universität Kaiserslautern, Germany (1996)

23. Gramlich, B.: On termination and confluence properties of disjoint and constructor-sharing conditional rewrite systems. TCS **165**, 97–131 (1996)
24. Patent pending, Ericsson Telecom AB 1999
25. Kamin, S., Levy, J.-J.: Two generalizations of the recursive path ordering. Department of Computer Science, University of Illinois, IL (1980)
26. Jouannaud, J.-P., Waldmann, B.: Reductive conditional term rewrite systems. In: 3rd IFIP Working Conference on Formal Description of Programming Concepts, Ebberup, Denmark. pp. 223–244. 1986
27. Kaplan, S.: Conditional rewrite rules. TCS **33**, 175–193 (1984)
28. Marchiori, M.: Unravelings and ultra-properties. In: Proc. ALP '96, Aachen, Germany. LNCS, Vol. 1139, pp. 107–121. Berlin Heidelberg New York: Springer 1996
29. Middeldorp, A.: Modular properties of conditional term rewriting systems. Inf. Comput. **104**, 110–158 (1993)
30. Ohlebusch, E.: Transforming conditional rewrite systems with extra variables into unconditional systems. In: Proc. LPAR '99, Tblisi, Georgia. LNAI, Vol. 1705, pp. 111–130. Berlin Heidelberg New York: Springer 1999
31. Steinbach, J.: Simplification orderings: history of results. Fundamenta Informaticae **24**, 47–87 (1995)
32. Suzuki, T., Middeldorp, A., Ida, T.: Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In: Proc. RTA-95, Kaiserslautern, Germany. LNCS, Vol. 914, pp. 179–193. Berlin Heidelberg New York: Springer 1995
33. Wirth, C.-P., Gramlich, B.: A constructor-based approach for positive/negative conditional equational specifications. JSC **17**, 51–90 (1994)