RWTH Aachen

Department of Computer Science
Technical Report

Proving and Disproving Termination of Higher-Order Functions

Jürgen Giesl, René Thiemann, Peter Schneider-Kamp

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2005-3

RWTH Aachen · Department of Computer Science · May 2005 (revised version)

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

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

Proving and Disproving Termination of Higher-Order Functions

Jürgen Giesl, René Thiemann, Peter Schneider-Kamp

LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany
{giesl|thiemann|psk}@informatik.rwth-aachen.de

Abstract. The dependency pair technique is a powerful modular method for automated termination proofs of term rewrite systems (TRSs). We present two important extensions of this technique: First, we show how to prove termination of *higher-order* functions using dependency pairs. To this end, the dependency pair technique is extended to handle (untyped) applicative TRSs. Second, we introduce a method to prove *non-termination* with dependency pairs, while up to now dependency pairs were only used to verify termination. Our results lead to a framework for combining termination and non-termination techniques for first- and higher-order functions in a very flexible way. We implemented and evaluated our results in the automated termination prover AProVE.

1 Introduction

One of the most powerful techniques to prove termination or innermost termination of TRSs automatically is the *dependency pair approach* [4, 12, 13]. In [16], we recently showed that dependency pairs can be used as a general framework to combine arbitrary techniques for termination analysis in a modular way. The general idea of this framework is to solve termination problems by repeatedly decomposing them into sub-problems. We call this new concept the "dependency pair *framework*" ("DP framework") to distinguish it from the old "dependency pair *approach*". In particular, this framework also facilitates the development of new methods for termination analysis. After recapitulating the basics of the DP framework in Sect. 2, we present two new significant improvements: in Sect. 3 we extend the framework in order to handle *higher-order* functions and in Sect. 4 we show how to use the DP framework to prove *non-termination.* Sect. 5 summarizes our results and describes their empirical evaluation with the system AProVE.

2 The Dependency Pair Framework

We refer to [6] for the basics of rewriting and to [4, 13, 16] for motivations and details on dependency pairs. We only regard finite signatures and TRSs and $\mathcal{T}(\mathcal{F}, \mathcal{V})$ denotes the set of terms over the signature $\mathcal{F}$ and the infinite set of variables $\mathcal{V} = \{x, y, z, \ldots, \alpha, \beta, \ldots\}$. $\mathcal{R}$ is a *TRS over* $\mathcal{F}$ if $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for all rules $l \to r \in \mathcal{R}$.

Our approach is restricted to untyped higher-order functions which do not use λ-abstraction. To represent higher-order functions, we use the well-known approach to encode them in curried form as *applicative* first-order TRSs (cf. e.g., [21]). A signature $\mathcal{F}$ is called *applicative* if it only contains nullary function symbols and a binary symbol $'$ for function application. Moreover, any TRS $\mathcal{R}$ over $\mathcal{F}$ is called *applicative.* So for example, instead of a term $\mathsf{map}(\alpha, x)$ we write $'('(\mathsf{map},\alpha),x)$. To ease readability, we use $'$ as an infix-symbol and to avoid unnecessary parentheses, we let $'$ associate to the left. Then this term can be written as $\mathsf{map}\,'\,\alpha\,'\,x$. In this way, one can easily encode typical higher-order functional programs as first-order TRSs, provided that these programs do not use λ-abstractions.

Example 1 *The function* map *is used to apply a function to all elements in a list. Instead of the higher-order rules* $\mathsf{map}(\alpha,\mathsf{nil}) \to \mathsf{nil}$ *and* $\mathsf{map}(\alpha,\mathsf{cons}(x,xs)) \to \mathsf{cons}(\alpha(x),\mathsf{map}(\alpha,xs))$*, we encode it by the following first-order TRS.*

$$\mathsf{map}\,'\,\alpha\,'\,\mathsf{nil} \to \mathsf{nil} \tag{1}$$

$$\mathsf{map}\,'\,\alpha\,'\,(\mathsf{cons}\,'\,x\,'\,xs) \to \mathsf{cons}\,'\,(\alpha\,'\,x)\,'\,(\mathsf{map}\,'\,\alpha\,'\,xs) \tag{2}$$

A TRS is terminating if all reductions are finite, i.e., if all functions encoded in the TRS terminate. So intuitively, the TRS $\{(1),(2)\}$ is terminating iff the function map terminates whenever its arguments are terminating terms.

For a TRS $\mathcal{R}$ over $\mathcal{F}$, the *defined symbols* are $\mathcal{D} = \{\mathrm{root}(l) \mid l \to r \in \mathcal{R}\}$ and the *constructors* are $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. For every $\mathsf{f} \in \mathcal{F}$ let $\mathsf{f}^\sharp$ be a fresh *tuple symbol* with the same arity as f, where we often write F for $\mathsf{f}^\sharp$. The set of tuple symbols is denoted by $\mathcal{F}^\sharp$. If $t = \mathsf{g}(t_1,\ldots,t_m)$ with $\mathsf{g} \in \mathcal{D}$, we let $t^\sharp$ denote $\mathsf{g}^\sharp(t_1,\ldots,t_m)$.

Definition 2 (Dependency Pair) *The set of* dependency pairs *for a TRS* $\mathcal{R}$ *is* $DP(\mathcal{R}) = \{l^\sharp \to t^\sharp \mid l \to r \in \mathcal{R},\ t \text{ is a subterm of } r,\ \mathrm{root}(t) \in \mathcal{D}\}$.

Example 3 *In the TRS of Ex. 1, the only defined symbol is* $'$ *and* map, cons, *and* nil *are constructors. Let* AP *denote the tuple symbol for* $'$. *Then we have the following dependency pairs where* s *is the term* $\mathsf{AP}(\mathsf{map}\,'\,\alpha,\mathsf{cons}\,'\,x\,'\,xs)$.

$$s \to \mathsf{AP}(\mathsf{cons}\,'\,(\alpha\,'\,x),\mathsf{map}\,'\,\alpha\,'\,xs) \tag{3}$$

$$s \to \mathsf{AP}(\mathsf{cons},\alpha\,'\,x) \tag{4}$$

$$s \to \mathsf{AP}(\alpha,x) \tag{5}$$

$$s \to \mathsf{AP}(\mathsf{map}\,'\,\alpha,xs) \tag{6}$$

$$s \to \mathsf{AP}(\mathsf{map},\alpha) \tag{7}$$

For termination, we try to prove that there are no infinite *chains* of dependency pairs. Intuitively, a dependency pair corresponds to a function call and a chain represents a possible sequence of calls that can occur during a reduction. We always assume that different occurrences of dependency pairs are variable disjoint and consider substitutions whose domains may be infinite. In the following definition, $\mathcal{P}$ is usually a set of dependency pairs.

Definition 4 (Chain) *Let* $\mathcal{P},\mathcal{R}$ *be TRSs. A (possibly infinite) sequence of pairs* $s_1 \to t_1, s_2 \to t_2, \ldots$ *from* $\mathcal{P}$ *is a* $(\mathcal{P},\mathcal{R})$-chain *iff there is a substitution* σ *with* $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ *for all* i. *It is an* innermost $(\mathcal{P},\mathcal{R})$*-chain iff* $t_i\sigma \xrightarrow{\mathrm{i}}{}_{\mathcal{R}}^* s_{i+1}\sigma$ *and* $s_i\sigma$ *is in normal form w.r.t.* $\mathcal{R}$ *for all* i. *Here, "*$\xrightarrow{\mathrm{i}}{}_{\mathcal{R}}$*" denotes innermost reductions.*

Example 5 *The sequence "(6), (6)" is a chain. The reason is that an instance of the right-hand side* $\mathsf{AP}(\mathsf{map}\,'\,\alpha_1,xs_1)$ *of (6) can reduce to an instance of its left-hand side* $\mathsf{AP}(\mathsf{map}\,'\,\alpha_2,\mathsf{cons}\,'\,x_2\,'\,xs_2)$.

Theorem 6 (Termination Criterion [4]) *A TRS* $\mathcal{R}$ *is (innermost) terminating iff there is no infinite (innermost)* $(DP(\mathcal{R}),\mathcal{R})$*-chain.*

The idea of the DP framework is to treat a set of dependency pairs $\mathcal{P}$ together with the TRS $\mathcal{R}$ and to prove absence of infinite $(\mathcal{P}, \mathcal{R})$-chains instead of examining $\to_{\mathcal{R}}$. The advantages of this framework were illustrated in [16].

Formally, a *dependency pair problem* ("DP problem")[1] consists of two TRSs $\mathcal{P}$ and $\mathcal{R}$ (where initially, $\mathcal{P} = DP(\mathcal{R})$) and a flag $e \in \{\mathbf{t}, \mathbf{i}\}$ which stands for "$\underline{\mathbf{t}}$ermination" or "$\underline{\mathbf{i}}$nnermost termination". Instead of "$(\mathcal{P}, \mathcal{R})$-chains" we also speak of "$(\mathcal{P}, \mathcal{R}, \mathbf{t})$-chains" and instead of "innermost $(\mathcal{P}, \mathcal{R})$-chains" we speak of "$(\mathcal{P}, \mathcal{R}, \mathbf{i})$-chains". Our goal is to show that there is no infinite $(\mathcal{P}, \mathcal{R}, e)$-chain. In this case, we call the problem *finite* and it is *infinite* iff it is not finite or if $\mathcal{R}$ is not terminating (if $e = \mathbf{t}$) resp. not innermost terminating (if $e = \mathbf{i}$). Thus, there can be DP problems which are both finite and infinite, but this does not cause any difficulties, cf. [16]. If one detects an infinite problem during a termination proof, one can always abort the proof, since termination has been disproved (if all proof steps were "complete", i.e., if they preserved the termination behavior).

A DP problem $(\mathcal{P}, \mathcal{R}, e)$ is *applicative* iff $\mathcal{R}$ is a TRS over an applicative signature $\mathcal{F}$, and for all $s \to t \in \mathcal{P}$, we have $t \notin \mathcal{V}$, $\{\mathrm{root}(s), \mathrm{root}(t)\} \subseteq \mathcal{F}^\sharp$, and all function symbols below the root of s or t are from $\mathcal{F}$. We also say that such a problem is an applicative DP problem *over* $\mathcal{F}$.

Termination techniques should now operate on DP problems instead of TRSs. We refer to such techniques as *dependency pair processors* ("DP processors"). Formally, a DP processor is a function *Proc* which takes a DP problem as input and returns a new set of DP problems which then have to be solved instead. Alternatively, it can also return "no". A DP processor *Proc* is *sound* if for all DP problems d, d is finite whenever $\mathit{Proc}(d)$ is not "no" and all DP problems in $\mathit{Proc}(d)$ are finite. *Proc* is *complete* if for all DP problems d, d is infinite whenever $\mathit{Proc}(d)$ is "no" or when $\mathit{Proc}(d)$ contains an infinite DP problem.

Soundness of a DP processor *Proc* is required to prove termination (in particular, to conclude that d is finite if $\mathit{Proc}(d) = \varnothing$). Completeness is needed to prove non-termination (in particular, to conclude that d is infinite if $\mathit{Proc}(d) = \mathsf{no}$).

So termination proofs in the DP framework start with the initial DP problem $(DP(\mathcal{R}), \mathcal{R}, e)$, where e depends on whether one wants to prove termination or innermost termination. Then this problem is transformed repeatedly by sound DP processors. If the final processors return empty sets of DP problems, then termination is proved. If one of the processors returns "no" and all processors used before were complete, then one has disproved termination of the TRS $\mathcal{R}$.

Example 7 *If d_0 is the initial DP problem $(DP(\mathcal{R}), \mathcal{R}, e)$ and there are sound processors Proc_0, Proc_1, Proc_2 with $\mathit{Proc}_0(d_0) = \{d_1, d_2\}$, $\mathit{Proc}_1(d_1) = \varnothing$, and $\mathit{Proc}_2(d_2) = \varnothing$, then one can conclude termination. But if $\mathit{Proc}_1(d_1) = \mathsf{no}$, and both Proc_0 and Proc_1 are complete, then one can conclude non-termination.*

3 DP Processors for Higher-Order Functions

Since we represent higher-order functions by first-order applicative TRSs, all existing techniques and DP processors for first-order TRSs can also be used for higher-order functions. However, most termination techniques rely on the outermost function symbol when comparing terms. This is also true for dependency

[1] To ease readability we use a simpler definition of *DP problems* than [16], since this simple definition suffices for the new results of this paper.

pairs and standard reduction orders. Therefore, they usually fail for applicative TRSs since here, all terms except variables and constants have the same root symbol $'$. For example, a direct termination proof of Ex. 1 is impossible with standard reduction orders and difficult[2] with dependency pairs.

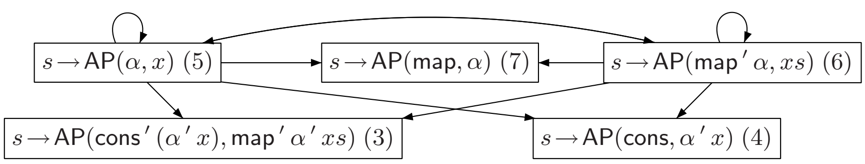
Therefore, in Sect. 3.1 and Sect. 3.2 we improve the most important processors of the DP framework in order to be successful on applicative TRSs. Moreover, we introduce a new processor in Sect. 3.3 which removes the symbol $'$ and transforms applicative TRSs and DP problems into ordinary (functional) form again. Sect. 5 shows that these contributions indeed yield a powerful termination technique for higher-order functions. Sect. 3.4 is a comparison with related work.

3.1 A DP Processor Based on the Dependency Graph

The *dependency graph* determines which pairs can follow each other in chains.

Definition 8 (Dependency Graph) *Let $(\mathcal{P}, \mathcal{R}, e)$ be a DP problem. The nodes of the* $(\mathcal{P}, \mathcal{R}, e)$-dependency graph *are the pairs of $\mathcal{P}$ and there is an arc from $s \to t$ to $u \to v$ iff $s \to t, u \to v$ is an $(\mathcal{P}, \mathcal{R}, e)$-chain.*

Example 9 *For Ex. 1, we obtain the following $(\mathcal{P}, \mathcal{R}, e)$-dependency graph for both $e = \mathbf{t}$ and $e = \mathbf{i}$. The reason is that the right-hand sides of (3), (4), and (7) have $\mathsf{cons}' (\alpha' x)$, cons, or map as their first arguments. No instance of these terms reduces to an instance of $\mathsf{map}' \alpha$ (which is the first argument of s).*



A set $\mathcal{P}'$ of dependency pairs is a *cycle* iff for all $s \to t$ and $u \to v$ in $\mathcal{P}'$, there is a path from $s \to t$ to $u \to v$ traversing only pairs of $\mathcal{P}'$. A cycle $\mathcal{P}'$ is a *strongly connected component* ("SCC") if $\mathcal{P}'$ is not a proper subset of any other cycle. As absence of infinite chains can be proved separately for every SCC, one can modularize termination proofs by decomposing a DP problem into several sub-problems.

Theorem 10 (Dependency Graph Processor [16]) *For any DP problem $(\mathcal{P}, \mathcal{R}, e)$, let Proc return $\{(\mathcal{P}_1, \mathcal{R}, e), \ldots, (\mathcal{P}_n, \mathcal{R}, e)\}$, where $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are the SCCs of the $(\mathcal{P}, \mathcal{R}, e)$-dependency graph. Then Proc is sound and complete.*

For Ex. 1, we start with the initial DP problem $(\mathcal{P}, \mathcal{R}, e)$, where $\mathcal{P} = \{(3), \ldots, (7)\}$. The only SCC of the dependency graph is $\{(5), (6)\}$. So the above processor transforms $(\mathcal{P}, \mathcal{R}, e)$ into $(\{(5), (6)\}, \mathcal{R}, e)$, i.e., (3), (4), and (7) are deleted.

Unfortunately, the dependency graph is not computable. Therefore, for automation one constructs an *estimated* graph containing at least all arcs of the real graph. The existing estimations that are used for automation [4, 17] assume that all subterms with defined root could possibly be evaluated. Therefore, they use a function $\textsc{cap}$, where $\textsc{cap}(t)$ results from replacing all subterms of t with

[2] It needs complex DP processors or base orders (e.g., non-linear polynomial orders).

defined root symbol by different fresh variables. To estimate whether $s \to t$ and $u \to v$ form a chain, one checks whether $\textsc{cap}(t)$ unifies with u (after renaming their variables). Moreover, if one regards termination instead of innermost termination, one first has to linearize $\textsc{cap}(t)$, i.e., multiple occurrences of the same variable in $\textsc{cap}(t)$ are renamed apart. Further refinements of this estimation can be found in [17]; however, they rely on the same function $\textsc{cap}$.

These estimations are not suitable for applicative TRSs. The problem is that there, *all* subterms except variables and constants have the defined root symbol $'$ and are thus replaced by variables when estimating the arcs of the dependency graph. So for Ex. 1, the estimations assume that (3) could be followed by any dependency pair in chains. The reason is that the right-hand side of (3) is $\mathsf{AP}(\mathsf{cons}' (\alpha' x), \mathsf{map}' \alpha' xs)$ and $\textsc{cap}$ replaces both arguments of AP by fresh variables, since their root symbol $'$ is defined. The resulting term $\mathsf{AP}(y, z)$ unifies with the left-hand side of every dependency pair. Therefore, the estimated dependency graph contains additional arcs from (3) to every dependency pair.

The problem is that these estimations do not check whether subterms with defined root can really be reduced further when being instantiated. For example, the first argument $\mathsf{cons}' (\alpha' x)$ of (3)'s right-hand side can never become a redex for any instantiation. The reason is that all left-hand sides of the TRS have the form $\mathsf{map}' t_1 ' t_2$. Thus, one should not replace $\mathsf{cons}' (\alpha' x)$ by a fresh variable.

Therefore, we now refine $\textsc{cap}$'s definition. If a subterm can clearly never become a redex, then it is not replaced by a variable anymore. Here, $\textsc{icap}$ is used for **i**nnermost termination proofs and $\textsc{tcap}$ differs from $\textsc{icap}$ by renaming multiple occurrences of variables, which is required when proving full **t**ermination.

Definition 11 ($\textsc{icap}$, $\textsc{tcap}$) *Let $\mathcal{R}$ be a TRS over $\mathcal{F}$, let $f \in \mathcal{F} \cup \mathcal{F}^\sharp$. We define* $\textsc{icap}$ *as follows:*

(i) $\textsc{icap}(x) = x$ *for all* $x \in \mathcal{V}$
(ii) $\textsc{icap}(f(t_1, \dots, t_n)) = f(\textsc{icap}(t_1), \dots, \textsc{icap}(t_n))$ *iff* $f(\textsc{icap}(t_1), ..., \textsc{icap}(t_n))$ *does not unify with any left-hand side of a rule from* $\mathcal{R}$
(iii) $\textsc{icap}(f(t_1, \dots, t_n))$ *is a fresh variable, otherwise*

We define $\textsc{tcap}$ *like* $\textsc{icap}$ *but in (i),* $\textsc{tcap}(x)$ *is a different fresh variable for every occurrence of* x*. Moreover in (ii), we use* $\textsc{tcap}(t_i)$ *instead of* $\textsc{icap}(t_i)$*:*

(i) $\textsc{tcap}(x)$ *is a different fresh variable for every occurrence of* $x \in \mathcal{V}$
(ii) $\textsc{tcap}(f(t_1, \dots, t_n)) = f(\textsc{tcap}(t_1), \dots, \textsc{tcap}(t_n))$ *iff* $f(\textsc{tcap}(t_1), \dots, \textsc{tcap}(t_n))$ *does not unify with any left-hand side of a rule from* $\mathcal{R}$
(iii) $\textsc{tcap}(f(t_1, \dots, t_n))$ *is a fresh variable, otherwise.*

Now one can detect that (3) should not be connected to any pair in the dependency graph, since $\textsc{icap}(\mathsf{AP}(\mathsf{cons}' (\alpha' x), \mathsf{map}' \alpha' xs)) = \mathsf{AP}(\mathsf{cons}' y, z)$ does not unify with left-hand sides of dependency pairs. Similar remarks hold for $\textsc{tcap}$. This leads to the following improved estimation.[3]

Definition 12 (Improved Estimated Dependency Graph) *In the* estimated $(\mathcal{P}, \mathcal{R}, \mathbf{t})$*-dependency graph there is an arc from* $s \to t$ *to* $u \to v$ *iff* $\textsc{tcap}(t)$

[3] Moreover, $\textsc{tcap}$ and $\textsc{icap}$ can also be combined with further refinements to approximate dependency graphs [4, 17].

and u are unifiable. In the estimated *$(\mathcal{P}, \mathcal{R}, \mathbf{i})$-dependency graph there is an arc from $s \to t$ to $u \to v$ iff* $\textsc{icap}(t)$ *and u are unifiable by an mgu μ (after renaming their variables) such that $s\mu$ and $u\mu$ are in normal form w.r.t. $\mathcal{R}$.*

Now the estimated graph is identical to the real dependency graph in Ex. 9, both in the termination and innermost termination case.

Theorem 13 (Soundness of the Improved Estimation) *The dependency graph is a subgraph of the estimated dependency graph.*

Proof. We first regard the termination case $e = \mathbf{t}$. Here we have to show that if $s \to t$, $u \to v$ is a $(\mathcal{P}, \mathcal{R}, \mathbf{t})$-chain, then $\textsc{tcap}(t)$ and u unify. To this end, we prove the following claim for all terms t and u and all substitutions σ:

$$\textsc{tcap}(t)\sigma \to_{\mathcal{R}}^* u \text{ implies that } u = \textsc{tcap}(t)\delta \text{ for some substitution } \delta \tag{8}$$

The claim (8) immediately implies the theorem in the termination case. The reason is that obviously $t = \textsc{tcap}(t)\sigma'$ for a suitable substitution σ'. Hence, if $s \to t$, $u \to v$ is a $(\mathcal{P}, \mathcal{R}, \mathbf{t})$-chain (i.e., if $t\sigma \to_{\mathcal{R}}^* u\sigma$ for some σ), then we have $t\sigma = \textsc{tcap}(t)\sigma'\sigma$ and hence $u\sigma = \textsc{tcap}(t)\delta$ for some substitution δ by (8). Hence, since $\textsc{tcap}(t)$ and u are variable disjoint, they are unifiable.

To prove (8), it suffices to regard the case $\textsc{tcap}(t)\sigma \to_{\mathcal{R}} u$, since then (8) follows by induction on the length of the reduction. We use induction on t. If $\textsc{tcap}(t) \in \mathcal{V}$, the claim (8) is trivial. Otherwise, $t = f(t_1, \ldots, t_n)$ and $\textsc{tcap}(t) = f(\textsc{tcap}(t_1), \ldots, \textsc{tcap}(t_n))$, where $\textsc{tcap}(t)$ does not unify with any left-hand side of $\mathcal{R}$. Thus, there is an $1 \leq i \leq n$ with $\textsc{tcap}(t_i)\sigma \to_{\mathcal{R}}^* u_i$ and $u = f(\textsc{tcap}(t_1)\sigma, \ldots, u_i, \ldots, \textsc{tcap}(t_n)\sigma)$. By the induction hypothesis we obtain $u_i = \textsc{tcap}(t_i)\delta$ for some δ. As the variables of all $\textsc{tcap}(t_j)$ are disjoint, we can extend δ to operate like σ on the variables of $\textsc{tcap}(t_j)$ for $j \neq i$. Then we have $u = f(\textsc{tcap}(t_1)\delta, \ldots, \textsc{tcap}(t_i)\delta, \ldots, \textsc{tcap}(t_n)\delta) = \textsc{tcap}(t)\delta$, as desired.

For innermost termination, we prove the following for all terms s, t, u with $\mathcal{V}(t) \subseteq \mathcal{V}(s)$ and all substitutions σ where $\sigma(x)$ is in normal form for $x \in \mathcal{V}(s)$:

$$\text{If } \textsc{icap}(t)\sigma \to_{\mathcal{R}}^* u \text{ then } u = \textsc{icap}(t)\delta \text{ for a } \delta \text{ with } \delta(x) = \sigma(x) \text{ for } x \in \mathcal{V}(s) \tag{9}$$

The claim (9) immediately implies the theorem in the innermost termination case. Note that $t = \textsc{icap}(t)\sigma'$ for a σ' where $\sigma'(x) = x$ for all $x \in \mathcal{V}(s)$. Hence, if $s \to t$, $u \to v$ is a $(\mathcal{P}, \mathcal{R}, \mathbf{i})$-chain (i.e., if $t\sigma \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* u\sigma$ for some σ where both $s\sigma$ and $u\sigma$ are in normal form), then $t\sigma = \textsc{icap}(t)\sigma'\sigma$ and hence $u\sigma = \textsc{icap}(t)\delta$ for some δ with $\delta(x) = \sigma(x)$ for all $x \in \mathcal{V}(s)$ by (9). Hence, since $\textsc{icap}(t)$ and u are variable disjoint and since $s\delta = s\sigma$ and $u\sigma$ are in normal form, $\textsc{icap}(t)$ and u are unifiable by a substitution that instantiates both s and u to normal forms.

To prove (9) it again suffices to regard $\textsc{icap}(t)\sigma \to_{\mathcal{R}} u$. We use induction on t. Note that $t \notin \mathcal{V}$, since otherwise σ would instantiate t by a term which is not in normal form. So if $\textsc{icap}(t) \in \mathcal{V}$ then $\textsc{icap}(t)$ is a fresh variable and (9) is trivial. Otherwise, $t = f(t_1, \ldots, t_n)$ and $\textsc{icap}(t) = f(\textsc{icap}(t_1), \ldots, \textsc{icap}(t_n))$. As in the termination case, we obtain $u = f(\textsc{icap}(t_1)\sigma, \ldots, u_i, \ldots, \textsc{icap}(t_n)\sigma)$, and $u_i = \textsc{icap}(t_i)\delta$ for some δ where $\delta(x) = \sigma(x)$ for all $x \in \mathcal{V}(s)$. Apart from the variables of s, the terms $\textsc{icap}(t_j)$ with $j \neq i$ only contain fresh variables not occurring in $\textsc{icap}(t_i)$. Thus, we can extend δ to operate like σ on these variables. Then we have $u = f(\textsc{icap}(t_1)\delta, \ldots, \textsc{icap}(t_i)\delta, \ldots, \textsc{icap}(t_n)\delta) = \textsc{icap}(t)\delta$, as desired. $\square$

3.2 DP Processors Based on Orders and on Usable Rules

Classical techniques for automated termination proofs try to find a *reduction order* $\succ$ such that $l \succ r$ holds for all rules $l \to r$. In practice, most orders are *simplification orders* [10]. However, termination of many important TRSs cannot be proved with such orders directly. Therefore, the following processor allows us to use such orders in the DP framework instead. It generates constraints which should be satisfied by a *reduction pair* [22] $(\succsim, \succ)$ where $\succsim$ is reflexive, transitive, monotonic, and stable and $\succ$ is a stable well-founded order compatible with $\succsim$ (i.e., $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$). Now one can use existing techniques to search for suitable relations $\succsim$ and $\succ$, and in this way, classical simplification orders can prove termination of TRSs where they would have failed otherwise.

For a DP problem $(\mathcal{P}, \mathcal{R}, e)$, the constraints require that at least one rule in $\mathcal{P}$ is strictly decreasing (w.r.t. $\succ$) and all remaining rules in $\mathcal{P}$ and $\mathcal{R}$ are weakly decreasing (w.r.t. $\succsim$). Requiring $l \succsim r$ for $l \to r \in \mathcal{R}$ ensures that in chains $s_1 \to t_1, s_2 \to t_2, \ldots$ with $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$, we have $t_i\sigma \succsim s_{i+1}\sigma$. Hence, if a reduction pair satisfies these constraints, then the strictly decreasing pairs of $\mathcal{P}$ cannot occur infinitely often in chains. Thus, the following processor deletes these pairs from $\mathcal{P}$. For any TRS $\mathcal{P}$ and any relation $\succ$, let $\mathcal{P}_\succ = \{s \to t \in \mathcal{P} \mid s \succ t\}$.

Theorem 14 (Reduction Pair Processor [16]) *Let $(\succsim, \succ)$ be a reduction pair. Then the following DP processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{R}, e)$, Proc returns*

- $\{(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{R}, e)\}$, *if* $\mathcal{P}_\succ \cup \mathcal{P}_\succsim = \mathcal{P}$ *and* $\mathcal{R}_\succsim = \mathcal{R}$
- $\{(\mathcal{P}, \mathcal{R}, e)\}$, *otherwise*

DP problems $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ for *innermost* termination can be simplified by replacing the second component $\mathcal{R}$ by those rules from $\mathcal{R}$ that are *usable* for $\mathcal{P}$ (i.e., by the *usable rules* of $\mathcal{P}$). Then by Thm. 14, a weak decrease $l \succsim r$ is not required for all rules but only for the usable rules. As defined in [4], the *usable rules* of a term t contain all f-rules for all function symbols f occurring in t. Moreover, if f's rules are usable and there is a rule $\mathsf{f}(\ldots) \to r$ in $\mathcal{R}$ whose right-hand side r contains a symbol g, then g is usable, too. The *usable rules* of a TRS $\mathcal{P}$ are defined as the usable rules of its right-hand sides.

For instance, after applying the dependency graph processor to Ex. 1, we have the remaining dependency pairs (5) and (6) with the right-hand sides $\mathsf{AP}(\alpha, x)$ and $\mathsf{AP}(\mathsf{map}\,'\,\alpha, xs)$. While $\mathsf{AP}(\alpha, x)$ has no usable rules, $\mathsf{AP}(\mathsf{map}\,'\,\alpha,\ xs)$ contains the defined function symbol $'$ and therefore, all $'$-rules are usable.

This indicates that the definition of usable rules has to be improved to handle applicative TRSs successfully. Otherwise, whenever $'$ occurs in the right-hand side of a dependency pair, then *all* rules (except rules of the form $\mathsf{f} \to \ldots$) would be usable. The problem is that the current definition of "usable rules" assumes that all $'$-rules can be applied to any subterm with the root symbol $'$.

Thus, we refine the definition of usable rules. Now a subterm starting with $'$ only influences the computation of the usable rules if this subterm can potentially start new reductions. To detect this, we again use the function ICAP from Def. 11. For example, $\mathsf{map}\,'\,\alpha$ can never be reduced if α is instantiated by a normal form, since $\mathsf{map}'\,\alpha$ does not unify with the left-hand side of any rule. Therefore, the right-hand side $\mathsf{AP}(\mathsf{map}'\,\alpha, xs)$ of (6) should not have any usable rules.[4]

Definition 15 (Improved Usable Rules) *For a DP problem* $(\mathcal{P}, \mathcal{R}, \mathbf{i})$*, we define the* usable rules $\mathcal{U}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t)$*. Here* $\mathcal{U}(t) \subseteq \mathcal{R}$ *is the smallest set with:*

- *If* $t = f(t_1, \ldots, t_n)$*,* $f \in \mathcal{F} \cup \mathcal{F}^\sharp$*, and* $f(\textsc{icap}(t_1), \ldots, \textsc{icap}(t_n))$ *unifies with a left-hand side* l *of a rule* $l \to r \in \mathcal{R}$*, then* $l \to r \in \mathcal{U}(t)$*.*
- *If* $l \to r \in \mathcal{U}(t)$*, then* $\mathcal{U}(r) \subseteq \mathcal{U}(t)$*.*
- *If* t' *is a subterm of* t*, then* $\mathcal{U}(t') \subseteq \mathcal{U}(t)$*.*

Theorem 16 (Usable Rule Processor) *For a DP problem* $(\mathcal{P}, \mathcal{R}, e)$*, let Proc return* $\{\, (\mathcal{P}, \mathcal{U}(\mathcal{P}), \mathbf{i}) \,\}$ *if* $e = \mathbf{i}$ *and* $\{\, (\mathcal{P}, \mathcal{R}, e) \,\}$ *otherwise. Then Proc is sound.*[5]

Proof. Let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite $(\mathcal{P}, \mathcal{R}, \mathbf{i})$-chain, i.e., $t_i\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ and $s_i\sigma$ is in normal form for some σ. We show that the innermost reduction from $t_i\sigma$ to $s_{i+1}\sigma$ only uses rules from $\mathcal{U}(\mathcal{P})$. It suffices to prove the following for all normal substitutions σ (i.e., $\sigma(x)$ is in normal form for all $x \in \mathcal{V}$), since it implies the above claim by induction on the length of the reduction:

> $t\sigma \xrightarrow{\mathrm{i}}_{\{l \to r\}} s$ with $l \to r \in \mathcal{R}$ implies that $l \to r \in \mathcal{U}(t)$ and that there is some term u and a normal substitution δ with $\delta(x) = \sigma(x)$ for $x \in \mathcal{V}(l)$, such that $s = u\delta$, $\mathcal{U}(t) \supseteq \mathcal{U}(u)$, and $\textsc{icap}(u) = \textsc{icap}(t)\rho$ for a substitution ρ whose domain only contains fresh variables introduced in $\textsc{icap}(t)$

We use induction on t. As σ is normal, we have $t \notin \mathcal{V}$. So $t = f(t_1, \ldots, t_n)$. If $t\sigma = f(t_1\sigma, \ldots, t_n\sigma) = l\tau \xrightarrow{\mathrm{i}}_{\mathcal{R}} r\tau = s$, then obviously, $f(\textsc{icap}(t_1), \ldots, \textsc{icap}(t_n))$ unifies with l. Thus, $l \to r \in \mathcal{U}(t)$ and by Def. 15 we have $\mathcal{U}(r) \subseteq \mathcal{U}(t)$. Moreover, any term is an instance of $\textsc{icap}(t)$, since $\textsc{icap}(t)$ is a fresh variable. As τ is normal, we can choose $u = r$, $\rho = [\textsc{icap}(t) \,/\, \textsc{icap}(u)]$, $\delta(x) = \tau(x)$ for $x \in \mathcal{V}(l)$, and $\delta(x) = \sigma(x)$ otherwise.

In the remaining case, we have $t\sigma \xrightarrow{\mathrm{i}}_{\mathcal{R}} f(t_1\sigma, \ldots, s_i, \ldots, t_n\sigma) = s$ where $t_i\sigma \xrightarrow{\mathrm{i}}_{\mathcal{R}} s_i$. By the induction hypothesis, the rule used for the reduction was from $\mathcal{U}(t_i)$ and thus, it is also contained in $\mathcal{U}(t)$ by Def. 15. By induction, there exist u_i, δ, and ρ such that $s_i = u_i\delta$, $\mathcal{U}(t_i) \supseteq \mathcal{U}(u_i)$, and $\textsc{icap}(u_i) = \textsc{icap}(t_i)\rho$. Thus, $s = f(t_1\sigma, \ldots, u_i\delta, \ldots, t_n\sigma) = f(t_1, \ldots, u_i, \ldots, t_n)\delta$, since w.l.o.g. l is variable disjoint from t, and we define $u = f(t_1, \ldots, u_i, \ldots, t_n)$. Now if $\textsc{icap}(t)$ is a fresh variable, then obviously $\textsc{icap}(u)$ is an instance of $\textsc{icap}(t)$ (i.e., $\rho = [\textsc{icap}(t) \,/\, \textsc{icap}(u)]$). Otherwise $\textsc{icap}(t) = f(\textsc{icap}(t_1), \ldots, \textsc{icap}(t_i), \ldots, \textsc{icap}(t_n))$ and for the term $u' = f(\textsc{icap}(t_1), \ldots, \textsc{icap}(u_i), \ldots, \textsc{icap}(t_n))$ we have $u' = \textsc{icap}(t)\rho$ by the induction hypothesis and since ρ only instantiates the fresh variables in $\textsc{icap}(t_i)$. Since $\textsc{icap}(t)$ does not unify with any left-hand side of a rule from $\mathcal{R}$, this also holds for u' and thus, $\textsc{icap}(u) = u' = \textsc{icap}(t)\rho$. Together with the fact that $\mathcal{U}(t_i) \supseteq \mathcal{U}(u_i)$, this also implies that $\mathcal{U}(t) \supseteq \mathcal{U}(u)$. $\square$

[4] Our new definition of usable rules can also be combined with other techniques to reduce the set of usable rules [14] and it can also be applied for dependency graph estimations or other DP processors that rely on usable rules [16, 17].

[5] Incompleteness is only due to our simplified definition of "DP problems". With the full definition of "DP problems" from [16], the processor is complete [16, Thm. 27].

Example 17 *In Ex. 1, now the dependency pairs in the remaining DP problem* $(\{(5),(6)\},\mathcal{R},\mathbf{i})$ *have no usable rules. Thus, Thm. 16 transforms this DP problem into* $(\{(5),(6)\},\varnothing,\mathbf{i})$*. Then with the processor of Thm. 14 we try to find a reduction pair such that* (5) *and* (6) *are decreasing. Any simplification order* $\succ$ *(even the embedding order) makes both pairs strictly decreasing:* $s \succ \mathsf{AP}(\alpha, x)$ *and* $s \succ \mathsf{AP}(\mathsf{map}'\,\alpha,\, xs)$ *for* $s = \mathsf{AP}(\mathsf{map}'\,\alpha,\, \mathsf{cons}'\,x'\,xs)$*. Thus, both dependency pairs are removed and the resulting DP problem* $(\varnothing,\mathcal{R},\mathbf{i})$ *is transformed into the empty set by the dependency graph processor of Thm. 10. So innermost termination of* map *can now easily be proved automatically. Note that this TRS is non-overlapping and thus, it belongs to a well-known class where innermost termination implies termination. Hence, we also proved termination of* map.

In [29], we showed that under certain conditions, the usable rules of [4] can also be used to prove full instead of just innermost termination (for arbitrary TRSs). Then, even for termination, it is enough to require $l \succsim r$ just for the usable rules in Thm. 14. This result also holds for the new improved usable rules of Def. 15, provided that one uses TCAP instead of ICAP in their definition.

3.3 A DP Processor to Transform Applicative to Functional Form

Some applicative DP problems can be transformed (back) to ordinary functional form. In particular, this holds for problems resulting from first-order functions (encoded by currying). This transformation is advantageous: e.g., the processor in Thm. 14 is significantly more powerful for DP problems in functional form, since standard reduction orders focus on the root symbol when comparing terms.

Example 18 *We extend the* map-*TRS by the following rules for* minus *and* div*. Note that a direct termination proof with simplification orders is impossible.*

$$\mathsf{minus}'x'\mathsf{0} \to x \quad (10) \qquad \mathsf{div}'\mathsf{0}'(\mathsf{s}'y) \to \mathsf{0} \quad (12)$$

$$\mathsf{minus}'(\mathsf{s}'x)'(\mathsf{s}'y) \to \mathsf{minus}'x'y \quad (11) \qquad \mathsf{div}'(\mathsf{s}'x)'(\mathsf{s}'y) \to \mathsf{s}'(\mathsf{div}'(\mathsf{minus}'x'y)'(\mathsf{s}'y)) \quad (13)$$

While map *is really a higher-order function,* minus *and* div *correspond to first-order functions. It again suffices to verify innermost termination, since this TRS* $\mathcal{R}$ *is non-overlapping. The improved estimated dependency graph has three SCCs corresponding to* map*,* minus*, and* div*. Thus, by the dependency graph and the usable rule processors (Thm. 10 and 16), the initial DP problem* $(DP(\mathcal{R}),\mathcal{R},\mathbf{i})$ *is transformed into three new problems. The first problem* $(\{(5),(6)\},\varnothing,\mathbf{i})$ *for* map *can be solved as before. The DP problems for* minus *and* div *are:*

$$(\{\mathsf{AP}(\mathsf{minus}'\,(\mathsf{s}'\,x),\, \mathsf{s}'\,y) \to \mathsf{AP}(\mathsf{minus}'\,x, y)\},\, \varnothing,\, \mathbf{i}) \quad (14)$$

$$(\{\mathsf{AP}(\mathsf{div}'\,(\mathsf{s}'\,x),\, \mathsf{s}'\,y) \to \mathsf{AP}(\mathsf{div}'\,(\mathsf{minus}'\,x'\,y),\, \mathsf{s}'\,y)\},\, \{(10),(11)\},\, \mathbf{i}) \quad (15)$$

Since (14) and (15) do not contain map *anymore, one would like to change them back to conventional functional form. Then they could be replaced by the following DP problems. Here, every (new) function symbol is labelled by its arity.*

$$(\{\mathsf{MINUS}^2(\mathsf{s}^1(x),\mathsf{s}^1(y)) \to \mathsf{MINUS}^2(x,y)\},\, \varnothing,\, \mathbf{i}) \quad (16)$$

$$\begin{aligned}&(\{\mathsf{DIV}^2(\mathsf{s}^1(x),\mathsf{s}^1(y)) \to \mathsf{DIV}^2(\mathsf{minus}^2(x,y),\mathsf{s}^1(y))\},\\ &\;\{\mathsf{minus}^2(x,\mathsf{0}^0) \to x,\, \mathsf{minus}^2(\mathsf{s}^1(x),\mathsf{s}^1(y)) \to \mathsf{minus}^2(x,y)\},\, \mathbf{i})\end{aligned} \quad (17)$$

These DP problems are easy to solve: for example, the constraints of the reduction pair processor (Thm. 14) are satisfied by the polynomial order which maps $\mathsf{s}^1(x)$ *to* $x + 1$, $\mathsf{minus}^2(x, y)$ *to* x, *and every other symbol to the sum of its arguments. Thus, termination could immediately be proved automatically.*

Now we characterize those applicative TRSs which correspond to first-order functions and can be translated into functional form. In these TRSs, for any function symbol f there is a number n (called its *arity*) such that f only occurs in terms of the form $\mathsf{f}\,'\,t_1\,'\,\ldots\,'\,t_n$. So there are no applications with too few or too many arguments. Moreover, there are no terms $x\,'\,t$ where the first argument of $'$ is a variable. Def. 19 extends this idea from TRSs to DP problems.

Definition 19 (Arity and Proper Terms) *Let* $(\mathcal{P}, \mathcal{R}, e)$ *be an applicative DP problem over* $\mathcal{F}$. *For each* $\mathsf{f} \in \mathcal{F} \setminus \{'\}$ *let* $\mathrm{arity}(\mathsf{f}) = max\{\, n \mid \mathsf{f}\,'\,t_1\,'\,\ldots\,'\,t_n$ *or* $(\mathsf{f}\,'\,t_1\,'\,\ldots\,'\,t_n)^\sharp$ *occurs in* $\mathcal{P} \cup \mathcal{R}\,\}$. *A term* t *is* proper *iff* $t \in \mathcal{V}$ *or* $t = \mathsf{f}\,'\,t_1\,'\,\ldots\,'\,t_n$ *or* $t = (\mathsf{f}\,'\,t_1 \ldots\,'\,t_n)^\sharp$ *where in the last two cases,* $\mathrm{arity}(\mathsf{f}) = n$ *and all* t_i *are proper. Moreover,* $(\mathcal{P}, \mathcal{R}, e)$ *is* proper *iff all terms in* $\mathcal{P} \cup \mathcal{R}$ *are proper.*

The DP problems (14) and (15) for minus and div are proper. Here, minus and div have arity 2, s has arity 1, and $\mathsf{0}$ has arity 0. But the problem $(\{(5), (6)\}, \varnothing, \mathbf{i})$ for map is not proper as (5) contains the subterm $\mathsf{AP}(\alpha, x)$ with $\alpha \in \mathcal{V}$.

The following transformation translates proper terms from applicative to functional form. To this end, $\mathsf{f}\,'\,t_1\,'\,\ldots\,'\,t_n$ is replaced by $\mathsf{f}^n(\ldots)$, where n is f's arity (as defined in Def. 19) and f^n is a new n-ary function symbol. In this way, (14) and (15) were transformed into (16) and (17) in Ex. 18.

Definition 20 ($\mathcal{A}$ Transformation) $\mathcal{A}$ *maps every proper term from* $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$ *to a term from* $\mathcal{T}(\{\mathsf{f}^n, \mathsf{F}^n \mid \mathsf{f} \in \mathcal{F} \setminus \{'\}, \mathrm{arity}(\mathsf{f}) = n\}, \mathcal{V})$*:*

- $\mathcal{A}(x) = x$ *for all* $x \in \mathcal{V}$
- $\mathcal{A}(\mathsf{f}\,'\,t_1\,'\,\ldots\,'\,t_n) = \mathsf{f}^n(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n))$ *for all* $\mathsf{f} \in \mathcal{F} \setminus \{'\}$
- $\mathcal{A}((\mathsf{f}\,'\,t_1\,'\,\ldots\,'\,t_n)^\sharp) = \mathsf{F}^n(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n))$ *for all* $\mathsf{f} \in \mathcal{F} \setminus \{'\}$

For any TRS $\mathcal{R}$ *with proper terms, let* $\mathcal{A}(\mathcal{R}) = \{\mathcal{A}(l) \to \mathcal{A}(r) \mid l \to r \in \mathcal{R}\}$.

In the following, we say that a substitution σ is *proper* if $\sigma(x)$ is proper for all $x \in \mathcal{V}$ and for a proper substitution σ we define $\mathcal{A}(\sigma)$ as the substitution with $\mathcal{A}(\sigma)(x) = \mathcal{A}(\sigma(x))$. Moreover, let $\mathcal{T}_{proper}$ be the set of proper terms from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$ and let $\mathcal{T}_{func} = \mathcal{T}(\{\mathsf{f}^n, \mathsf{F}^n \mid \mathsf{f} \in \mathcal{F} \setminus \{'\}, \mathrm{arity}(\mathsf{f}) = n\}, \mathcal{V})$.

Lemma 21 (Properties of $\mathcal{A}$) *Let* $(\mathcal{P}, \mathcal{R}, e)$ *be a proper DP problem and let* $\mathcal{A}^{-1}$ *be the inverse mapping to* $\mathcal{A}$. *For all* t, s *from* $\mathcal{T}_{proper}$, *all* u, v *from* $\mathcal{T}_{func}$, *all substitutions* $\sigma : \mathcal{V} \to \mathcal{T}_{proper}$, *and all substitutions* $\delta : \mathcal{V} \to \mathcal{T}_{func}$, *we have*

(a) $\mathcal{A}(t\sigma) = \mathcal{A}(t)\mathcal{A}(\sigma)$ *and* $\mathcal{A}^{-1}(p\delta) = \mathcal{A}^{-1}(p)\mathcal{A}^{-1}(\delta)$
(b) $t \to_{\mathcal{R}}^m s$ *implies* $\mathcal{A}(t) \to_{\mathcal{A}(\mathcal{R})}^m \mathcal{A}(s)$, *and* $u \to_{\mathcal{A}(\mathcal{R})}^m v$ *implies* $\mathcal{A}^{-1}(u) \to_{\mathcal{R}}^m \mathcal{A}^{-1}(v)$
(c) $t \xrightarrow{\mathrm{i}}{}_{\mathcal{R}}^m s$ *implies* $\mathcal{A}(t) \xrightarrow{\mathrm{i}}{}_{\mathcal{A}(\mathcal{R})}^m \mathcal{A}(s)$, *and* $u \xrightarrow{\mathrm{i}}{}_{\mathcal{A}(\mathcal{R})}^m v$ *implies* $\mathcal{A}^{-1}(u) \xrightarrow{\mathrm{i}}{}_{\mathcal{R}}^m \mathcal{A}^{-1}(v)$

Proof. The claim in (a) is proved by straightforward structural inductions. For (b), one easily shows that $t \to_{\mathcal{R}} s$ iff $\mathcal{A}(t) \to_{\mathcal{A}(\mathcal{R})} \mathcal{A}(s)$ by structural induction on t. Then the claim for $m > 1$ follows by induction. The proof of (c) is as for (b) since t is normal iff $\mathcal{A}(t)$ is normal for any $t \in \mathcal{T}_{proper}$ by (b). □

We want to define a DP processor which replaces proper DP problems $(\mathcal{P}, \mathcal{R}, e)$ by $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), e)$. For its soundness, we have to show that every $(\mathcal{P}, \mathcal{R}, e)$-chain results in an $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), e)$-chain, i.e., that $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ implies $\mathcal{A}(t_i)\sigma' \to_{\mathcal{A}(\mathcal{R})}^* \mathcal{A}(s_{i+1})\sigma'$ for some substitution σ'. The problem is that although all terms in $\mathcal{P}$ and $\mathcal{R}$ are proper, the substitution σ may introduce non-proper terms.

Therefore, we now show that every $(\mathcal{P}, \mathcal{R}, e)$-chain which uses a substitution σ can also be obtained by using a substitution $\mathcal{Z}(\sigma)$ with *proper* terms. Here, $\mathcal{Z}$ transforms arbitrary terms t, s into proper ones such that $t \to_{\mathcal{R}}^* s$ implies $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$. $\mathcal{Z}$ replaces terms where a variable is on the first argument of $'$ or AP or where a function symbol f has too few arguments by a fresh variable $\bot$. If f is applied to more arguments than its arity n, the first n arguments are modified by applying them to the arguments on positions $n+1, n+2, \ldots$ Afterwards, the arguments on the positions $n+1, n+2, \ldots$ are deleted.

As an example, regard the non-proper term $t = \mathsf{minus}\,'\,\mathsf{s}\,'\,\mathsf{0}\,'\,x$ where the symbol minus with arity 2 is applied to 3 arguments. $\mathcal{Z}$ removes the argument x and modifies the arguments s and $\mathsf{0}$ by applying them to x. So t is replaced by $\mathsf{minus}\,'\,(\mathsf{s}\,'\,x)\,'\,(\mathsf{0}\,'\,x)$. Now $\mathcal{Z}$ is called recursively on the subterms and therefore, the argument x of the symbol $\mathsf{0}$ with arity 0 is removed. Hence, $\mathcal{Z}(t) = \mathsf{minus}\,'\,(\mathsf{s}\,'\,x)\,'\,\mathsf{0}$. Note that for the original non-proper term t, we have $t \to_{\mathcal{R}} \mathsf{s}\,'\,x$ by the collapsing minus-rule (10). Similarly, we now also have $\mathcal{Z}(t) \to_{\mathcal{R}} \mathcal{Z}(\mathsf{s}\,'\,x) = \mathsf{s}\,'\,x$. In the following, let $?_j$ denote function symbols from $\{', \mathsf{AP}\}$ where we now use infix notation for both $'$ and AP to ease readability.

Definition 22 ($\mathcal{Z}$ Transformation) *$\mathcal{Z}$ is the following transformation from terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$ to $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V} \cup \{\bot\})$, where $\bot$ is a fresh variable. Here, $x \in \mathcal{V}$ and $f \in \{\mathsf{f}, \mathsf{F}\}$ for some $\mathsf{f} \in \mathcal{F} \setminus \{'\}$ with* $\mathrm{arity}(\mathsf{f}) = n$.

- $\mathcal{Z}(x) = x$
- $\mathcal{Z}(f \,?_1\, t_1 \,?_2\, \ldots \,?_k\, t_k) =$
  $\mathsf{f}\,'\,\mathcal{Z}(t_1 \,?_{n+1}\, t_{n+1} \,?_{n+2}\, \ldots \,?_k\, t_k)\,'\,\ldots\,'\,\mathcal{Z}(t_n \,?_{n+1}\, t_{n+1} \,?_{n+2}\, \ldots \,?_k\, t_k)$,
  if $k \geq n$ and either $?_n = '$ or both $n = 0$ and $f \in \mathcal{F}$
- $\mathcal{Z}(f \,?_1\, t_1 \,?_2\, \ldots \,?_k\, t_k) =$
  $(\mathsf{f}\,'\,\mathcal{Z}(t_1 \,?_{n+1}\, t_{n+1} \,?_{n+2}\, \ldots \,?_k\, t_k)\,'\,\ldots\,'\,\mathcal{Z}(t_n \,?_{n+1}\, t_{n+1} \,?_{n+2}\, \ldots \,?_k\, t_k))^\sharp$,
  if $k \geq n$ and either $?_n = \mathsf{AP}$ or both $n = 0$ and $f \in \mathcal{F}^\sharp$
- $\mathcal{Z}(t) = \bot$*, for all other $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$*

Moreover, for any substitution σ, $\mathcal{Z}(\sigma)$ is the substitution with $\mathcal{Z}(\sigma)(x) = \mathcal{Z}(\sigma(x))$.

Lemma 23 (Properties of $\mathcal{Z}$) *Let $(\mathcal{P}, \mathcal{R}, e)$ be a proper DP problem over $\mathcal{F}$, let t and s be from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$, and let $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$.*

(a) $\mathcal{Z}(t)$ is proper
(b) If t is proper then $\mathcal{Z}(t\sigma) = t\mathcal{Z}(\sigma)$
(c) $t \to_{\mathcal{R}}^* s$ implies $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$

Proof. The claims (a) and (b) are easily obtained by structural induction on t. For (c), it suffices to show that $t \to_{\mathcal{R}} s$ implies $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$. We use induction on t with the embedding order as induction relation. Obviously, $t \notin \mathcal{V}$. First let $t = f \,?_1\, t_1 \,?_2\, \ldots \,?_k\, t_k$, where $\mathrm{arity}(f) = n$ and $k \geq n$. We only regard the case where either ${}^{?_n} = {}'$ or both $n = 0$ and $f \in \mathcal{F}$, since the other case is analogous. First assume that s is obtained by reducing $t_i \to_{\mathcal{R}} s_i$. If $i \leq n$, then $\mathcal{Z}(t_i \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k) \to_{\mathcal{R}}^* \mathcal{Z}(s_i \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k)$ by the induction hypothesis. Hence, $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$ by the definition of $\mathcal{Z}$. If $i > n$, then $\mathcal{Z}(t_j \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_i}\, t_i \,{}^{?_{i+1}} \ldots {}^{?_k}\, t_k) \to^* \mathcal{Z}(t_j \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_i}\, s_i \,{}^{?_{i+1}} \ldots {}^{?_k}\, t_k)$ for all $1 \leq j \leq n$ by the induction hypothesis. By the definition of $\mathcal{Z}$ we again get $\mathcal{Z}(t) \to_{\mathcal{R}}^* \mathcal{Z}(s)$. Otherwise, $t = l\sigma \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k \to r\sigma \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k = s$. Let $\overline{\sigma}$ be the substitution with $\overline{\sigma}(x) = \sigma(x) \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k$. Then $\mathcal{Z}(q\sigma \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k) = q\mathcal{Z}(\overline{\sigma})$ can easily be shown by structural induction for any proper term q. Hence, $l = \mathsf{f}\,'\,l_1\,' \ldots '\,l_n$ and

$$\begin{array}{rl} \mathcal{Z}(t) = & \mathsf{f}\,'\,\mathcal{Z}(l_1\sigma \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k)\,' \ldots '\,\mathcal{Z}(l_n\sigma \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k) \\ = & \mathsf{f}\,'\,l_1\mathcal{Z}(\overline{\sigma})\,' \ldots '\,l_n\mathcal{Z}(\overline{\sigma}) \\ = & l\mathcal{Z}(\overline{\sigma}) \\ \to_{\mathcal{R}} & r\mathcal{Z}(\overline{\sigma}) \\ = & \mathcal{Z}(r\sigma \,{}^{?_{n+1}}\, t_{n+1} \,{}^{?_{n+2}} \ldots {}^{?_k}\, t_k) \\ = & \mathcal{Z}(s). \end{array}$$

Otherwise, we have $t = x \,{}^{?_1}\, t_1 \,{}^{?_2} \ldots {}^{?_k}\, t_k$ or $t = f \,{}^{?_1}\, t_1 \,{}^{?_2} \ldots {}^{?_k}\, t_k$ where $k < \mathrm{arity}(f)$. Here, s is obtained by reducing $t_i \to_{\mathcal{R}} s_i$ for some i, since $(\mathcal{P}, \mathcal{R}, e)$ is proper. Thus, $\mathcal{Z}(t) = \bot = \mathcal{Z}(s)$. □

However, the transformation $\mathcal{Z}$ cannot be used in the innermost case, since $t \xrightarrow{\mathsf{i}}_{\mathcal{R}} s$ does not imply $\mathcal{Z}(t) \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* \mathcal{Z}(s)$. To see this, regard the TRS with the rules $\mathsf{f}\,'\,(\mathsf{g}\,'\,x\,'\,y\,'\,z) \to z$ and $\mathsf{g}\,'\,x\,'\,x\,'\,y \to \mathsf{0}$. We obtain $t \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{0}$ for the non-proper term $t = \mathsf{f}\,'\,(\mathsf{g}\,'\,(\mathsf{0}\,'\,x)\,'\,(\mathsf{0}\,'\,y)\,'\,\mathsf{0})$, whereas $\mathcal{Z}(t) = \mathsf{f}\,'\,(\mathsf{g}\,'\,\mathsf{0}\,'\,\mathsf{0}\,'\,\mathsf{0})$ only reduces innermost to $\mathsf{f}\,'\,\mathsf{0}$. So the problem is that $\mathcal{Z}$ can make different subterms equal by eliminating "superfluous" arguments.

Therefore, we now introduce an alternative transformation $\mathcal{I}$ from arbitrary to proper terms which simply replaces non-proper subterms t by a fresh variable $\bot_t$. So in the above example, we have $\mathcal{I}(t) = \mathsf{f}\,'\,(\mathsf{g}\,'\,\bot_{\mathsf{0}\,'\,x}\,'\,\bot_{\mathsf{0}\,'\,y}\,'\,\mathsf{0})$. Since $\bot_{\mathsf{0}\,'\,x} \neq \bot_{\mathsf{0}\,'\,y}$, we now obtain $\mathcal{I}(t) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathcal{I}(\mathsf{0}) = \mathsf{0}$, as desired. Now $t \xrightarrow{\mathsf{i}}_{\mathcal{R}} s$ indeed implies $\mathcal{I}(t) \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* \mathcal{I}(s)$, provided that t has the form $q\sigma$ for a proper term q and a *normal* substitution σ, i.e., $\sigma(x)$ is in normal form w.r.t. $\mathcal{R}$ for all $x \in \mathcal{V}$.[6]

Definition 24 ($\mathcal{I}$ Transformation) *$\mathcal{I}$ is the transformation from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$ to $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V} \cup \mathcal{V}')$, where $\mathcal{V}' = \{\bot_t \mid t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})\}$ are fresh variables.*

- $\mathcal{I}(x) = x$ *for all* $x \in \mathcal{V}$
- $\mathcal{I}(\mathsf{f}\,'\,t_1\,' \ldots '\,t_n) = \mathsf{f}\,'\,\mathcal{I}(t_1)\,' \ldots '\,\mathcal{I}(t_n)$ *for all* $\mathsf{f} \in \mathcal{F}$ *where* $\mathrm{arity}(\mathsf{f}) = n$
- $\mathcal{I}((\mathsf{f}\,'\,t_1\,' \ldots '\,t_n)^\sharp) = (\mathsf{f}\,'\,\mathcal{I}(t_1)\,' \ldots '\,\mathcal{I}(t_n))^\sharp$ *for all* $\mathsf{f} \in \mathcal{F}$ *where* $\mathrm{arity}(\mathsf{f}) = n$
- $\mathcal{I}(t) = \bot_t$, *for all other* $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$

Moreover, for any substitution σ, $\mathcal{I}(\sigma)$ is the substitution with $\mathcal{I}(\sigma)(x) = \mathcal{I}(\sigma(x))$.

Lemma 25 (Properties of $\mathcal{I}$) *Let $(\mathcal{P}, \mathcal{R}, e)$ be a proper DP problem over $\mathcal{F}$, let t and s be from $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$ and let $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V})$.*

[6] This does not hold for arbitrary terms t as can be seen from $t = \mathsf{minus}\,'\,\mathsf{s}\,'\,\mathsf{0}\,'\,x$. While $t \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{s}\,'\,x$ using the rule (10), the term $\mathcal{I}(t) = \bot_{\mathsf{minus}\,'\,\mathsf{s}\,'\,\mathsf{0}\,'\,x}$ is a normal form. Thus, the transformation $\mathcal{I}$ cannot be used in the termination case where we have to consider arbitrary (possibly non-normal) substitutions σ. So we really need two different transformations $\mathcal{Z}$ and $\mathcal{I}$ for termination and innermost termination, respectively.

(a) $\mathcal{I}(t)$ *is proper*
(b) If t *is proper then* $\mathcal{I}(t\sigma) = t\mathcal{I}(\sigma)$
(c) If t *is in normal form then* $\mathcal{I}(t)$ *is in normal form w.r.t.* $\mathcal{R}$
(d) If σ *is normal and* t *is proper then* $t\sigma \xrightarrow{\mathsf{i}}{}^m_{\mathcal{R}} s$ *implies* $\mathcal{I}(t\sigma) \xrightarrow{\mathsf{i}}{}^m_{\mathcal{R}} \mathcal{I}(s)$

Proof. Again, the claims (a) and (b) can be proved by straightforward structural induction on t. We prove (c) by induction on t. If $\mathcal{I}(t)$ is a variable then the claim is trivial. Otherwise, let $t = \mathsf{f}\,'\,t_1\,'\,\dots\,'\,t_n$ (the case $t = (\mathsf{f}\,'\,t_1\,'\,\dots\,'\,t_n)^\sharp$ is analogous). We say that a term is *normal* if it is in normal form w.r.t. $\mathcal{R}$. Hence, all t_i are normal and by induction all $\mathcal{I}(t_i)$ are normal, too. Thus, $\mathcal{I}(t) = \mathsf{f}\,'\,\mathcal{I}(t_1)\,'\,\dots\,'\,\mathcal{I}(t_n)$ can only be reduced at the root, i.e., $\mathcal{I}(t) = l\delta$ for some $l \to r \in \mathcal{R}$. By induction on l, we now show that all $\delta(x)$ are proper and that $t = l\mathcal{I}^{-1}(\delta)$. ($\mathcal{I}$ is injective and it is surjective on the proper terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}^\sharp, \mathcal{V} \cup \mathcal{V}')$.) This contradicts the prerequisite that t is normal.

If $l = x$ then $\delta(x) = \mathcal{I}(t)$ is proper by (a) and $t = \mathcal{I}^{-1}(\mathcal{I}(t)) = \mathcal{I}^{-1}(\delta(x)) = l\mathcal{I}^{-1}(\delta)$. If $l = \mathsf{f}\,'\,l_1\,'\,\dots\,'\,l_n$ we have $\mathcal{I}(t) = \mathsf{f}\,'\,l_1\delta\,'\,\dots\,'\,l_n\delta$. Thus, t must be of the form $\mathsf{f}\,'\,t_1\,'\,\dots\,'\,t_n$ and $\mathcal{I}(t_i) = l_i\delta$. By the induction hypothesis, all $\delta(x)$ are proper and $t_i = l_i\mathcal{I}^{-1}(\delta)$ which implies $t = \mathsf{f}\,'\,t_1\,'\,\dots\,'\,t_n = \mathsf{f}\,'\,l_1\mathcal{I}^{-1}(\delta)\,'\,\dots\,'\,l_n\mathcal{I}^{-1}(\delta) = (\mathsf{f}\,'\,l_1\,'\,\dots\,'\,l_n)\mathcal{I}^{-1}(\delta) = l\mathcal{I}^{-1}(\delta)$.

For (d), we prove that $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} s$ implies both $\mathcal{I}(t\sigma) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathcal{I}(s)$ and $s = u\delta$ for a proper term u and a normal substitution δ. Then (d) follows by induction.

As σ is normal, t is no variable. We only regard the case $t = \mathsf{f}\,'\,t_1\,'\,\dots\,'\,t_n$ since the case $t = (\mathsf{f}\,'\,t_1\,'\,\dots\,'\,t_n)^\sharp$ is analogous. If s is obtained by reducing $t_i\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} s_i$ then by the induction hypothesis we conclude $\mathcal{I}(t_i\sigma) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathcal{I}(s_i)$ and $s_i = u_i\delta$ for some normal substitution δ and proper term u_i. We may assume that u_i is variable disjoint from t_j for all $j \neq i$. Then we can extend δ to behave like σ on the variables of t_j for all $j \neq i$. Hence, for $u = \mathsf{f}\,'\,t_1\,'\,\dots\,'\,u_i\,'\,\dots\,'\,t_n$ we obtain $u\delta = s$. Moreover,

$$\begin{array}{rcl}
\mathcal{I}(t\sigma) & = & \mathsf{f}\,'\mathcal{I}(t_1\sigma)\,'\,\dots\,'\mathcal{I}(t_i\sigma)\,'\,\dots\,'\mathcal{I}(t_n\sigma) \\
& \xrightarrow{\mathsf{i}}_{\mathcal{R}} & \mathsf{f}\,'\mathcal{I}(t_1\sigma)\,'\,\dots\,'\mathcal{I}(u_i\delta)\,'\,\dots\mathcal{I}(t_n\sigma) \\
& = & \mathsf{f}\,'\mathcal{I}(t_1\delta)\,'\,\dots\,'\mathcal{I}(u_i\delta)\,'\,\dots\,'\mathcal{I}(t_n\delta) \\
& = & \mathcal{I}(\mathsf{f}\,'\,t_1\delta\,'\,\dots\,'\,u_i\delta\,'\,\dots\,'\,t_n\delta) \\
& = & \mathcal{I}(u\delta) \\
& = & \mathcal{I}(s).
\end{array}$$

Otherwise, the reduction is on the root position, i.e., $t\sigma = l\tau \xrightarrow{\mathsf{i}}_{\mathcal{R}} r\tau = s$ where $l = \mathsf{f}\,'\,l_1\,'\,\dots\,'\,l_n$. We choose $u = r$ and $\delta = \tau$ to obtain $s = u\delta$. Then $\mathcal{I}(t\sigma) = \mathcal{I}(l\tau) = l\mathcal{I}(\tau) \xrightarrow{\mathsf{i}}_{\mathcal{R}} r\mathcal{I}(\tau) = \mathcal{I}(r\tau) = \mathcal{I}(s)$ by (b). This is indeed an innermost step since all $l_i\tau$ are normal and by (c) all $\mathcal{I}(l_i\tau)$ are normal, too. □

Now we can formulate the desired processor which transforms proper applicative DP problems into functional form.

Theorem 26 (DP Processor for Transformation in Functional Form) *For any DP problem* $(\mathcal{P}, \mathcal{R}, e)$*, let Proc return* $\{(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), e)\}$ *if* $(\mathcal{P}, \mathcal{R}, e)$ *is proper and* $\{(\mathcal{P}, \mathcal{R}, e)\}$ *otherwise. Then Proc is sound and complete.*

Proof. We first prove soundness in the termination case $e = \mathbf{t}$. To this end, we show that every infinite $(\mathcal{P}, \mathcal{R}, e)$-chain $s_1 \to t_1, s_2 \to t_2, \dots$ corresponds to an infinite $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), e)$-chain. There is some σ with $t_i\sigma \to^*_{\mathcal{R}} s_{i+1}\sigma$ for all i. Hence,

$t_i\mathcal{Z}(\sigma) = \mathcal{Z}(t_i\sigma) \to_{\mathcal{R}}^* \mathcal{Z}(s_{i+1}\sigma) = s_{i+1}\mathcal{Z}(\sigma)$ by Lemma 23 (b) and (c), where t_i, s_{i+1}, and $\mathcal{Z}(\sigma)$ are proper by Lemma 23 (a). Thus, using Lemma 21 (a) and (b) we obtain $\mathcal{A}(t_i)\mathcal{A}(\mathcal{Z}(\sigma)) = \mathcal{A}(t_i\mathcal{Z}(\sigma)) \to_{\mathcal{A}(\mathcal{R})}^* \mathcal{A}(s_{i+1}\mathcal{Z}(\sigma)) = \mathcal{A}(s_{i+1})\mathcal{A}(\mathcal{Z}(\sigma))$. Hence, $\mathcal{A}(s_1) \to \mathcal{A}(t_1)$, $\mathcal{A}(s_2) \to \mathcal{A}(t_2)$, $\ldots$ is an $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), \mathbf{t})$ chain.

For soundness in the innermost case we know that all $s_i\sigma$ are normal and that $t_i\sigma \xrightarrow{\mathrm{i}}_{\mathcal{R}}{}^* s_{i+1}\sigma$. Hence, $t_i\mathcal{I}(\sigma) = \mathcal{I}(t_i\sigma) \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* \mathcal{I}(s_{i+1}\sigma) = s_{i+1}\mathcal{I}(\sigma)$ by Lemma 25 (b) and (d), where t_i, s_{i+1}, and $\mathcal{I}(\sigma)$ are proper by Lemma 25 (a). As in the termination case one can prove that $\mathcal{A}(s_1) \to \mathcal{A}(t_1)$, $\mathcal{A}(s_2) \to \mathcal{A}(t_2)$, $\ldots$ is an $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), \mathbf{i})$-chain by using the substitution $\mathcal{A}(\mathcal{I}(\sigma))$ (since $\mathcal{A}(s_i)\mathcal{A}(\mathcal{I}(\sigma))$ is normal by Lemma 25 (c) and Lemma 21 (b)).

For completeness let $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), e)$ be infinite. Thus, $\mathcal{A}(\mathcal{R})$ is not (innermost) terminating or there is an infinite $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), e)$-chain. In the former case we obtain that $\mathcal{R}$ is not (innermost) terminating either by Lemma 21 (b) and (c). Otherwise, let $\mathcal{A}(s_1) \to \mathcal{A}(t_1)$, $\mathcal{A}(s_2) \to \mathcal{A}(t_2)$, $\ldots$ be an infinite $(\mathcal{A}(\mathcal{P}), \mathcal{A}(\mathcal{R}), e)$-chain. If $e = \mathbf{t}$ then there is some substitution δ such that $\mathcal{A}(t_i)\delta \to_{\mathcal{A}(\mathcal{R})}^* \mathcal{A}(s_{i+1})\delta$. By Lemma 21 (a) and (b) we obtain $t_i\mathcal{A}^{-1}(\delta) = \mathcal{A}^{-1}(\mathcal{A}(t_i)\delta) \to_{\mathcal{R}}^* \mathcal{A}^{-1}(\mathcal{A}(s_{i+1})\delta) = s_{i+1}\mathcal{A}^{-1}(\delta)$ which shows that there is an infinite $(\mathcal{P}, \mathcal{R}, \mathbf{t})$-chain. Otherwise, if $e = \mathbf{i}$ we know that $\mathcal{A}(t_i)\delta \xrightarrow{\mathrm{i}}_{\mathcal{A}(\mathcal{R})}^* \mathcal{A}(s_{i+1})\delta$ and all $\mathcal{A}(s_i)\delta$ are in normal form w.r.t. $\mathcal{A}(\mathcal{R})$. By Lemma 21 (a) and (c) we obtain $t_i\mathcal{A}^{-1}(\delta) \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* s_{i+1}\mathcal{A}^{-1}(\delta)$ as in the termination case. Moreover, $s_i\mathcal{A}^{-1}(\delta)$ is in normal form w.r.t. $\mathcal{R}$, because otherwise $\mathcal{A}(s_i\mathcal{A}^{-1}(\delta)) = \mathcal{A}(s_i)\delta$ would be reducible w.r.t. $\mathcal{A}(\mathcal{R})$ by Lemma 21 (b). Thus, there is an infinite $(\mathcal{P}, \mathcal{R}, \mathbf{i})$-chain. $\square$

With the new processor of Thm. 26 and our new improved estimation of dependency graphs (Def. 12), it does not matter anymore for the termination proof whether first-order functions are represented in applicative or in ordinary functional form. The reason is that if they are represented by applicative rules, then all dependency pairs with non-proper right-hand sides are not in SCCs of the improved estimated dependency graph. Hence, after applying the dependency graph processor of Thm. 10, all remaining DP problems are proper and can be transformed into functional form by Thm. 26.

As an alternative to the processor of Thm. 26, one can also couple the transformation $\mathcal{A}$ with the reduction pair processor from Thm. 14. Then a DP problem $(\mathcal{P}, \mathcal{R}, e)$ is transformed into $\{(\mathcal{P} \setminus \{s \to t \mid \mathcal{A}(s) \succ \mathcal{A}(t)\}, \mathcal{R}, e)\}$ if $(\mathcal{P}, \mathcal{R}, e)$ is proper, if $\mathcal{A}(\mathcal{P})_{\succ} \cup \mathcal{A}(\mathcal{P})_{\succsim} = \mathcal{A}(\mathcal{P})$, and if $\mathcal{A}(\mathcal{R})_{\succsim} = \mathcal{A}(\mathcal{R})$ holds for some reduction pair $(\succsim, \succ)$. An advantage of this alternative processor is that it can be combined with our results from [29] on applying usable rules for termination instead of innermost termination proofs, cf. Sect. 3.2.

3.4 Comparison with Related Work

Most approaches for higher-order functions in term rewriting use *higher-order TRSs*. However, the main automated termination techniques for such TRSs are *simplification orders* (e.g., [19]) which fail on functions like div in Ex. 18.

Exceptions are the *monotonic higher-order semantic path order* [8] and the existing variants of dependency pairs for higher-order TRSs. However, these variants require considerable restrictions (e.g., on the TRSs [28] or on the orders that may be used [3, 23, 27].) So in contrast to our results, they are less powerful than the original dependency pair technique when applied to first-order functions.

Termination techniques for higher-order TRSs often handle a richer language than our results. But an efficient automation of these approaches is usually not straightforward (there are hardly any implementations of these techniques available). In contrast, only minor modifications are needed to integrate our results into termination provers for ordinary first-order TRSs that use dependency pairs.

Other approaches represent higher-order functions by first-order TRSs [1, 2, 18, 24, 30], similar to us. However, they mostly use *monomorphic* types without type variables (this restriction is also imposed in some approaches for higher-order TRSs [8]). Then terms like "$\mathsf{map}\,'\,\mathsf{minus}\,'\,xs$" and "$\mathsf{map}\,'\,(\mathsf{minus}\,'\,x)\,'\,xs$" cannot both be well typed, but one needs different map-symbols for arguments of different types. In contrast, our approach uses untyped term rewriting. Hence, it can be applied for termination analysis of polymorphic or untyped functional languages. Moreover, [24] and [30] only consider extensions of the lexicographic path order, whereas we can also handle non-simply terminating TRSs like Ex. 18.

4 A DP Processor for Proving Non-Termination

Almost all techniques for automated termination analysis try to *prove termination* and there are hardly any methods to *prove non-termination*. But detecting non-termination automatically would be very helpful when debugging programs.

We show that the DP framework is particularly suitable for combining both termination and *non*-termination analysis. We introduce a DP processor which tries to detect infinite DP problems in order to answer "no". The processor also handles higher-order functions if they are represented by first-order TRSs. Then, if all previous processors were complete, we can conclude non-termination of the original TRS. An important advantage of the DP framework is that it can couple the search for a proof and a disproof of termination: Processors which try to prove termination are also helpful for the non-termination proof because they transform the initial DP problem into sub-problems, where most of them can easily be proved finite. So they detect those sub-problems which could cause non-termination. Therefore, the non-termination processors should only operate on these sub-problems and thus, they only have to regard a subset of the rules when searching for non-termination. On the other hand, processors that try to disprove termination are also helpful for the termination proof, even if some of the previous processors were incomplete. The reason is that there are many indeterminisms in a termination proof attempt, since usually many DP processors can be applied to a DP problem (possibly in several different ways). Thus, if one can find out that a DP problem is infinite, one knows that one has reached a "dead end" and should backtrack.

To prove non-termination within the DP framework, in Sect. 4.1 we introduce *looping* DP problems and in Sect. 4.2 we show how to detect such DP problems automatically. Finally, Sect. 4.3 is a comparison with related work.

4.1 A DP Processor Based on Looping DP Problems

An obvious approach to find infinite reductions is to search for a term s which evaluates to a term $C[s\mu]$ containing an instance of s. A TRS with such reductions is called *looping*. Clearly, a naive search for looping terms is very costly.

In contrast to "looping TRSs", when adapting the concept of *loopingness* to DP problems, we only have to consider terms s occurring in dependency pairs and we do not have to regard any contexts C. The reason is that such contexts are already removed by the construction of dependency pairs. In Thm. 29, we will show that in this way one can indeed detect all looping TRSs.

Definition 27 (Looping DP Problems) *A DP problem $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is* looping *iff there is a $(\mathcal{P}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$ with $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ for all i such that $s_1\sigma$ matches $s_k\sigma$ for some $k > 1$ (i.e., $s_1\sigma\mu = s_k\sigma$ for a substitution μ).*

To prove that loopingness of TRSs corresponds to loopingness of DP problems, we first need the following auxiliary lemma about the form of looping reductions. Let $\trianglerighteq$ denote the subterm relation and $\triangleright$ is the proper subterm relation. Any reduction of the form $s = s_0 \trianglerighteq t_0 \to_{\mathcal{R}} s_1 \trianglerighteq t_1 \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} s_{m-1} \trianglerighteq t_{m-1} \to_{\mathcal{R}} s_m \trianglerighteq t_m = s\mu$ with $m > 0$ is called a *cyclic reduction of $\mathcal{R}$ of length m*. Obviously, any looping TRS has such a cyclic reduction, since loopingness implies that there is a term s and a substitution μ with $s \to_{\mathcal{R}}^+ C[s\mu] \trianglerighteq s\mu$. In the following, $\to_{\mathcal{R},\varepsilon}$ denotes $\mathcal{R}$-reductions at the root position and $\to_{\mathcal{R},>\varepsilon}$ denotes $\mathcal{R}$-reductions below the root.

Lemma 28 (Form of Looping Reductions) *Let $\mathcal{R}$ be a looping TRS and let m be the length of the shortest cyclic reduction of $\mathcal{R}$. Then there is some term s and some substitution μ such that $s\ ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon})^m\ s\mu$, i.e., steps with $\triangleright$ can only take place after $\mathcal{R}$-reductions at the root. Moreover, the reduction contains at least one root reduction step (i.e., one step with $\to_{\mathcal{R},\varepsilon}$) and every term in the reduction has a defined root.*

Proof. Let s be a minimal term (w.r.t. $\triangleright$) that has a cyclic reduction of length m. We show that in the cyclic reduction of s we can always exchange the sequence $\to_{\mathcal{R},>\varepsilon} \circ \triangleright_d$ by $\triangleright_d \circ \to_{\mathcal{R}}$, where $\triangleright_d$ is the *direct* subterm relation. This proves that we have $s \trianglerighteq s'\ ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon})^m\ s\mu$ for some term s'. So let $u \to_{\mathcal{R},>\varepsilon} v \triangleright_d w$ be a reduction occurring in the cyclic reduction of s. Then we have $u = \mathsf{f}(u_1, \ldots, u_n)$, $v = \mathsf{f}(v_1, \ldots, v_n)$, $w = v_j$, $u_i \to_{\mathcal{R}} v_i$, and $u_k = v_k$ for all $k \neq i$. If $j = i$, then $u \triangleright_d u_i \to_{\mathcal{R}} v_i = v_j = w$. Otherwise, $u \triangleright_d u_j = v_j = w$ which is a contradiction to the minimality of m.

Now note that $s = s'$. The reason is that $s \triangleright s'$ would imply that there is a cyclic reduction $s'\ ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon})^m\ s\mu \triangleright s'\mu$ of length m for the term s'. This is a contradiction to the minimality of s. Hence, we indeed obtain $s\ ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon})^m\ s\mu$.

Next we show that there must be at least one reduction step at the root. Otherwise, we would obtain $s \to_{\mathcal{R},>\varepsilon}^m s\mu$. Hence, $s = \mathsf{f}(s_1, \ldots, s_n) \to_{\mathcal{R},>\varepsilon}^m \mathsf{f}(s_1\mu, \ldots, s_n\mu)$. For all i, we have $s_i \to_{\mathcal{R},>\varepsilon}^{m_i} s_i\mu$ for some m_i with $m_1 + \ldots + m_n = m$. Since $m > 0$, there must be at least one $m_i > 0$ which shows that $s_i \to_{\mathcal{R},>\varepsilon}^{m_i} s_i\mu$ is a cyclic reduction with a length of at most m. Note that $m_i = m$ because of minimality of m. But since $s \triangleright s_i$, this is a contradiction to the minimality of s.

Finally, we show that all terms in the reduction have a defined root symbol. First note that the root of s must be defined, since s starts a reduction w.r.t. $((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon})^*$ that contains a root step with $\to_{\mathcal{R},\varepsilon}$. Moreover, for any term s' with $s'\ ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon})^*\ s\mu$, the root of s' must also be defined.

The reason is that otherwise, s' can only be reduced by $\to_{\mathcal{R},>\varepsilon}$-steps, but then one cannot obtain the term $s\mu$ which has a defined root. $\square$

Now we can show the desired result that looping TRSs correspond to looping DP problems.

Theorem 29 (Loopingness of TRSs and DP problems) *A TRS $\mathcal{R}$ is looping iff the DP problem $(DP(\mathcal{R}), \mathcal{R}, \mathbf{t})$ is looping.*

Proof. We first prove that loopingness of $(DP(\mathcal{R}), \mathcal{R}, \mathbf{t})$ implies that $\mathcal{R}$ is looping. Let $s_1 \to t_1, s_2 \to t_2, \dots$ be a looping $(\mathcal{P}, \mathcal{R})$-chain, i.e., $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ for all i and $s_1\sigma\mu = s_k\sigma$ for some $k > 1$. For any term t whose root symbol is a tuple symbol, let $t^\flat$ denote the term which results from replacing the tuple symbol by the corresponding defined symbol (i.e., $\mathsf{F}(t_1, \dots, t_n)^\flat = \mathsf{f}(t_1, \dots, t_n)$). As $\mathcal{R}$ does not contain tuple symbols and as all s_i and t_i have a tuple symbol on the root position, we obtain $t_i^\flat\sigma \to_{\mathcal{R}}^* s_{i+1}^\flat\sigma$ for all i. Moreover, for each dependency pair $s_i \to t_i$, there is a rule $s_i^\flat \to C_i[t_i^\flat] \in \mathcal{R}$. Hence, $s_1^\flat\sigma \to_{\mathcal{R}} C_1\sigma[t_1^\flat\sigma] \to_{\mathcal{R}}^* C_1\sigma[s_2^\flat\sigma] \to_{\mathcal{R}} C_1\sigma[C_2\sigma[t_2^\flat\sigma]] \to_{\mathcal{R}}^* \dots \to_{\mathcal{R}} C_1\sigma[\dots C_{k-1}\sigma[t_{k-1}^\flat\sigma]\dots] \to_{\mathcal{R}}^* C_1\sigma[\dots C_{k-1}\sigma[s_k^\flat\sigma]\dots] = C_1\sigma[\dots C_{k-1}\sigma[s_1^\flat\sigma\mu]\dots]$. Hence, $\mathcal{R}$ is looping.

For the other direction, let $\mathcal{R}$ be looping. By Lemma 28 there is a term s such that $s = s_0 \; ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon}) \; s_1 \; ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon}) \dots ((\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \cup \to_{\mathcal{R},>\varepsilon}) \; s_m = s\mu$ where m is the length of the shortest cyclic reduction. Here, all s_i have a defined root and we have used at least one $(\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq)$-step. Obviously, whenever we have $s_i \to_{\mathcal{R},>\varepsilon} s_{i+1}$, then we also obtain $s_i^\sharp \to_{\mathcal{R}} s_{i+1}^\sharp$. Next we show that $s_i \; (\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \; s_{i+1}$ implies $s_i^\sharp = u\sigma$ and $s_{i+1}^\sharp = v\sigma$ for some dependency pair $u \to v \in DP(\mathcal{R})$. Thus, let $s_i = l\sigma \to_{\mathcal{R}} r\sigma \trianglerighteq r\sigma|_p = s_{i+1}$ for some rule $l \to r \in \mathcal{R}$ and some position p in $r\sigma$. First note that due to the minimality of m, the term s_{i+1} cannot be entirely in σ, i.e., p is a position of r and $r|_p$ is not a variable. The reason is that otherwise, we would have $s_i \trianglerighteq s_{i+1}$ and thus, we could obtain a cyclic reduction of shorter length. Thus, $s_{i+1} = r|_p\sigma$ where $r|_p \notin \mathcal{V}$. As the root of s_{i+1} is defined, $r|_p$ has a defined root as well and hence, $l^\sharp \to r|_p^\sharp \in DP(\mathcal{R})$. Thus, $s_i^\sharp = l^\sharp\sigma$ and $s_{i+1}^\sharp = r|_p^\sharp\sigma$.

So if $s = s_0 \to_{\mathcal{R},>\varepsilon}^* s_{i_1} \; (\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \; s_{i_1+1} \to_{\mathcal{R},>\varepsilon}^* s_{i_2} \; (\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \dots \to_{\mathcal{R},>\varepsilon}^* s_{i_k} \; (\to_{\mathcal{R},\varepsilon} \circ \trianglerighteq) \; s_{i_k+1} \to_{\mathcal{R},>\varepsilon}^* s_{i_{k+1}} = s_m = s\mu = s_0\mu$, then by the above results we know that $s^\sharp = s_0^\sharp \to_{\mathcal{R}}^* s_{i_1}^\sharp = u_1\sigma \to_{DP(\mathcal{R})} v_1\sigma = s_{i_1+1}^\sharp \to_{\mathcal{R}}^* s_{i_2}^\sharp = u_2\sigma \to_{DP(\mathcal{R})} \dots \to_{\mathcal{R}}^* s_{i_k}^\sharp = u_k\sigma \to_{DP(\mathcal{R})} v_k\sigma = s_{i_k+1}^\sharp \to_{\mathcal{R}}^* s_{i_{k+1}}^\sharp = s^\sharp\mu = s_0^\sharp\mu \to_{\mathcal{R}}^* s_{i_1}^\sharp\mu = u_1\sigma\mu$ for some dependency pairs $u_i \to v_i \in DP(\mathcal{R})$. As there was at least one root step in the original reduction, we know that $k \geq 1$. Thus, the $(\mathcal{P}, \mathcal{R})$-chain $u_1 \to v_1, \dots, u_k \to v_k, u_1 \to v_1$ shows that $(DP(\mathcal{R}), \mathcal{R}, \mathbf{t})$ is looping. $\square$

Example 30 *Consider Toyama's example $\mathcal{R} = \{\mathsf{f}(\mathsf{0}, \mathsf{1}, x) \to \mathsf{f}(x, x, x), \mathsf{g}(y, z) \to y, \mathsf{g}(y, z) \to z\}$ and $\mathcal{P} = DP(\mathcal{R}) = \{\mathsf{F}(\mathsf{0}, \mathsf{1}, x) \to \mathsf{F}(x, x, x)\}$. We have the $(\mathcal{P}, \mathcal{R})$-chain $\mathsf{F}(\mathsf{0}, \mathsf{1}, x_1) \to \mathsf{F}(x_1, x_1, x_1)$, $\mathsf{F}(\mathsf{0}, \mathsf{1}, x_2) \to \mathsf{F}(x_2, x_2, x_2)$, since $\mathsf{F}(x_1, x_1, x_1)\sigma \to_{\mathcal{R}}^* \mathsf{F}(\mathsf{0}, \mathsf{1}, x_2)\sigma$ for $\sigma(x_1) = \sigma(x_2) = \mathsf{g}(\mathsf{0}, \mathsf{1})$. As the term $\mathsf{F}(\mathsf{0}, \mathsf{1}, x_1)\sigma$ matches $\mathsf{F}(\mathsf{0}, \mathsf{1}, x_2)\sigma$ (they are even identical), the DP problem $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is looping.*

Our goal is to detect looping DP problems. In the termination case, every looping DP problem is infinite and hence, if all preceding DP processors were complete, then termination is disproved. However, the definition of "looping" from Def. 27 cannot be used for innermost termination: in Ex. 30, $(DP(\mathcal{R}), \mathcal{R}, \mathbf{t})$ is looping, but $(DP(\mathcal{R}), \mathcal{R}, \mathbf{i})$ is finite and $\mathcal{R}$ is innermost terminating.[7]

Nevertheless, for *non-overlapping* DP problems, $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is infinite whenever $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is infinite. So here loopingness of $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ indeed implies that $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is infinite. We call $(\mathcal{P}, \mathcal{R}, e)$ *non-overlapping* if $\mathcal{R}$ is non-overlapping and no left-hand side of $\mathcal{R}$ unifies with a non-variable subterm of a left-hand side of $\mathcal{P}$.

Lemma 31 (Looping and Infinite DP Problems)

(a) If $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is looping, then $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is infinite.
(b) If $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is infinite and non-overlapping, then $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is infinite.

Proof. For (a), let $s_1 \to t_1, \ldots, s_k \to t_k$ be a chain where $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ for all i and $s_1\sigma\mu = s_k\sigma$. Hence, $s_1 \to t_1, \ldots, s_{k-1} \to t_{k-1}, s_1 \to t_1, \ldots, s_{k-1} \to t_{k-1}, \ldots$ is an infinite $(\mathcal{P}, \mathcal{R})$-chain. To see this, one can use a substitution which behaves like σ on the variables of the first $k-1$ dependency pairs, like $\sigma\mu$ on the variables of the next $k-1$ pairs, like $\sigma\mu^2$ on the next $k-1$ pairs, etc. Then obviously, the instantiated right-hand side of each pair in the chain reduces to the instantiated left-hand side of the next pair.

For (b), let $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ be infinite. If $\mathcal{R}$ is not terminating then $\mathcal{R}$ is also not innermost terminating since $\mathcal{R}$ is non-overlapping, and thus, $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is infinite. Otherwise, there is some infinite $(\mathcal{P}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2 \ldots$ with $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$. The chain is *minimal* (i.e., all $t_i\sigma$ are terminating), since $\mathcal{R}$ is terminating. Hence, by [16, Thm. 32] there is also an infinite innermost $(\mathcal{P}, \mathcal{R})$-chain. Thus, $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is infinite. □

Now we can define the DP processor for proving non-termination.

Theorem 32 (Non-Termination Processor) *The following DP processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{R}, e)$, Proc returns*

- *"no", if $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is looping and ($e = \mathbf{t}$ or $(\mathcal{P}, \mathcal{R}, e)$ is non-overlapping)*
- *$\{(\mathcal{P}, \mathcal{R}, e)\}$, otherwise*

Proof. The theorem is an immediate consequence of Lemma 31. □

4.2 Detecting Looping DP Problems

Our criteria to detect looping DP problems automatically use *narrowing*.

Definition 33 (Narrowing) *Let $\mathcal{R}$ be a TRS which may also have rules $l \to r$ where $\mathcal{V}(r) \not\subseteq \mathcal{V}(l)$ or $l \in \mathcal{V}$. A term t narrows to s, denoted $t \rightsquigarrow_{\mathcal{R},\delta,p} s$, iff there is a substitution δ, a (variable-renamed) rule $l \to r \in \mathcal{R}$ and a non-variable position p of t such that $\delta = mgu(t|_p, l)$ and $s = t[r]_p\delta$. Let $\rightsquigarrow_{\mathcal{R},\delta}$ be the relation which*

[7] One can adapt "loopingness" to the innermost case: $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is *looping* iff there is an *innermost* $(\mathcal{P}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$ such that $t_i\sigma\mu^n \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* s_{i+1}\sigma\mu^n$, $s_1\sigma\mu = s_k\sigma$, and $s_i\sigma\mu^n$ is in normal form for all i *and all* $n \geq 0$. Then loopingness implies that the DP problem is infinite, but since one has to examine *infinitely* many instantiations $s_i\sigma\mu^n$ and $t_i\sigma\mu^n$, in general loopingness is hard to check. Nevertheless, for special TRSs (e.g., overlay systems), one can also formulate sufficient conditions for loopingness in the innermost case which are amenable to automation.

permits narrowing steps on all positions p. Let $\rightsquigarrow_{(\mathcal{P},\mathcal{R}),\delta}$ *denote* $\rightsquigarrow_{\mathcal{P},\delta,\varepsilon} \cup \rightsquigarrow_{\mathcal{R},\delta}$, *where* ε *is the root position. Moreover, let* $\rightsquigarrow^*_{(\mathcal{P},\mathcal{R}),\delta}$ *be the smallest relation which contains* $\rightsquigarrow_{(\mathcal{P},\mathcal{R}),\delta_1} \circ \ldots \circ \rightsquigarrow_{(\mathcal{P},\mathcal{R}),\delta_n}$ *for all* $n \geq 0$ *and all substitutions where* $\delta = \delta_1 \ldots \delta_n$.

Example 34 *Let* $\mathcal{R} = \{\mathsf{f}(x,y,z) \to \mathsf{g}(x,y,z),\ \mathsf{g}(\mathsf{s}(x),y,z) \to \mathsf{f}(z,\mathsf{s}(y),z)\}$ *and let* $\mathcal{P} = DP(\mathcal{R}) = \{\mathsf{F}(x,y,z) \to \mathsf{G}(x,y,z),\ \mathsf{G}(\mathsf{s}(x),y,z) \to \mathsf{F}(z,\mathsf{s}(y),z)\}$. *The term* $\mathsf{G}(x,y,z)$ *can only be narrowed by the rule* $\mathsf{G}(\mathsf{s}(x'),y',z') \to \mathsf{F}(z',\mathsf{s}(y'),z')$ *on the root position and hence, we obtain* $\mathsf{G}(x,y,z) \rightsquigarrow_{\mathcal{P},[x/\mathsf{s}(x'),\, y'/y,\, z'/z],\varepsilon} \mathsf{F}(z,\mathsf{s}(y),z)$.

To find loops, we narrow the right-hand side t of a dependency pair $s \to t$ until one reaches a term s' such that $s\delta$ *semi-unifies* with s' (i.e., $s\delta\mu_1\mu_2 = s'\mu_1$ for some substitutions μ_1 and μ_2). Here, δ is the substitution used for narrowing. Then we indeed have a loop as in Def. 27 by defining $\sigma = \delta\mu_1$ and $\mu = \mu_2$. Semi-unification encompasses both matching and unification and algorithms for semi-unification can for example be found in [20, 25].

Theorem 35 (Loop Detection by Forward Narrowing) *Let* $(\mathcal{P},\mathcal{R},e)$ *be a DP problem. If there is an* $s \to t \in \mathcal{P}$ *such that* $t \rightsquigarrow^*_{(\mathcal{P},\mathcal{R}),\delta} s'$ *and* $s\delta$ *semi-unifies with* s', *then* $(\mathcal{P},\mathcal{R},\mathbf{t})$ *is looping.*

Proof. We have $s\delta\mu_1\mu_2 = s'\mu_1$ for some substitutions μ_1 and μ_2. Let $\sigma = \delta\mu_1$. We know that $s\sigma \to_{\mathcal{P}} t\sigma = t\delta\mu_1 \to^*_{\mathcal{P}\cup\mathcal{R}} s'\mu_1 = s\sigma\mu_2$ where $\mathcal{P}$-rules are only used on the root position. Thus, $(\mathcal{P},\mathcal{R},\mathbf{t})$ is looping. □

Example 36 *We continue with Ex. 34. We had* $\mathsf{G}(x,y,z) \rightsquigarrow_{(\mathcal{P},\mathcal{R}),\delta} \mathsf{F}(z,\mathsf{s}(y),z)$ *where* $\delta = [x/\mathsf{s}(x'),\ y'/y,\ z'/z]$. *Applying* δ *to the left-hand side* $s = \mathsf{F}(x,y,z)$ *of the first dependency pair yields* $\mathsf{F}(\mathsf{s}(x'),y,z)$. *Now* $\mathsf{F}(\mathsf{s}(x'),y,z)$ *semi-unifies with* $\mathsf{F}(z,\mathsf{s}(y),z)$, *since* $\mathsf{F}(\mathsf{s}(x'),y,z)\mu_1\mu_2 = \mathsf{F}(z,\mathsf{s}(y),z)\mu_1$ *for the substitutions* $\mu_1 = [z/\mathsf{s}(x')]$ *and* $\mu_2 = [y/\mathsf{s}(y)]$. *(However, the first term does not match or unify with the second.) Thus,* $(\mathcal{P},\mathcal{R},\mathbf{t})$ *is looping and* $\mathcal{R}$ *does not terminate.*

However, while the DP problem of Toyama's example (Ex. 30) is looping, this is not detected by Thm. 35. The reason is that the right-hand side $\mathsf{F}(x,x,x)$ of the only dependency pair cannot be narrowed. Therefore, we now introduce a variant of the above criterion which narrows with the reversed TRSs $\mathcal{P}^{-1}$ and $\mathcal{R}^{-1}$.

Theorem 37 (Loop Detection by Backward Narrowing) *Let* $(\mathcal{P},\mathcal{R},e)$ *be a DP problem. If there is an* $s \to t \in \mathcal{P}$ *such that* $s \rightsquigarrow^*_{(\mathcal{P}^{-1},\mathcal{R}^{-1}),\delta} t'$ *and* t' *semi-unifies with* $t\delta$, *then* $(\mathcal{P},\mathcal{R},\mathbf{t})$ *is looping.*

Proof. We have $t'\mu_1\mu_2 = t\delta\mu_1$ for some substitutions μ_1 and μ_2. Let $\sigma = \delta\mu_1$. From $s \rightsquigarrow^*_{(\mathcal{P}^{-1},\mathcal{R}^{-1}),\delta} t'$ we obtain $s\delta \to^*_{\mathcal{P}^{-1}\cup\mathcal{R}^{-1}} t'$ where all $\mathcal{P}^{-1}$-rules are only used on the root position. Thus, $t' \to^*_{\mathcal{P}\cup\mathcal{R}} s\delta$ which implies $s\sigma \to_{\mathcal{P}} t\sigma = t\delta\mu_1 = t'\mu_1\mu_2 \to^*_{\mathcal{P}\cup\mathcal{R}} s\delta\mu_1\mu_2 = s\sigma\mu_2$ where $\mathcal{P}$-rules are only used on the root position. This shows that $(\mathcal{P},\mathcal{R},\mathbf{t})$ is looping. □

Example 38 *To detect that Toyama's example (Ex. 30) is looping, we start with the left-hand side* $s = \mathsf{F}(\mathsf{0},\mathsf{1},x)$ *and narrow* $\mathsf{0}$ *to* $\mathsf{g}(\mathsf{0},z)$ *using* $y \to \mathsf{g}(y,z) \in \mathcal{R}^{-1}$.

Then we narrow 1 *to* $\mathsf{g}(y',1)$ *by* $z' \to \mathsf{g}(y',z')$. *Therefore we obtain* $\mathsf{F}(0,1,x) \rightsquigarrow^*_{(\mathcal{P}^{-1},\mathcal{R}^{-1}),[y/0,\, z'/1]} \mathsf{F}(\mathsf{g}(0,z),\mathsf{g}(y,1),x)$. *Now* $t' = \mathsf{F}(\mathsf{g}(0,z),\mathsf{g}(y,1),x)$ *(semi-) unifies with the corresponding right-hand side* $t = \mathsf{F}(x,x,x)$ *using* $\mu_1 = [x/\mathsf{g}(0,1),\ y/0,\ z/1]$. *Thus,* $(DP(\mathcal{R}),\mathcal{R},\mathbf{t})$ *is looping and the TRS is not terminating.*

However, there are also TRSs where backward narrowing fails and forward narrowing succeeds.

Example 39 *Let* $\mathcal{R} = \{\mathsf{f}(x,x) \to \mathsf{f}(0,1),\ 0 \to \mathsf{a},\ 1 \to \mathsf{a}\}$ *and* $\mathcal{P} = DP(\mathcal{R}) = \{\mathsf{F}(x,x) \to \mathsf{F}(0,1)\}$. *For* $\sigma(x) = \mathsf{a}$ *we have an infinite* $(\mathcal{P},\mathcal{R})$*-chain. But the left-hand side* $\mathsf{F}(x,x)$ *of* $\mathcal{P}$*'s only pair cannot be narrowed backwards and thus, Thm. 37 fails.*

On the other hand, the right-hand side $\mathsf{F}(0,1)$ *can be narrowed to* $\mathsf{F}(\mathsf{a},\mathsf{a})$. *This term obviously unifies with the left-hand side* $\mathsf{F}(x,x)$ *by* $\mu_1 = [x/\mathsf{a}]$. *Thus, with forward narrowing (Thm. 35) we can detect that this DP problem is looping and that the TRS is not terminating.*

Note that Ex. 30 where forward narrowing fails is not right-linear and that Ex. 39 where backward narrowing fails is not left-linear.[8] Therefore, we implemented the non-termination processor of Thm. 32 with the following heuristic in our system AProVE [15]:

- If $\mathcal{P} \cup \mathcal{R}$ is right- and not left-linear, then use forward narrowing (Thm. 35).
- Otherwise, we use backward narrowing (Thm. 37). If $\mathcal{P} \cup \mathcal{R}$ is not left-linear, then moreover we also permit narrowing steps in variables (i.e., $t|_p \in \mathcal{V}$ is permitted in Def. 33). The reason is that then there are looping DP problems which otherwise cannot be detected by forward or backward narrowing.[9]
- Moreover, to obtain a finite search space, we use an upper bound on the number of times that a rule from $\mathcal{P} \cup \mathcal{R}$ can be used for narrowing.

4.3 Comparison with Related Work

We use narrowing to identify looping DP problems. This is related to the concept of *forward closures* of a TRS $\mathcal{R}$ [10]. However, our approach differs from forward closures by starting from the rules of another TRS $\mathcal{P}$ and by also allowing narrowings with $\mathcal{P}$'s rules on root level. (The reason is that we prove non-termination within the DP framework.) Moreover, we also regard backward narrowing.

There are only few papers on automatically proving *non-termination* of TRSs. An early work is [26] which detects TRSs that are not *simply* terminating (but

[8] In fact, we conjecture that all looping DP problems can be detected by forward narrowing if $\mathcal{P} \cup \mathcal{R}$ is right-linear and by backward narrowing if $\mathcal{P} \cup \mathcal{R}$ is left-linear.

[9] An example is the well-known TRS of Drosten [11] (cf. also [5, Ex. 4.13]). Nevertheless, then there are also looping DP problems which cannot even be found when narrowing into variables. An example is $(\mathcal{P},\mathcal{R},\mathbf{t})$ where $\mathcal{P} = \{\mathsf{F}(x,x,y) \to \mathsf{F}(\mathsf{c}(\mathsf{g}(y,y)),\mathsf{c}(\mathsf{h}(y,y)),y)\}$ and $\mathcal{R} = \{\mathsf{g}(\mathsf{c}(0),\mathsf{c}(1)) \to \mathsf{b},\ \mathsf{h}(\mathsf{c}(0),\mathsf{c}(1)) \to \mathsf{b},\ \mathsf{a} \to 0,\ \mathsf{a} \to 1\}$. We have an infinite chain since $\mathsf{F}(\mathsf{c}(\mathsf{b}),\mathsf{c}(\mathsf{b}),\mathsf{c}(\mathsf{a})) \to_{\mathcal{P}} \mathsf{F}(\mathsf{c}(\mathsf{g}(\mathsf{c}(\mathsf{a}),\mathsf{c}(\mathsf{a}))),\mathsf{c}(\mathsf{h}(\mathsf{c}(\mathsf{a}),\mathsf{c}(\mathsf{a}))),\mathsf{c}(\mathsf{a})) \to^*_{\mathcal{R}} \mathsf{F}(\mathsf{c}(\mathsf{g}(\mathsf{c}(0),\mathsf{c}(1))),\ \mathsf{c}(\mathsf{h}(\mathsf{c}(0),\mathsf{c}(1))),\mathsf{c}(\mathsf{a})) \to^*_{\mathcal{R}} \mathsf{F}(\mathsf{c}(\mathsf{b}),\mathsf{c}(\mathsf{b}),\mathsf{c}(\mathsf{a})) \to_{\mathcal{P}} \ldots$ However, an infinite chain can only be obtained if one instantiates y by $\mathsf{c}(\mathsf{a})$. But this instantiation cannot be found by narrowing, not even when narrowing into variables (since no left-hand side of a rule starts with c).

they may still terminate). Recently, [31, 33] presented methods for proving non-termination of *string rewrite systems* (i.e., TRSs where all function symbols have arity 1). Similar to our approach, [31] uses (forward) narrowing and [33] uses ancestor graphs which correspond to (backward) narrowing. However, our approach differs substantially from [31, 33]: our technique works within the DP framework, whereas [31, 33] operate on the whole set of rules. Therefore, we can benefit from all previous DP processors which decompose the initial DP problem into smaller sub-problems and identify those parts which could cause non-termination. Moreover, we regard full term rewriting instead of string rewriting. Therefore, we use semi-unification to detect loops, whereas for string rewriting, matching is sufficient. Finally, we also presented a condition to disprove *innermost* termination, whereas [31, 33] only try to disprove full termination.

5 Experiments and Conclusion

The DP framework is a general concept for combining termination techniques in a modular way. We presented two important improvements: First, we extended the framework in order to handle higher-order functions, represented as applicative first-order TRSs. To this end, we developed three new contributions: a refined approximation of dependency graphs, an improved definition of usable rules, and a new processor to transform applicative DP problems into functional form. The advantages of our approach, also compared to related work, are the following: it is simple and very easy to integrate into any termination prover based on dependency pairs (e.g., AProVE [15], CiME [9], TTT [18]). Moreover, it encompasses the original DP framework, e.g., it is at least as successful on ordinary first-order functions as the original dependency pair technique. Finally, our approach treats untyped higher-order functions, i.e., it can be used for termination analysis of polymorphic and untyped functional languages.

As a second extension within the DP framework, we introduced a new processor for disproving termination automatically (an important problem which had hardly been tackled up to now). A major advantage of our approach is that it combines techniques for proving and for disproving termination in the DP framework, which is beneficial for both termination and non-termination analysis.

We implemented all these contributions in the newest version of our termination prover AProVE [15]. Due to the results of this paper, AProVE 1.2 was the most powerful tool for both termination and non-termination proofs of TRSs at the *Annual International Competition of Termination Tools 2005* [32]. In the following table, we compare AProVE 1.2 with its predecessor AProVE 1.1d-γ, which was the winning tool for TRSs at the competition in 2004. While AProVE 1.1d-γ already contained our results on non-termination analysis, the contributions on handling applicative TRSs from Sect. 3 were missing. For the experiments, we used the same setting as in the competition with a timeout of 60 seconds for each example (where however most proofs take less than two seconds).

	higher-order (61 TRSs)		*non-term* (90 TRSs)		*TPDB* (838 TRSs)	
	t	**n**	**t**	**n**	**t**	**n**
AProVE 1.2	43	8	25	61	639	95
AProVE 1.1d-γ	13	7	24	60	486	92

Here, "*higher-order*" is a collection of untyped versions of typical higher-order functions from [2, 3, 7, 23, 24, 30] and "*non-term*" contains particularly many non-terminating examples. "*TPDB*" is the *Termination Problem Data Base* used in the annual termination competition [32]. It consists of 838 (innermost) termination problems for TRSs from different sources. In the tables, **t** and **n** are the numbers of TRSs where **t**ermination resp. **n**on-termination could be proved.

AProVE 1.2 solves the vast majority of the examples in the "*higher-order*"- and the "*non-term*"-collection. This shows that our results for higher-order functions and non-termination are indeed successful in practice. In contrast, the first column demonstrates that previous techniques for automated termination proofs often fail on applicative TRSs representing higher-order functions. Finally, the last two columns show that our contributions also increase power substantially on ordinary non-applicative TRSs (which constitute most of the TPDB). For further details on our experiments and to download AProVE, the reader is referred to `http://www-i2.informatik.rwth-aachen.de/AProVE/1.2/`.

References

1. T. Aoto and T. Yamada. Termination of simply typed term rewriting systems by translation and labelling. In *Proc. RTA '03*, LNCS 2706, pages 380–394, 2003.
2. T. Aoto and T. Yamada. Termination of simply-typed applicative term rewriting systems. In *Proc. HOR '04*, Technical Report AIB-2004-03, RWTH Aachen, Germany, pages 61–65, 2004.
3. T. Aoto and T. Yamada. Dependency pairs for simply typed term rewriting. In *Proc. RTA '05*, LNCS 3467, pages 120–134, 2005.
4. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
5. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001. Available from `http://aib.informatik.rwth-aachen.de`.
6. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge, 1998.
7. R. Bird. *Introduction to Functional Programming using* Haskell. Prentice Hall, 1998.
8. C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proc. LPAR '01*, LNAI 2250, pages 531–547, 2001.
9. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME. `http://cime.lri.fr`.
10. N. Dershowitz. Termination of rewriting. *J. Symb. Comp.*, 3:69–116, 1987.
11. K. Drosten. *Termersetzungssysteme: Grundlagen der Prototyp-Generierung algebraischer Spezifikationen.* Springer, 1989.
12. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Appl. Algebra in Engineering, Communication and Computing*, 12(1,2):39–72, 2001.
13. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
14. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proc. LPAR '03*, LNAI 2850, pages 165–179, 2003.
15. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. RTA '04*, LNCS 3091, pages 210–220, 2004.
16. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR '04*, LNAI 3452, pages 301–331, 2005.
17. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proc. CADE '03*, LNAI 2741, pages 32–46, 2003. Full version to appear in *Information and Computation.*
18. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool. In *Proc. RTA '05*, LNCS 3467, pages 175–184, 2005.

19. J.-P. Jouannaud and A. Rubio. Higher-order recursive path orderings. In *Proc. LICS '99*, pages 402–411, 1999.
20. D. Kapur, D. Musser, P. Narendran, and J. Stillman. Semi-unification. *Theoretical Computer Science*, 81(2):169–187, 1991.
21. R. Kennaway, J. W. Klop, R. Sleep, and F.-J. de Vries. Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*, 21(1):15–39, 1996.
22. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. PPDP '99*, LNCS 1702, pages 48–62, 1999.
23. K. Kusakari. On proving termination of term rewriting systems with higher-order variables. *IPSJ Transactions on Programming*, 42(SIG 7 (PRO 11)):35–45, 2001.
24. M. Lifantsev and L. Bachmair. An LPO-based termination ordering for higher-order terms without λ-abstraction. In *Proc. TPHOLs '98*, LNCS 1479, 1998.
25. A. Oliart and W. Snyder. A fast algorithm for uniform semi-unification. In *Proc. CADE '98*, LNCS 1421, pages 239–253, 1998.
26. D. A. Plaisted. A simple non-termination test for the Knuth-Bendix method. In *Proc. CADE '86*, LNCS 230, pages 79–88, 1986.
27. M. Sakai, Y. Watanabe, and T. Sakabe. An extension of dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8):1025–1032, 2001.
28. M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Trans. on Inf. & Sys.*, 2005. To appear.
29. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proc. IJCAR '04*, LNAI 3097, pages 75–90, 2004.
30. Y. Toyama. Termination of S-expression rewriting systems: Lexicographic path ordering for higher-order terms. In *Proc. RTA '04*, LNCS 3091, pages 40–54, 2004.
31. J. Waldmann. `Matchbox`: A tool for match-bounded string rewriting. In *Proc. 15th RTA*, LNCS 3091, pages 85–94, 2004.
32. TPDB web page. `http://www.lri.fr/~marche/termination-competition/`.
33. H. Zantema. TORPA: Termination of string rewriting proved automatically. *Journal of Automated Reasoning*, 2005. To appear.

Aachener Informatik-Berichte

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

1987-01 *	Fachgruppe Informatik: Jahresbericht 1986
1987-02 *	David de Frutos Escrig, Klaus Indermark: Equivalence Relations of Non-Deterministic Ianov-Schemes
1987-03 *	Manfred Nagl: A Software Development Environment based on Graph Technology
1987-04 *	Claus Lewerentz, Manfred Nagl, Bernhard Westfechtel: On Integration Mechanisms within a Graph-Based Software Development Environment
1987-05 *	Reinhard Rinn: Über Eingabeanomalien bei verschiedenen Inferenzmodellen
1987-06 *	Werner Damm, Gert Döhmen: Specifying Distributed Computer Architectures in AADL*
1987-07 *	Gregor Engels, Claus Lewerentz, Wilhelm Schäfer: Graph Grammar Engineering: A Software Specification Method
1987-08 *	Manfred Nagl: Set Theoretic Approaches to Graph Grammars
1987-09 *	Claus Lewerentz, Andreas Schürr: Experiences with a Database System for Software Documents
1987-10 *	Herbert Klaeren, Klaus Indermark: A New Implementation Technique for Recursive Function Definitions
1987-11 *	Rita Loogen: Design of a Parallel Programmable Graph Reduction Machine with Distributed Memory
1987-12	J. Börstler, U. Möncke, R. Wilhelm: Table compression for tree automata
1988-01 *	Gabriele Esser, Johannes Rückert, Frank Wagner: Gesellschaftliche Aspekte der Informatik
1988-02 *	Peter Martini, Otto Spaniol: Token-Passing in High-Speed Backbone Networks for Campus-Wide Environments
1988-03 *	Thomas Welzel: Simulation of a Multiple Token Ring Backbone
1988-04 *	Peter Martini: Performance Comparison for HSLAN Media Access Protocols
1988-05 *	Peter Martini: Performance Analysis of Multiple Token Rings
1988-06 *	Andreas Mann, Johannes Rückert, Otto Spaniol: Datenfunknetze
1988-07 *	Andreas Mann, Johannes Rückert: Packet Radio Networks for Data Exchange
1988-08 *	Andreas Mann, Johannes Rückert: Concurrent Slot Assignment Protocol for Packet Radio Networks
1988-09 *	W. Kremer, F. Reichert, J. Rückert, A. Mann: Entwurf einer Netzwerktopologie für ein Mobilfunknetz zur Unterstützung des öffentlichen Straßenverkehrs
1988-10 *	Kai Jakobs: Towards User-Friendly Networking
1988-11 *	Kai Jakobs: The Directory - Evolution of a Standard
1988-12 *	Kai Jakobs: Directory Services in Distributed Systems - A Survey
1988-13 *	Martine Schümmer: RS-511, a Protocol for the Plant Floor

1988-14 * U. Quernheim: Satellite Communication Protocols - A Performance Comparison Considering On-Board Processing

1988-15 * Peter Martini, Otto Spaniol, Thomas Welzel: File Transfer in High Speed Token Ring Networks: Performance Evaluation by Approximate Analysis and Simulation

1988-16 * Fachgruppe Informatik: Jahresbericht 1987

1988-17 * Wolfgang Thomas: Automata on Infinite Objects

1988-18 * Michael Sonnenschein: On Petri Nets and Data Flow Graphs

1988-19 * Heiko Vogler: Functional Distribution of the Contextual Analysis in Block-Structured Programming Languages: A Case Study of Tree Transducers

1988-20 * Thomas Welzel: Einsatz des Simulationswerkzeuges QNAP2 zur Leistungsbewertung von Kommunikationsprotokollen

1988-21 * Th. Janning, C. Lewerentz: Integrated Project Team Management in a Software Development Environment

1988-22 * Joost Engelfriet, Heiko Vogler: Modular Tree Transducers

1988-23 * Wolfgang Thomas: Automata and Quantifier Hierarchies

1988-24 * Uschi Heuter: Generalized Definite Tree Languages

1989-01 * Fachgruppe Informatik: Jahresbericht 1988

1989-02 * G. Esser, J. Rückert, F. Wagner (Hrsg.): Gesellschaftliche Aspekte der Informatik

1989-03 * Heiko Vogler: Bottom-Up Computation of Primitive Recursive Tree Functions

1989-04 * Andy Schürr: Introduction to PROGRESS, an Attribute Graph Grammar Based Specification Language

1989-05 J. Börstler: Reuse and Software Development - Problems, Solutions, and Bibliography (in German)

1989-06 * Kai Jakobs: OSI - An Appropriate Basis for Group Communication?

1989-07 * Kai Jakobs: ISO's Directory Proposal - Evolution, Current Status and Future Problems

1989-08 * Bernhard Westfechtel: Extension of a Graph Storage for Software Documents with Primitives for Undo/Redo and Revision Control

1989-09 * Peter Martini: High Speed Local Area Networks - A Tutorial

1989-10 * P. Davids, Th. Welzel: Performance Analysis of DQDB Based on Simulation

1989-11 * Manfred Nagl (Ed.): Abstracts of Talks presented at the WG '89 15th International Workshop on Graphtheoretic Concepts in Computer Science

1989-12 * Peter Martini: The DQDB Protocol - Is it Playing the Game?

1989-13 * Martine Schümmer: CNC/DNC Communication with MAP

1989-14 * Martine Schümmer: Local Area Networks for Manufactoring Environments with hard Real-Time Requirements

1989-15 * M. Schümmer, Th. Welzel, P. Martini: Integration of Field Bus and MAP Networks - Hierarchical Communication Systems in Production Environments

1989-16 * G. Vossen, K.-U. Witt: SUXESS: Towards a Sound Unification of Extensions of the Relational Data Model

1989-17 * J. Derissen, P. Hruschka, M.v.d. Beeck, Th. Janning, M. Nagl: Integrating Structured Analysis and Information Modelling

1989-18 A. Maassen: Programming with Higher Order Functions

1989-19 * Mario Rodriguez-Artalejo, Heiko Vogler: A Narrowing Machine for Syntax Directed BABEL

1989-20 H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Graph-based Implementation of a Functional Logic Language

1990-01 * Fachgruppe Informatik: Jahresbericht 1989

1990-02 * Vera Jansen, Andreas Potthoff, Wolfgang Thomas, Udo Wermuth: A Short Guide to the AMORE System (Computing Automata, MOnoids and Regular Expressions)

1990-03 * Jerzy Skurczynski: On Three Hierarchies of Weak SkS Formulas

1990-04 R. Loogen: Stack-based Implementation of Narrowing

1990-05 H. Kuchen, A. Wagener: Comparison of Dynamic Load Balancing Strategies

1990-06 * Kai Jakobs, Frank Reichert: Directory Services for Mobile Communication

1990-07 * Kai Jakobs: What's Beyond the Interface - OSI Networks to Support Cooperative Work

1990-08 * Kai Jakobs: Directory Names and Schema - An Evaluation

1990-09 * Ulrich Quernheim, Dieter Kreuer: Das CCITT - Signalisierungssystem Nr. 7 auf Satellitenstrecken; Simulation der Zeichengabestrecke

1990-11 H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Lazy Narrowing in a Graph Machine

1990-12 * Kai Jakobs, Josef Kaltwasser, Frank Reichert, Otto Spaniol: Der Computer fährt mit

1990-13 * Rudolf Mathar, Andreas Mann: Analyzing a Distributed Slot Assignment Protocol by Markov Chains

1990-14 A. Maassen: Compilerentwicklung in Miranda - ein Praktikum in funktionaler Programmierung (written in german)

1990-15 * Manfred Nagl, Andreas Schürr: A Specification Environment for Graph Grammars

1990-16 A. Schürr: PROGRESS: A VHL-Language Based on Graph Grammars

1990-17 * Marita Möller: Ein Ebenenmodell wissensbasierter Konsultationen - Unterstützung für Wissensakquisition und Erklärungsfähigkeit

1990-18 * Eric Kowalewski: Entwurf und Interpretation einer Sprache zur Beschreibung von Konsultationsphasen in Expertensystemen

1990-20 Y. Ortega Mallen, D. de Frutos Escrig: A Complete Proof System for Timed Observations

1990-21 * Manfred Nagl: Modelling of Software Architectures: Importance, Notions, Experiences

1990-22 H. Fassbender, H. Vogler: A Call-by-need Implementation of Syntax Directed Functional Programming

1991-01 Guenther Geiler (ed.), Fachgruppe Informatik: Jahresbericht 1990

1991-03 B. Steffen, A. Ingolfsdottir: Characteristic Formulae for Processes with Divergence

1991-04 M. Portz: A new class of cryptosystems based on interconnection networks

1991-05 H. Kuchen, G. Geiler: Distributed Applicative Arrays

1991-06 * Ludwig Staiger: Kolmogorov Complexity and Hausdorff Dimension

1991-07 * Ludwig Staiger: Syntactic Congruences for w-languages

1991-09 * Eila Kuikka: A Proposal for a Syntax-Directed Text Processing System

1991-10 K. Gladitz, H. Fassbender, H. Vogler: Compiler-based Implementation of Syntax-Directed Functional Programming

1991-11 R. Loogen, St. Winkler: Dynamic Detection of Determinism in Functional Logic Languages

1991-12 * K. Indermark, M. Rodriguez Artalejo (Eds.): Granada Workshop on the Integration of Functional and Logic Programming

1991-13 * Rolf Hager, Wolfgang Kremer: The Adaptive Priority Scheduler: A More Fair Priority Service Discipline

1991-14 * Andreas Fasbender, Wolfgang Kremer: A New Approximation Algorithm for Tandem Networks with Priority Nodes

1991-15 J. Börstler, A. Zündorf: Revisiting extensions to Modula-2 to support reusability

1991-16 J. Börstler, Th. Janning: Bridging the gap between Requirements Analysis and Design

1991-17 A. Zündorf, A. Schürr: Nondeterministic Control Structures for Graph Rewriting Systems

1991-18 * Matthias Jarke, John Mylopoulos, Joachim W. Schmidt, Yannis Vassiliou: DAIDA: An Environment for Evolving Information Systems

1991-19 M. Jeusfeld, M. Jarke: From Relational to Object-Oriented Integrity Simplification

1991-20 G. Hogen, A. Kindler, R. Loogen: Automatic Parallelization of Lazy Functional Programs

1991-21 * Prof. Dr. rer. nat. Otto Spaniol: ODP (Open Distributed Processing): Yet another Viewpoint

1991-22 H. Kuchen, F. Lücking, H. Stoltze: The Topology Description Language TDL

1991-23 S. Graf, B. Steffen: Compositional Minimization of Finite State Systems

1991-24 R. Cleaveland, J. Parrow, B. Steffen: The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems

1991-25 * Rudolf Mathar, Jürgen Mattfeldt: Optimal Transmission Ranges for Mobile Communication in Linear Multihop Packet Radio Networks

1991-26 M. Jeusfeld, M. Staudt: Query Optimization in Deductive Object Bases

1991-27 J. Knoop, B. Steffen: The Interprocedural Coincidence Theorem

1991-28 J. Knoop, B. Steffen: Unifying Strength Reduction and Semantic Code Motion

1991-30 T. Margaria: First-Order theories for the verification of complex FSMs

1991-31 B. Steffen: Generating Data Flow Analysis Algorithms from Modal Specifications

1992-01 Stefan Eherer (ed.), Fachgruppe Informatik: Jahresbericht 1991

1992-02 * Bernhard Westfechtel: Basismechanismen zur Datenverwaltung in strukturbezogenen Hypertextsystemen

1992-04 S. A. Smolka, B. Steffen: Priority as Extremal Probability

1992-05 * Matthias Jarke, Carlos Maltzahn, Thomas Rose: Sharing Processes: Team Coordination in Design Repositories

1992-06 O. Burkart, B. Steffen: Model Checking for Context-Free Processes
1992-07 * Matthias Jarke, Klaus Pohl: Information Systems Quality and Quality Information Systems
1992-08 * Rudolf Mathar, Jürgen Mattfeldt: Analyzing Routing Strategy NFP in Multihop Packet Radio Networks on a Line
1992-09 * Alfons Kemper, Guido Moerkotte: Grundlagen objektorientierter Datenbanksysteme
1992-10 Matthias Jarke, Manfred Jeusfeld, Andreas Miethsam, Michael Gocek: Towards a logic-based reconstruction of software configuration management
1992-11 Werner Hans: A Complete Indexing Scheme for WAM-based Abstract Machines
1992-12 W. Hans, R. Loogen, St. Winkler: On the Interaction of Lazy Evaluation and Backtracking
1992-13 * Matthias Jarke, Thomas Rose: Specification Management with CAD
1992-14 Th. Noll, H. Vogler: Top-down Parsing with Simultaneous Evaluation on Noncircular Attribute Grammars
1992-15 A. Schuerr, B. Westfechtel: Graphgrammatiken und Graphersetzungssysteme(written in german)
1992-16 * Graduiertenkolleg Informatik und Technik (Hrsg.): Forschungsprojekte des Graduiertenkollegs Informatik und Technik
1992-17 M. Jarke (ed.): ConceptBase V3.1 User Manual
1992-18 * Clarence A. Ellis, Matthias Jarke (Eds.): Distributed Cooperation in Integrated Information Systems - Proceedings of the Third International Workshop on Intelligent and Cooperative Information Systems
1992-19-00 H. Kuchen, R. Loogen (eds.): Proceedings of the 4th Int. Workshop on the Parallel Implementation of Functional Languages
1992-19-01 G. Hogen, R. Loogen: PASTEL - A Parallel Stack-Based Implementation of Eager Functional Programs with Lazy Data Structures (Extended Abstract)
1992-19-02 H. Kuchen, K. Gladitz: Implementing Bags on a Shared Memory MIMD-Machine
1992-19-03 C. Rathsack, S.B. Scholz: LISA - A Lazy Interpreter for a Full-Fledged Lambda-Calculus
1992-19-04 T.A. Bratvold: Determining Useful Parallelism in Higher Order Functions
1992-19-05 S. Kahrs: Polymorphic Type Checking by Interpretation of Code
1992-19-06 M. Chakravarty, M. Köhler: Equational Constraints, Residuation, and the Parallel JUMP-Machine
1992-19-07 J. Seward: Polymorphic Strictness Analysis using Frontiers (Draft Version)
1992-19-08 D. Gärtner, A. Kimms, W. Kluge: pi-Red^+ - A Compiling Graph-Reduction System for a Full Fledged Lambda-Calculus
1992-19-09 D. Howe, G. Burn: Experiments with strict STG code
1992-19-10 J. Glauert: Parallel Implementation of Functional Languages Using Small Processes
1992-19-11 M. Joy, T. Axford: A Parallel Graph Reduction Machine
1992-19-12 A. Bennett, P. Kelly: Simulation of Multicache Parallel Reduction

1992-19-13 K. Langendoen, D.J. Agterkamp: Cache Behaviour of Lazy Functional Programs (Working Paper)
1992-19-14 K. Hammond, S. Peyton Jones: Profiling scheduling strategies on the GRIP parallel reducer
1992-19-15 S. Mintchev: Using Strictness Information in the STG-machine
1992-19-16 D. Rushall: An Attribute Grammar Evaluator in Haskell
1992-19-17 J. Wild, H. Glaser, P. Hartel: Statistics on storage management in a lazy functional language implementation
1992-19-18 W.S. Martins: Parallel Implementations of Functional Languages
1992-19-19 D. Lester: Distributed Garbage Collection of Cyclic Structures (Draft version)
1992-19-20 J.C. Glas, R.F.H. Hofman, W.G. Vree: Parallelization of Branch-and-Bound Algorithms in a Functional Programming Environment
1992-19-21 S. Hwang, D. Rushall: The nu-STG machine: a parallelized Spineless Tagless Graph Reduction Machine in a distributed memory architecture (Draft version)
1992-19-22 G. Burn, D. Le Metayer: Cps-Translation and the Correctness of Optimising Compilers
1992-19-23 S.L. Peyton Jones, P. Wadler: Imperative functional programming (Brief summary)
1992-19-24 W. Damm, F. Liu, Th. Peikenkamp: Evaluation and Parallelization of Functions in Functional + Logic Languages (abstract)
1992-19-25 M. Kesseler: Communication Issues Regarding Parallel Functional Graph Rewriting
1992-19-26 Th. Peikenkamp: Charakterizing and representing neededness in functional loginc languages (abstract)
1992-19-27 H. Doerr: Monitoring with Graph-Grammars as formal operational Models
1992-19-28 J. van Groningen: Some implementation aspects of Concurrent Clean on distributed memory architectures
1992-19-29 G. Ostheimer: Load Bounding for Implicit Parallelism (abstract)
1992-20 H. Kuchen, F.J. Lopez Fraguas, J.J. Moreno Navarro, M. Rodriguez Artalejo: Implementing Disequality in a Lazy Functional Logic Language
1992-21 H. Kuchen, F.J. Lopez Fraguas: Result Directed Computing in a Functional Logic Language
1992-22 H. Kuchen, J.J. Moreno Navarro, M.V. Hermenegildo: Independent AND-Parallel Narrowing
1992-23 T. Margaria, B. Steffen: Distinguishing Formulas for Free
1992-24 K. Pohl: The Three Dimensions of Requirements Engineering
1992-25 * R. Stainov: A Dynamic Configuration Facility for Multimedia Communications
1992-26 * Michael von der Beeck: Integration of Structured Analysis and Timed Statecharts for Real-Time and Concurrency Specification
1992-27 W. Hans, St. Winkler: Aliasing and Groundness Analysis of Logic Programs through Abstract Interpretation and its Safety
1992-28 * Gerhard Steinke, Matthias Jarke: Support for Security Modeling in Information Systems Design
1992-29 B. Schinzel: Warum Frauenforschung in Naturwissenschaft und Technik

1992-30 A. Kemper, G. Moerkotte, K. Peithner: Object-Orientation Axiomatised by Dynamic Logic

1992-32 * Bernd Heinrichs, Kai Jakobs: Timer Handling in High-Performance Transport Systems

1992-33 * B. Heinrichs, K. Jakobs, K. Lenßen, W. Reinhardt, A. Spinner: Euro-Bridge: Communication Services for Multimedia Applications

1992-34 C. Gerlhof, A. Kemper, Ch. Kilger, G. Moerkotte: Partition-Based Clustering in Object Bases: From Theory to Practice

1992-35 J. Börstler: Feature-Oriented Classification and Reuse in IPSEN

1992-36 M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassiliou: Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis

1992-37 * K. Pohl, M. Jarke: Quality Information Systems: Repository Support for Evolving Process Models

1992-38 A. Zuendorf: Implementation of the imperative / rule based language PROGRES

1992-39 P. Koch: Intelligentes Backtracking bei der Auswertung funktional-logischer Programme

1992-40 * Rudolf Mathar, Jürgen Mattfeldt: Channel Assignment in Cellular Radio Networks

1992-41 * Gerhard Friedrich, Wolfgang Neidl: Constructive Utility in Model-Based Diagnosis Repair Systems

1992-42 * P. S. Chen, R. Hennicker, M. Jarke: On the Retrieval of Reusable Software Components

1992-43 W. Hans, St.Winkler: Abstract Interpretation of Functional Logic Languages

1992-44 N. Kiesel, A. Schuerr, B. Westfechtel: Design and Evaluation of GRAS, a Graph-Oriented Database System for Engineering Applications

1993-01 * Fachgruppe Informatik: Jahresbericht 1992

1993-02 * Patrick Shicheng Chen: On Inference Rules of Logic-Based Information Retrieval Systems

1993-03 G. Hogen, R. Loogen: A New Stack Technique for the Management of Runtime Structures in Distributed Environments

1993-05 A. Zuendorf: A Heuristic for the Subgraph Isomorphism Problem in Executing PROGRES

1993-06 A. Kemper, D. Kossmann: Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis

1993-07 * Graduiertenkolleg Informatik und Technik (Hrsg.): Graduiertenkolleg Informatik und Technik

1993-08 * Matthias Berger: k-Coloring Vertices using a Neural Network with Convergence to Valid Solutions

1993-09 M. Buchheit, M. Jeusfeld, W. Nutt, M. Staudt: Subsumption between Queries to Object-Oriented Databases

1993-10 O. Burkart, B. Steffen: Pushdown Processes: Parallel Composition and Model Checking

1993-11 * R. Große-Wienker, O. Hermanns, D. Menzenbach, A. Pollacks, S. Repetzki, J. Schwartz, K. Sonnenschein, B. Westfechtel: Das SUKITS-Projekt: A-posteriori-Integration heterogener CIM-Anwendungssysteme

1993-12 * Rudolf Mathar, Jürgen Mattfeldt: On the Distribution of Cumulated Interference Power in Rayleigh Fading Channels
1993-13 O. Maler, L. Staiger: On Syntactic Congruences for omega-languages
1993-14 M. Jarke, St. Eherer, R. Gallersdoerfer, M. Jeusfeld, M. Staudt: ConceptBase - A Deductive Object Base Manager
1993-15 M. Staudt, H.W. Nissen, M.A. Jeusfeld: Query by Class, Rule and Concept
1993-16 * M. Jarke, K. Pohl, St. Jacobs et al.: Requirements Engineering: An Integrated View of Representation Process and Domain
1993-17 * M. Jarke, K. Pohl: Establishing Vision in Context: Towards a Model of Requirements Processes
1993-18 W. Hans, H. Kuchen, St. Winkler: Full Indexing for Lazy Narrowing
1993-19 W. Hans, J.J. Ruz, F. Saenz, St. Winkler: A VHDL Specification of a Shared Memory Parallel Machine for Babel
1993-20 * K. Finke, M. Jarke, P. Szczurko, R. Soltysiak: Quality Management for Expert Systems in Process Control
1993-21 M. Jarke, M.A. Jeusfeld, P. Szczurko: Three Aspects of Intelligent Cooperation in the Quality Cycle
1994-01 Margit Generet, Sven Martin (eds.), Fachgruppe Informatik: Jahresbericht 1993
1994-02 M. Lefering: Development of Incremental Integration Tools Using Formal Specifications
1994-03 * P. Constantopoulos, M. Jarke, J. Mylopoulos, Y. Vassiliou: The Software Information Base: A Server for Reuse
1994-04 * Rolf Hager, Rudolf Mathar, Jürgen Mattfeldt: Intelligent Cruise Control and Reliable Communication of Mobile Stations
1994-05 * Rolf Hager, Peter Hermesmann, Michael Portz: Feasibility of Authentication Procedures within Advanced Transport Telematics
1994-06 * Claudia Popien, Bernd Meyer, Axel Kuepper: A Formal Approach to Service Import in ODP Trader Federations
1994-07 P. Peters, P. Szczurko: Integrating Models of Quality Management Methods by an Object-Oriented Repository
1994-08 * Manfred Nagl, Bernhard Westfechtel: A Universal Component for the Administration in Distributed and Integrated Development Environments
1994-09 * Patrick Horster, Holger Petersen: Signatur- und Authentifikationsverfahren auf der Basis des diskreten Logarithmusproblems
1994-11 A. Schürr: PROGRES, A Visual Language and Environment for PROgramming with Graph REwrite Systems
1994-12 A. Schürr: Specification of Graph Translators with Triple Graph Grammars
1994-13 A. Schürr: Logic Based Programmed Structure Rewriting Systems
1994-14 L. Staiger: Codes, Simplifying Words, and Open Set Condition
1994-15 * Bernhard Westfechtel: A Graph-Based System for Managing Configurations of Engineering Design Documents
1994-16 P. Klein: Designing Software with Modula-3
1994-17 I. Litovsky, L. Staiger: Finite acceptance of infinite words

1994-18 G. Hogen, R. Loogen: Parallel Functional Implementations: Graphbased vs. Stackbased Reduction

1994-19 M. Jeusfeld, U. Johnen: An Executable Meta Model for Re-Engineering of Database Schemas

1994-20 * R. Gallersdörfer, M. Jarke, K. Klabunde: Intelligent Networks as a Data Intensive Application (INDIA)

1994-21 M. Mohnen: Proving the Correctness of the Static Link Technique Using Evolving Algebras

1994-22 H. Fernau, L. Staiger: Valuations and Unambiguity of Languages, with Applications to Fractal Geometry

1994-24 * M. Jarke, K. Pohl, R. Dömges, St. Jacobs, H. W. Nissen: Requirements Information Management: The NATURE Approach

1994-25 * M. Jarke, K. Pohl, C. Rolland, J.-R. Schmitt: Experience-Based Method Evaluation and Improvement: A Process Modeling Approach

1994-26 * St. Jacobs, St. Kethers: Improving Communication and Decision Making within Quality Function Deployment

1994-27 * M. Jarke, H. W. Nissen, K. Pohl: Tool Integration in Evolving Information Systems Environments

1994-28 O. Burkart, D. Caucal, B. Steffen: An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes

1995-01 * Fachgruppe Informatik: Jahresbericht 1994

1995-02 Andy Schürr, Andreas J. Winter, Albert Zündorf: Graph Grammar Engineering with PROGRES

1995-03 Ludwig Staiger: A Tight Upper Bound on Kolmogorov Complexity by Hausdorff Dimension and Uniformly Optimal Prediction

1995-04 Birgitta König-Ries, Sven Helmer, Guido Moerkotte: An experimental study on the complexity of left-deep join ordering problems for cyclic queries

1995-05 Sophie Cluet, Guido Moerkotte: Efficient Evaluation of Aggregates on Bulk Types

1995-06 Sophie Cluet, Guido Moerkotte: Nested Queries in Object Bases

1995-07 Sophie Cluet, Guido Moerkotte: Query Optimization Techniques Exploiting Class Hierarchies

1995-08 Markus Mohnen: Efficient Compile-Time Garbage Collection for Arbitrary Data Structures

1995-09 Markus Mohnen: Functional Specification of Imperative Programs: An Alternative Point of View of Functional Languages

1995-10 Rainer Gallersdörfer, Matthias Nicola: Improving Performance in Replicated Databases through Relaxed Coherency

1995-11 * M.Staudt, K.von Thadden: Subsumption Checking in Knowledge Bases

1995-12 * G.V.Zemanek, H.W.Nissen, H.Hubert, M.Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology

1995-13 * M.Staudt, M.Jarke: Incremental Maintenance of Externally Materialized Views

1995-14 * P.Peters, P.Szczurko, M.Jeusfeld: Oriented Information Management: Conceptual Models at Work

1995-15 * Matthias Jarke, Sudha Ram (Hrsg.): WITS 95 Proceedings of the 5th Annual Workshop on Information Technologies and Systems

1995-16 * W.Hans, St.Winkler, F.Saenz: Distributed Execution in Functional Logic Programming

1996-01 * Jahresbericht 1995

1996-02 Michael Hanus, Christian Prehofer: Higher-Order Narrowing with Definitional Trees

1996-03 * W.Scheufele, G.Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates

1996-04 Klaus Pohl: PRO-ART: Enabling Requirements Pre-Traceability

1996-05 Klaus Pohl: Requirements Engineering: An Overview

1996-06 * M.Jarke, W.Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools

1996-07 Olaf Chitil: The Sigma-Semantics: A Comprehensive Semantics for Functional Programs

1996-08 * S.Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics

1996-09 Michael Hanus (Ed.): Proceedings of the Poster Session of ALP96 - Fifth International Conference on Algebraic and Logic Programming

1996-09-0 Michael Hanus (Ed.): Proceedings of the Poster Session of ALP 96 - Fifth International Conference on Algebraic and Logic Programming: Introduction and table of contents

1996-09-1 Ilies Alouini: An Implementation of Conditional Concurrent Rewriting on Distributed Memory Machines

1996-09-2 Olivier Danvy, Karoline Malmkjær: On the Idempotence of the CPS Transformation

1996-09-3 Victor M. Gulias, José L. Freire: Concurrent Programming in Haskell

1996-09-4 Sébastien Limet, Pierre Réty: On Decidability of Unifiability Modulo Rewrite Systems

1996-09-5 Alexandre Tessier: Declarative Debugging in Constraint Logic Programming

1996-10 Reidar Conradi, Bernhard Westfechtel: Version Models for Software Configuration Management

1996-11 * C.Weise, D.Lenzkes: A Fast Decision Algorithm for Timed Refinement

1996-12 * R.Dömges, K.Pohl, M.Jarke, B.Lohmann, W.Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models

1996-13 * K.Pohl, R.Klamma, K.Weidenhaupt, R.Dömges, P.Haumer, M.Jarke: A Framework for Process-Integrated Tools

1996-14 * R.Gallersdörfer, K.Klabunde, A.Stolz, M.Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996

1996-15 * H.Schimpe, M.Staudt: VAREX: An Environment for Validating and Refining Rule Bases

1996-16 * M.Jarke, M.Gebhardt, S.Jacobs, H.Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization

1996-17 Manfred A. Jeusfeld, Tung X. Bui: Decision Support Components on the Internet

1996-18 Manfred A. Jeusfeld, Mike Papazoglou: Information Brokering: Design, Search and Transformation

1996-19 * P.Peters, M.Jarke: Simulating the impact of information flows in networked organizations

1996-20 Matthias Jarke, Peter Peters, Manfred A. Jeusfeld: Model-driven planning and design of cooperative information systems

1996-21 * G.de Michelis, E.Dubois, M.Jarke, F.Matthes, J.Mylopoulos, K.Pohl, J.Schmidt, C.Woo, E.Yu: Cooperative information systems: a manifesto

1996-22 * S.Jacobs, M.Gebhardt, S.Kethers, W.Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality

1996-23 * M.Gebhardt, S.Jacobs: Conflict Management in Design

1997-01 Michael Hanus, Frank Zartmann (eds.): Jahresbericht 1996

1997-02 Johannes Faassen: Using full parallel Boltzmann Machines for Optimization

1997-03 Andreas Winter, Andy Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems

1997-04 Markus Mohnen, Stefan Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler

1997-05 * S.Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge

1997-06 Matthias Nicola, Matthias Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries

1997-07 Petra Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen

1997-08 Dorothea Blostein, Andy Schürr: Computing with Graphs and Graph Rewriting

1997-09 Carl-Arndt Krapp, Bernhard Westfechtel: Feedback Handling in Dynamic Task Nets

1997-10 Matthias Nicola, Matthias Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases

1997-11 * R. Klamma, P. Peters, M. Jarke: Workflow Support for Failure Management in Federated Organizations

1997-13 Markus Mohnen: Optimising the Memory Management of Higher-Order Functional Programs

1997-14 Roland Baumann: Client/Server Distribution in a Structure-Oriented Database Management System

1997-15 George Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms

1998-01 * Fachgruppe Informatik: Jahresbericht 1997

1998-02 Stefan Gruner, Manfred Nagl, Andy Schürr: Fine-grained and Structure-Oriented Document Integration Tools are Needed for Development Processes

1998-03 Stefan Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr

1998-04 * O. Kubitz: Mobile Robots in Dynamic Environments

1998-05 Martin Leucker, Stephan Tobies: Truth - A Verification Platform for Distributed Systems

1998-06 * Matthias Oliver Berger: DECT in the Factory of the Future
1998-07 M. Arnold, M. Erdmann, M. Glinz, P. Haumer, R. Knoll, B. Paech, K. Pohl, J. Ryser, R. Studer, K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects
1998-08 * H. Aust: Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen
1998-09 * Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien
1998-10 * M. Nicola, M. Jarke: Performance Modeling of Distributed and Replicated Databases
1998-11 * Ansgar Schleicher, Bernhard Westfechtel, Dirk Jäger: Modeling Dynamic Software Processes in UML
1998-12 * W. Appelt, M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web
1998-13 Klaus Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation
1999-01 * Jahresbericht 1998
1999-02 * F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version
1999-03 * R. Gallersdörfer, M. Jarke, M. Nicola: The ADR Replication Manager
1999-04 Maria Alpuente, Michael Hanus, Salvador Lucas, Germán Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing
1999-05 * W. Thomas (Ed.): DLT 99 - Developments in Language Theory Fourth International Conference
1999-06 * Kai Jakobs, Klaus-Dieter Kleefeld: Informationssysteme für die angewandte historische Geographie
1999-07 Thomas Wilke: CTL+ is exponentially more succinct than CTL
1999-08 Oliver Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures
2000-01 * Jahresbericht 1999
2000-02 Jens Vöge, Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games
2000-04 Andreas Becks, Stefan Sklorz, Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach
2000-05 Mareike Schoop: Cooperative Document Management
2000-06 Mareike Schoop, Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling
2000-07 * Markus Mohnen, Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages
2000-08 Thomas Arts, Thomas Noll: Verifying Generic Erlang Client-Server Implementations
2001-01 * Jahresbericht 2000
2001-02 Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03 Thierry Cachat: The power of one-letter rational languages

2001-04 Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05 Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07 Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08 Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09 Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11 Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 * Jahresbericht 2001
2002-02 Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03 Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04 Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05 Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07 Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09 Markus Mohnen: Interfaces with Default Implementations in Java
2002-10 Martin Leucker: Logics for Mazurkiewicz traces
2002-11 Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 * Jahresbericht 2002
2003-02 Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03 Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations
2003-04 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05 Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06 Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates
2003-07 Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs
2004-01 * Fachgruppe Informatik: Jahresbericht 2004

2004-02 Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03 Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04 Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05 Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06 Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07 Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08 Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10 Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 * Fachgruppe Informatik: Jahresbericht 2005
2005-02 Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"

* These reports are only available as a printed version.
Please contact biblio@informatik.rwth-aachen.de to obtain copies.