Certification of Nontermination Proofs using Strategies and Nonlooping Derivations*

Julian Nagele, René Thiemann, and Sarah Winkler

Institute of Computer Science, University of Innsbruck, Austria
{julian.nagele,rene.thiemann,sarah.winkler}@uibk.ac.at

Abstract. The development of sophisticated termination criteria for term rewrite systems has led to powerful and complex tools that produce (non)termination proofs automatically. While many techniques to establish termination have already been formalized—thereby allowing to certify such proofs—this is not the case for nontermination. In particular, the proof checker CeTA was so far limited to (innermost) loops. In this paper we present an Isabelle/HOL formalization of an extended repertoire of nontermination techniques. First, we formalized techniques for nonlooping nontermination. Second, the available strategies include (an extended version of) forbidden patterns, which cover in particular outermost and context-sensitive rewriting. Finally, a mechanism to support partial nontermination proofs further extends the applicability of our proof checker.

1 Introduction

Program verification aims to establish certain properties of pieces of software, such as termination. But in presence of bugs it is often at least as important to show the negative property by means of a counter-example or, more generally, a disproof, such as a nontermination argument.

In this paper we consider term rewrite systems (TRSs) which constitute a powerful means to express functional programs in a compact way, and are thus a natural input format for program analysis. However, many programming languages employ particular evaluation strategies that are to be considered in program analysis. Thus also TRSs have to be analyzed with respect to specific strategies. In particular, a TRS which is nonterminating when ignoring the strategy may still be terminating when the evaluation respects the strategy.

Sophisticated techniques to analyze termination of TRSs (under strategies) have been developed and implemented in tools for automated termination analysis like AProVE [6] and $\mathsf{T_TT_2}$ [12]. However, these tools are complex and thus one should not blindly trust them: ever so often some tool delivers an incorrect proof, which remains undetected unless another prover gives the opposite answer on the same TRS. Therefore, it is of major importance to independently certify the generated proofs, which can be done using various certifiers [3,4,21] that rely on formalizations within some trusted proof assistant. Due to certification, bugs

* This research was supported by the Austrian Science Fund (FWF): P22767 and I963.

have been revealed in termination tools that have gone unnoticed for years and were easily fixed after they have been detected.

Our certifier for nontermination techniques is developed in the proof assistant Isabelle/HOL [16], and a preliminary version was already described in [23], which however was quite limited: only looping TRSs $\mathcal{R}$ could be treated, i.e., TRSs which admit derivations of the form $t \to_{\mathcal{R}}^{+} C[t\mu]$ for some term t, context C, and substitution μ; and the only supported strategy was innermost. There are even more severe restrictions for the other certifiers: [3] only supports loops without strategy, and [4] does not support nontermination proofs at all.

In the meanwhile, we extended our repertoire of formalized nontermination techniques. It now covers techniques for *nonlooping* nonterminating TRSs. Moreover, as strategy specification we now support an extended version of *forbidden patterns* [9], which generalizes many common strategies like (leftmost)-innermost, (leftmost)-outermost, and context-sensitive rewriting [15]. Finally, we also integrated a mechanism to support *partial* nontermination proofs, which further increases the applicability of our certifier and led to the detection of a severe soundness bug of AProVE, which has now been fixed.

We consider our contributions threefold. First and foremost, our extensions significantly increased the number of certifiable nontermination proofs. Second, on the theory level we could drastically simplify one of the algorithms for checking nontermination using forbidden patterns, and relax the preconditions for applying the technique of rewriting dependency pairs (cf. Theorem 14). Finally, we illustrate how termination checkers can benefit from certification: we used Isabelle's code generator [10] to integrate the executable functions from our certifier in TTT2, such that this tool is now able to automatically generate nontermination proofs involving general forbidden pattern strategies. This nearly doubled the number of generated nontermination proofs of TTT2.

The remainder is structured as follows. In § 2 we give preliminaries. In § 3 we explain our formalization of loop detection involving forbidden patterns. Afterwards, § 4 deals with techniques that allow to disprove termination of nonlooping TRSs, namely the techniques of rewriting and narrowing dependency pairs [7], the switch between innermost termination and termination [8], and a direct technique to disprove termination [5]. Experimental data is provided in § 5 where we also explain how we integrated forbidden patterns in TTT2, and why and how we added support for partial nontermination proofs to CeTA. We conclude in § 6.

Our formalization is part of the **Isa**belle **F**ormalization **o**f **R**ewriting (IsaFoR) which also includes our certifier CeTA [21]. Since IsaFoR contains every tiny detail of each proof, in the paper we just highlight some differences between the formalization and the paper proofs. Both IsaFoR and all details on our experiments are available at http://cl-informatik.uibk.ac.at/software/ceta/experiments/ntcert/.

2 Preliminaries

We refer to [2] for the basics of rewriting. We use ℓ, r, s, t, u, w for terms, f, g

for function symbols, x, y for variables, σ, μ, τ, δ for substitutions, i, j, k, n, m for natural numbers, o, p, q for positions, C, D for contexts, and $\mathcal{P}$, $\mathcal{R}$ for TRSs. Here, substitutions are mappings from variables to terms, and $t\mu$ is the term t where each variable x in t has been replaced by $\mu(x)$; contexts are terms which contain exactly one hole $\Box$, $t[\cdot]_p$ is the context that is obtained by replacing the subterm $t|_p$ of t at position p by the hole $\Box$. The term $C[t]$ is the term where the hole in C is replaced by t. We write $s \trianglerighteq t$ if $s = C[t]$ for some context C and $s \vartriangleright t$ if $s \trianglerighteq t$ and $s \neq t$. A position p is left of q iff $p = o\,i\,p'$, $q = o\,j\,q'$, and $i < j$. The set of positions in a term t is written as $\mathcal{P}\mathsf{os}(t)$ and ε denotes the empty position. The set of variables is $\mathcal{V}$, and $\mathcal{V}(t)$ are the variables within a term t.

A TRS $\mathcal{R}$ is a set of rewrite rules $\ell \to r$. The rewrite relation of $\mathcal{R}$ at position p is defined by $t \to_{\mathcal{R},p} s$ iff $t = C[\ell\sigma]$ and $s = C[r\sigma]$ for some rule $\ell \to r \in \mathcal{R}$, substitution σ, and context C with $C|_p = \Box$. In this case, the term $\ell\sigma$ is called a redex at position p. The reduction is outermost iff there is no redex above p, and it is innermost (denoted $\xrightarrow{\mathsf{i}}_{\mathcal{R},p}$) iff there are no redexes below p. We often omit p and $\mathcal{R}$ in a reduction $\to_{\mathcal{R},p}$, if $\mathcal{R}$ is obvious from the context, and if p can be chosen freely. A TRS is overlay iff all critical pairs of the TRS are due to root overlaps, i.e., there are no rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$ such that a non-variable proper subterm of ℓ_1 unifies with ℓ_2. A TRS is locally confluent if every critical pair (s, t) is joinable, i.e., there is some u such that $s \to_{\mathcal{R}}^{*} u$ and $t \to_{\mathcal{R}}^{*} u$.

We write $t \to^{!} s$ if both $t \to^{*} s$ and s is in normal form w.r.t. $\to$, i.e., there is no u such that $s \to u$. Strong normalization of $\to$ is denoted by $SN(\to)$, and $SN_{\to}(t)$ denotes that t admits no infinite derivation w.r.t. $\to$. We sometimes write $SN_{\mathcal{R}}(t)$ instead of $SN_{\to_{\mathcal{R}}}(t)$. A DP problem is a pair of two TRSs $(\mathcal{P}, \mathcal{R})$ where $\mathcal{P}$ is a set of dependency pairs encoding recursive calls, and $\mathcal{R}$ is used to evaluate the arguments between two recursive calls. A $(\mathcal{P}, \mathcal{R})$ chain is an infinite derivation of the form $s_1\sigma_1 \to_{\mathcal{P},\varepsilon} t_1\sigma_1 \to_{\mathcal{R}}^{*} s_2\sigma_2 \to_{\mathcal{P},\varepsilon} t_2\sigma_2 \to_{\mathcal{R}}^{*} \ldots$ where each $s_i \to t_i \in \mathcal{P}$. The chain is an innermost chain, iff additionally $t_i\sigma_i \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} s_{i+1}\sigma_{i+1}$ is satisfied for all i. A TRS $\mathcal{R}$ is (innermost) nonterminating iff $SN(\to_{\mathcal{R}})$ $(SN(\xrightarrow{\mathsf{i}}_{\mathcal{R}}))$ does not hold. A DP problem $(\mathcal{P}, \mathcal{R})$ is (innermost) nonterminating iff it admits an (innermost) $(\mathcal{P}, \mathcal{R})$ chain or if $\mathcal{R}$ is (innermost) nonterminating.[1]

Since the paper describes the formalization on an informal level which does not require deep knowledge of Isabelle, we omit an introduction to this proof assistant here. The logic we are using is classical HOL, which is based on simply-typed lambda-calculus, enriched with a simple form of ML-like polymorphism.

3 Forbidden Patterns

This section deals with checking whether a loop is indeed a loop with respect to a particular evaluation strategy: Given a certificate containing a TRS $\mathcal{R}$, a loop and some strategy, our proof checker CeTA can check whether there does indeed exist an infinite $\mathcal{R}$-rewrite sequence which adheres to this strategy.

[1] In the literature (e.g., in [7]) a nonterminating DP problem is also called *infinite*. This is the reason why in IsaFoR this property is defined as *infinite-dpp*.

To support a broad variety of strategies we consider forbidden pattern rewriting, which covers for instance innermost, outermost, and context-sensitive rewriting [9,15]. Hence the formalization of techniques for forbidden pattern strategies has the significant advantage that a wide range of strategies can be treated by the same formalism, so CeTA internally converts all outermost and context-sensitive strategies into forbidden patterns before the certifier for nontermination proofs is invoked, cf. *certify-cert-problem* in `Proof_Checker.thy`. We give a motivating example before recalling some preliminaries on forbidden pattern rewriting.

Example 1. Consider the following applicative TRS which models a buggy implementation of the map function, where ‘ denotes a binary infix application symbol, and : the cons operator. In the recursive call one forgot to invoke tl on xs and hence the TRS does not terminate.

$$\mathsf{map} ‘ f ‘ xs \to \mathsf{if} ‘ (\mathsf{empty} ‘ xs) ‘ \mathsf{nil} ‘ (: ‘ (f ‘ (\mathsf{hd} ‘ xs)) ‘ (\mathsf{map} ‘ f ‘ xs))$$

$$\begin{array}{ccc} \mathsf{hd} ‘ (: ‘ x ‘ xs) \to x & \mathsf{if} ‘ \mathsf{true} ‘ t ‘ e \to t & \mathsf{empty} ‘ (: ‘ x ‘ xs) \to \mathsf{false} \\ \mathsf{tl} ‘ (: ‘ x ‘ xs) \to xs & \mathsf{if} ‘ \mathsf{false} ‘ t ‘ e \to e & \mathsf{empty} ‘ \mathsf{nil} \to \mathsf{true} \end{array}$$

Without strategy there is a loop $\mathsf{map} ‘ f ‘ \mathsf{nil} \to C[\mathsf{map} ‘ f ‘ \mathsf{nil})]$ for $C = \mathsf{if} ‘ (\mathsf{empty} ‘ \mathsf{nil}) ‘ \mathsf{nil} ‘ (: ‘ (f ‘ (\mathsf{hd} ‘ \mathsf{nil})) ‘ \Box)$ which definitely does not show the real problem of map to the user: the loop ignores the common evaluation strategy for if which disallows reductions in the *then* and *else* branches. Note that due to the applicative setting this desired behavior is not expressible by a context-sensitive strategy, but it can be modeled by a forbidden strategy, as shown in Example 3.

3.1 Background

Using forbidden pattern strategies one can specify that the position of any redex may not be below (or above) certain patterns. In this way one can express outermost (or innermost) strategies. We consider the following extended definition of a forbidden pattern which allows for patterns with location R. This admits to also express strategies like leftmost-outermost with special treatment for if.

Definition 2. *A* forbidden pattern *is a triple* (ℓ, o, λ) *for a term* ℓ*, position* $o \in \mathcal{P}\mathsf{os}(\ell)$*, and* $\lambda \in \{\mathrm{H}, \mathrm{A}, \mathrm{B}, \mathrm{R}\}$*. For a set* Π *of forbidden patterns the relation* $\xrightarrow{\Pi}$ *is defined by* $t \xrightarrow{\Pi}_p s$ *iff* $t \to_p s$ *and there is no pattern* $(\ell, o, \lambda) \in \Pi$ *such that there exist a position* $o' \in \mathcal{P}\mathsf{os}(t)$*, a substitution* σ *with* $t|_{o'} = \ell\sigma$*, and*

- $p = o'o$ *if* $\lambda = \mathrm{H}$ *(here),*
- $p > o'o$ *if* $\lambda = \mathrm{B}$ *(below), and*
- $p < o'o$ *if* $\lambda = \mathrm{A}$ *(above),*
- p *is right of* o' *if* $\lambda = \mathrm{R}$ *(right of).*

Example 3. For the TRS in Example 1, the forbidden pattern strategy where Π consists of $(\mathsf{if} ‘ b ‘ t ‘ e, p, \lambda)$ for all $p \in \{12, 2\}$ and $\lambda \in \{\mathrm{H}, \mathrm{B}\}$ has the intended effect that reductions in the *then* and *else* branches are not allowed.

A TRS $\mathcal{R}$ is forbidden-pattern nonterminating w.r.t. Π iff $\neg SN(\xrightarrow{\Pi})$, which can be proven via *forbidden pattern loops* (Π-loops). To succinctly describe infinite derivations that are induced by loops we use context-substitutions.

Definition 4 ([22]). *A* context-substitution *is a pair (C, μ) consisting of a context C and a substitution μ. The n-fold application of (C, μ) to a term t, denoted $t(C, \mu)^n$, is inductively defined as $t(C, \mu)^0 = t$ and $t(C, \mu)^{n+1} = C[t(C, \mu)^n \mu]$.*

As an example for context-substitutions, we refer to Figure 1 which illustrates the term $t(C, \mu)^3$.

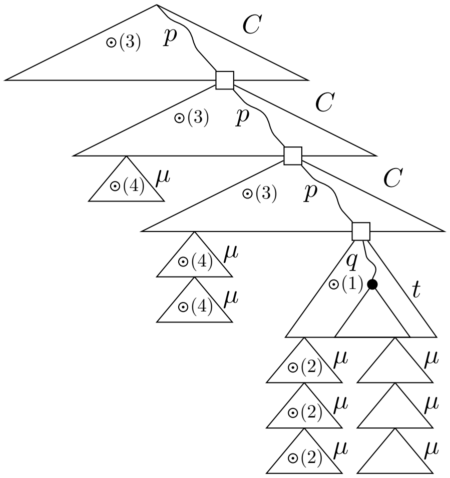

Fig. 1. Redexes left of $p^n q$.

Context-substitutions allow to concisely write the infinite derivation induced by a loop $t \to^+ C[t\mu]$ as $t = t(C, \mu)^0 \to^+ t(C, \mu)^1 \to^+ \ldots \to^+ t(C, \mu)^n \to^+ \ldots$.

To facilitate the certification of loops under strategies, one needs to analyze its constituting steps. In the remainder of this section we will consider a loop with starting term t, context C and substitution μ with $C|_p = \Box$ of the form

$$t = t(C, \mu)^0 = t_1 \to_{q_1} t_2 \to_{q_2} \cdots \to_{q_m} t_{m+1} = t(C, \mu)^1 \qquad (1)$$

A loop of the form (1) is a *Π-loop* iff the step $t_i(C, \mu)^n \to_{p^n q_i} t_{i+1}(C, \mu)^n$ respects the forbidden pattern strategy induced by Π for all $i \leqslant m$ and all $n \in \mathbb{N}$. For instance, assuming that one of the loop's redexes is $t|_q$ as illustrated in Figure 1, we need to know whether this position remains a redex w.r.t. to the strategy, no matter how many contexts and substitutions are applied around t.

The problem of whether a loop constitutes a Π-loop is decidable. To this end, the following notions from innermost and outermost loops are useful.

Definition 5 ([20], [22]). *A* matching problem *is a pair $(u \gtrdot \ell, \mu)$. It is* solvable *iff there are n and σ such that $u\mu^n = \ell\sigma$. An* extended matching problem *is a tuple $(D \gtrdot \ell, C, t, \mathcal{M}, \mu)$ where $\mathcal{M} = \{s_1 \gtrdot \ell_1, \ldots, s_n \gtrdot \ell_n\}$. It is* solvable *iff there are m, k, σ, such that $D[t(C, \mu)^m]\mu^k = \ell\sigma$ and $s_i\mu^k = \ell_i\sigma$ for all i. If $\mathcal{M} = \varnothing$, we just omit it.*

A set of (extended) matching problems is solvable iff some element is solvable. Given a loop, in order to decide whether it indeed constitutes a Π-loop one computes a set of (extended) matching problems which has no solution if and only if the loop is indeed a Π-loop.

3.2 From Forbidden Pattern Loops to Matching Problems

A rewrite step is a Π-step iff it adheres to every single pattern $\pi \in \Pi$. In fact a loop (1) is a Π-loop if and only if the following key property holds for all choices of $\pi \in \Pi$, $t = t_i$, $t' = t_{i+1}$ and $q = q_i$ where $1 \leqslant i < m$ [24]:

Property 6. For a forbidden pattern $\pi = (\ell, o, \lambda)$ and $t \to_q t'$ all reductions $t(C, \mu)^n \to_{p^n q} t'(C, \mu)^n$ are allowed with respect to π, i.e., there are no n, o', and σ such that $t(C,\mu)^n|_{o'} = \ell\sigma$ and $p^n q = o'o$ if $\lambda = \mathrm{H}$, $p^n q < o'o$ if $\lambda = \mathrm{A}$, $p^n q > o'o$ if $\lambda = \mathrm{B}$, and $p^n q$ is right of o' if $\lambda = \mathrm{R}$.

This property can be decided by a case analysis on λ, defining suitable sets of (extended) matching problems for each case. In the following paragraphs we give these sets for patterns of type $(\cdot,\cdot,\mathrm{R})$ and $(\cdot,\cdot,\mathrm{B})$. The other cases are similar, details can be found in the formalization.

Forbidden Patterns of Type $(\cdot,\cdot,\mathbf{R})$ For patterns $\pi = (\ell, o, \mathrm{R})$, it has to be checked whether $p^n q$ occurs to the right of o'. There are four possibilities, as illustrated in Figure 1: (1) o' ends in t, (2) o' ends in a term $t\mu^k$, (3) o' ends in a position of C^k, or otherwise (4) o' ends in a position of $C^k\mu^{k-1}$, for some $k \leqslant n$. Let $\mathcal{W}(t) = \bigcup_{k\in\mathbb{N}} \mathcal{V}(t\mu^k)$ denote the set of variables introduced by the substitution μ when applied iteratively. Then each case can be covered by a set of matching problems as follows:

Definition 7. *Let $\mathcal{M}_{\mathrm{R},\pi}$ denote the union of the following four sets:*

$$\mathcal{M}_{\mathrm{R},1} = \{(u \gtrdot \ell, \mu) \mid q' \in \mathcal{P}\mathsf{os}(t),\ q' \text{ is left of } q, \text{ and } u = t|_{q'}\}$$
$$\mathcal{M}_{\mathrm{R},2} = \{(u \gtrdot \ell, \mu) \mid q' \in \mathcal{P}\mathsf{os}(t),\ q' \text{ is left of } q,\ x \in \mathcal{W}(t|_{q'}), \text{ and } u \trianglelefteq x\mu\}$$
$$\mathcal{M}_{\mathrm{R},3} = \{(u \gtrdot \ell, \mu) \mid p' \in \mathcal{P}\mathsf{os}(C),\ p' \text{ is left of } p, \text{ and } u = C|_{p'}\}$$
$$\mathcal{M}_{\mathrm{R},4} = \{(u \gtrdot \ell, \mu) \mid p' \in \mathcal{P}\mathsf{os}(C),\ p' \text{ is left of } p,\ x \in \mathcal{W}(C|_{p'}), \text{ and } u \trianglelefteq x\mu\}$$

For the formalization of patterns $(\cdot,\cdot,\mathrm{R})$, we first had to incorporate support for the *left-of* relation on positions. However, the most effort was spent on the case analysis, i.e., an induction proof showing that any position in a context-substitution $t(C,\mu)^n$ fits into one of the four cases.

Forbidden Patterns of Type $(\cdot,\cdot,\mathbf{B})$ For patterns $\pi = (\ell, o, \mathrm{B})$ the position $o'o$ has to be strictly above the redex, i.e., $p^n q > o'o$. Here two cases can be distinguished: $o'o$ may end in t, so $o'o \geqslant p^n$, or it may end in some occurrence of C, so $o'o < p^n$ (similar to cases (1) and (3) in Figure 1).

In case of the former, $o'o$ has finitely many possibilities to hit a position in t above q. Thus, this case reduces to finitely many $(\cdot,\cdot,\mathrm{H})$ cases.

In the latter case, $o'o$ is a non-hole position of C^n, i.e., $p^n > o'o$ (and hence $p > \varepsilon$). We consider all possibilities for non-empty subcontexts D, and compute a number n_0 such that it suffices to consider the term $t(C,\mu)^{n_0}$ in order to account for all loop iterations.[2] A detailed analysis of these two cases leads to the following sets of matching problems $\mathcal{M}_{\mathrm{B}}$ and $\mathcal{E}_{\mathrm{B}}$:

Definition 8. *The (extended) matching problems $\mathcal{M}_{\mathrm{B},\pi} = \mathcal{M}_{\mathrm{B},1} \cup \mathcal{E}_{\mathrm{B},2}$ are*

$$\mathcal{M}_{\mathrm{B},1} = \bigcup_{\bar{q}<q} \mathcal{M}_{\mathrm{H},(\ell,\bar{q},\mathrm{H})}$$
$$\mathcal{E}_{\mathrm{B},2} = \{(D \gtrdot \ell, C\mu, t(C,\mu)^{n_0}\mu, \mu) \mid \Box \lhd D \trianglelefteq C,\ D|_{p''} = \Box,\ p''p^{n_0} > o\}$$

[2] More precisely, n_0 can be set to 0 if $p = \varepsilon$ and to $\lceil \frac{|o| \dot{-} |q|}{|p|} \rceil$ otherwise.

where $\mathcal{M}_{\mathrm{H},(\ell,\bar{q},\mathrm{H})}$ refers to the H *matching problem for t, q, and (C,μ), and n_0 is, dependent on p'', the minimal number satisfying $|p''| + n_0|p| > |o|$.*

Unsolvability of the respective sets of (extended) matching problems is a sufficient and necessary condition for Property 6:

Theorem 9 ([24]). *Let $t \rightarrow_q t'$ and let (C,μ) be a context-substitution such that $C|_p = \square$. All reductions $t(C,\mu)^n \rightarrow_{p^n q} t'(C,\mu)^n$ are allowed with respect to the pattern $\pi = (\ell, o, \lambda)$ if and only if $\mathcal{M}_{\lambda,\pi}$ is not solvable.*

As to be expected from the technical definitions, the soundness and completeness results for the respective cases required a considerable amount of reasoning about contexts and positions. We preferred contexts over positions whenever possible: position reasoning tends to be tedious because one always needs to ensure that they are valid in the term where they are to be used. For instance, IsaFoR internally represents forbidden patterns as triples $(\ell[\cdot]_o, \ell|_o, \lambda)$ rather than (ℓ, o, λ) to avoid the obvious side condition $o \in \mathcal{P}\mathsf{os}(\ell)$. The amount of bureaucracy on valid positions required throughout the formalization was nevertheless substantial. Apart from this, the proofs for all cases could be formalized along the lines of the paper proof. For the case of B patterns the results crucially rely on the new solving procedure for extended matching problems.

3.3 Deciding Solvability of Extended Matching Problems

Solvability of (extended) matching problems is known to be decidable [20,22], and in [23] we already formalized and simplified the decision procedure for non-extended matching problems. In the remainder of this section we present our algorithm to decide solvability of extended matching problems—these problems originate from the outermost loop checking procedure and are also required in the case of forbidden patterns, cf. Definition 8.

As in [23], our proofs deviate from the paper proofs considerably and result in a simplified decision procedure which we also integrated in termination tools. For example, in AProVE we have been able to delete some sub-algorithms (180 lines) and replace them by a single line of code.

The decision procedure in [22] works in three phases: first, any extended matching problem is simplified to solved form; second, from the simplified matching problem a set of (extended) identity problems is generated, and finally, solvability of the identity problems is decided. We followed this general structure in the formalization, and only report on the first and the third phase, since the second phase was straightforward.

The algorithm for the first phase consists of a set of strongly normalizing inference rules. It contains rules for decomposition and symbol clash as in a standard matching algorithm[3], but also incorporates rules to apply a (context-)substitution in cases where a standard matching algorithm would fail.

[3] Rules (i)-(iv) in [22, Def. 5], which are omitted here for brevity.

Definition 10 ([22, Def. 5]). *Let $MP = (D \gtrdot \ell_0, C, t, \mathcal{M}, \mu)$ be an extended matching problem where $\mathcal{M} = \{s_1 \gtrdot \ell_1, \ldots, s_m \gtrdot \ell_m\}$ and $C \neq \square$. Then MP is in* solved form *iff each ℓ_i is a variable. Let $\mathcal{V}_{\mathsf{incr},\mu} = \{x \in \mathcal{V} \mid \exists n : x\mu^n \notin \mathcal{V}\}$ be the set of increasing variables.*

We define a relation $\Rightarrow$ which simplifies extended matching problems that are not in solved form, so suppose $\ell_j = f(\ell'_1, \ldots, \ell'_{m'})$.

(v) $MP \Rightarrow \bot$ *if* $s_j \in \mathcal{V} \backslash \mathcal{V}_{\mathsf{incr},\mu}$
(vi) $MP \Rightarrow (D\mu \gtrdot \ell_0, C\mu, t\mu, \{s_i\mu \gtrdot \ell_i \mid 1 \leqslant i \leqslant m\}, \mu)$ *if* $s_j \in \mathcal{V}_{\mathsf{incr},\mu}$
(vii) $MP \Rightarrow \top$ *if* $j = 0$*,* $D = \square$*, and* $(\mathcal{M} \cup \{t \gtrdot \ell_0\}, \mu)$ *is solvable*
(viii) $MP \Rightarrow (C \gtrdot \ell_0, C\mu, t\mu, \mathcal{M}, \mu)$ *if* $j = 0$*,* $D = \square$*, and* $(\mathcal{M} \cup \{t \gtrdot \ell_0\}, \mu)$ *is not solvable*

As in [23], where we formalized the inference rules for simplifying non-extended matching problems, we implemented these rules directly as a function *simplify-emp-main* using Isabelle's function package [13]. In this way, we did not have to formalize confluence of $\Rightarrow$.

Note that for this function one faces the problem of getting it terminating and efficient at the same time: if one has to recompute $\mathcal{V}_{\mathsf{incr},\mu}$ in every iteration, the function becomes inefficient; on the other hand, if one passes $\mathcal{V}_{\mathsf{incr},\mu}$ using an additional parameter (e.g., V_i) then the function is not necessarily terminating as it is not guaranteed that V_i is indeed instantiated by $\mathcal{V}_{\mathsf{incr},\mu}$. To see this, suppose the simplification algorithm is invoked on the problem $(D \gtrdot \ell, C, t, \{x \gtrdot \mathsf{c}\}, \mu)$ where μ is the empty substitution but V_i a set containing x. Then an application of Rule (vi) immediately leads to a recursive call with the same arguments.

To solve this problem, in [23] it was proposed to write two functions: The main soundness result is proven for a terminating but inefficient one where $\mathcal{V}_{\mathsf{incr},\mu}$ gets recomputed in every recursive call. A second, possibly nonterminating function has V_i as additional argument and is proven to be equivalent to the first function if invoked with the right arguments, i.e., in this case with $V_i = \mathcal{V}_{\mathsf{incr},\mu}$.

Although this solution leads to an efficient and sound implementation, it imposes quite some overhead. First, one has to write the simplification algorithm twice, and second one has to perform an equivalence proof of the two functions.

Therefore we propose a different solution for *simplify-emp-main*. The simple idea is to pass the pair (μ, V_i) as an argument to *simplify-emp-main*, where this pair is encapsulated in a new type with the invariant that $V_i = \mathcal{V}_{\mathsf{incr},\mu}$. Thus, in the implementation one just has to provide selectors from the new type to both μ and V_i, where it now suffices to write only one implementation of *simplify-emp-main*. Moreover, the whole quotient construction—creation of the new type, writing the selectors, reasoning about this new type—can conveniently be done via the lifting and transfer-package of Isabelle [11]. Note that in the meantime we also rewrote the simplification algorithm for matching problems in [23] using the same idea, again by using lifting and transfer.

For the third phase where (extended) identity problems are to be solved, we could of course reuse the algorithm for non-extended identity problems that has been developed in [23]. However, we did not stick to the complicated algorithm of [22] for extended identity problems, since it requires several auxiliary algorithms and the soundness proofs are difficult or tedious to formalize. (The whole description takes 3.5 pages in [22] where these pages do not even cover all proofs.) Instead, we developed a new, partial algorithm which is easy to implement and easy to formalize. In detail, we show that all extended identity problems that are constructed for forbidden patterns via *simplify-emp-main* belong to a special class of extended identity problems where the context within such a problem is large in comparison to the other terms. This class of problems can easily be translated into non-extended identity problems via the following mini-algorithm: an extended identity problem $(D \approx s, \mu, C, t)$ is solvable iff the identity problem $(D[t] \approx s, \mu)$ is solvable, provided there is some i such that $C \rhd s\mu^i$. For more details on (extended) identity problems and our new proofs we refer to [22] and lemmas *eident-prob-to-ident-prob* and *simplify-emp-main-large-C* within the theory `Outermost_Loops.thy`.

4 Nonlooping Nontermination

While in the previous section we restricted ourselves to loops (though for every forbidden pattern strategy), we now aim at possibly nonlooping nonterminating TRSs, but only consider innermost strategies. More precisely, we consider the variant of innermost rewriting which corresponds to $\xrightarrow{\Pi}$ where $\Pi = \{(\ell, \varepsilon, \mathrm{A}) \mid \ell \in \mathcal{Q}\}$ for some set of terms $\mathcal{Q}$. The corresponding rewrite relation is *qrstep* within IsaFoR, and it generalizes rewriting without strategy ($\mathcal{Q} = \varnothing$) and innermost rewriting ($\mathcal{Q} = \{\ell \mid \ell \to r \in \mathcal{R}\}$). To ease the presentation, in the paper we just consider the special cases $\to_{\mathcal{R}}$ and $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ in the following. In total, we discuss three different techniques which can be used to disprove termination for nonlooping nonterminating TRSs. One disregards the strategy completely (§ 4.1), one performs rewrite steps which may violate the strategy (§ 4.2), and one directly constructs infinite possibly nonlooping derivations (§ 4.3).

4.1 Switching between Innermost Termination and Termination

Example 11. Let $\mathcal{R}'$ be a confluent overlay TRS which encodes a Turing machine $\mathcal{A}$ via innermost rewriting. We assume that the computation starts in a constant $\mathsf{tm}_{\mathsf{init}}$ which represents the initial configuration of $\mathcal{A}$. Now consider the TRS $\mathcal{R} = \mathcal{R}' \cup \{\mathsf{run\text{-}again}(x) \to \mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}})\}$ where $\mathsf{run\text{-}again}$ is some fresh symbol.

Obviously, $\mathcal{R}$ is not innermost terminating: if $\mathcal{A}$ terminates in some final configuration represented by a term t, then $\mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}}) \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} \mathsf{run\text{-}again}(t) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}})$ is an innermost loop. Otherwise, there is an infinite evaluation of $\mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}})$ when trying to rewrite the argument $\mathsf{tm}_{\mathsf{init}}$ to a normal form.

Observe that in the first case, the derivation may be long and thus hard to find, e.g., $\mathcal{A}$ may compute the Ackermann function; and in the latter case, there might be no looping derivation at all.

However, disproving termination of $\mathcal{R}$ is simple when disregarding the strategy: the loop $\mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}}) \to_{\mathcal{R}} \mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}})$ is easily detected. Hence, for nontermination analysis one tries to get rid of strategy restrictions, and indeed there are known criteria where $SN(\xrightarrow{\mathsf{i}}_{\mathcal{R}})$ and $SN(\to_{\mathcal{R}})$ coincide: for example, locally confluent overlay TRSs fall into this class [8]. Thus, the simple loop above constitutes a valid innermost nontermination proof.

We formalized the criterion of [8], though we did not follow the original proof structure, but developed a simpler proof via dependency pairs [1]. To this end, we first integrated a similar theorem for DP problems, as it is utilized in AProVE, cf. *switch-to-innermost-proc* in `Innermost_Switch.thy`.

Theorem 12. *Let $\mathcal{P}$ and $\mathcal{R}$ be TRSs such that $\mathcal{R}$ is locally confluent and such that there is no overlap between $\mathcal{P}$ and $\mathcal{R}$. Then any $(\mathcal{P}, \mathcal{R})$ chain shows the existence of some innermost $(\mathcal{P}, \mathcal{R})$ chain.*

Theorem 12 can not only be used on its own—to switch from innermost termination to termination for DP problems—but it can also be utilized to derive Gramlich's result to switch from innermost termination to termination for TRSs.

Theorem 13 ([8]). *Let $\mathcal{R}$ be some finite TRS, let there be infinitely many symbols. If $\mathcal{R}$ is locally confluent and overlay, then $\neg SN(\to_{\mathcal{R}}) \implies \neg SN(\xrightarrow{\mathsf{i}}_{\mathcal{R}})$.*

Proof. Let $\mathcal{P}$ be the set of dependency pairs of $\mathcal{R}$. If $\mathcal{R}$ is not terminating, then by soundness of dependency pairs there must be some $(\mathcal{P}, \mathcal{R})$ chain. By Theorem 12 we conclude that there also is some innermost $(\mathcal{P}, \mathcal{R})$ chain: $\mathcal{R}$ is locally confluent by assumption and there is no overlap between $\mathcal{P}$ and $\mathcal{R}$ since $\mathcal{R}$ is an overlay TRS. Finally, by completeness of dependency pairs we conclude from the innermost chain that $\mathcal{R}$ must be innermost nonterminating. □

The formalization of this proof was straightforward: IsaFoR already contained the required results on critical pairs, confluence, and dependency pairs [18,21], cf. *switch-to-innermost-locally-confluent-overlay-finite* in `Innermost_Switch.thy`.

The formalization also reveals side conditions which one never finds in paper proofs: Finiteness of $\mathcal{R}$ and an unbounded supply of function symbols are taken for granted, but are crucial to construct fresh function symbols (fresh symbols are required in order to build the set of dependency pairs). With more bureaucracy, one would be able to drop the condition that $\mathcal{R}$ is finite—by arguing that in an infinite reduction only countably many symbols can occur, and by implementing Hilbert's hotel one can always construct enough fresh symbols—but since for certification we are only interested in finite TRSs, we did not spend this additional effort.

In order to guarantee local confluence we had to provide new means for checking joinability. Whereas in [18] the main algorithm was a comparison of normal forms of s and t, this is no longer the best solution in our setting, since $\mathcal{R}$ is usually nonterminating. To this end, we now offer a breadth-first-search algorithm to check joinability. The certificate just has to set a limit on the search depth which ensures termination of the algorithm.

In total, we can now easily certify innermost nontermination proofs like the one for Example 11: the certificate just has to contain the looping derivation $\mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}}) \to_{\mathcal{R}} \mathsf{run\text{-}again}(\mathsf{tm}_{\mathsf{init}})$ and an indication in how many steps each critical pair of $\mathcal{R}$ can be joined.

4.2 Rewriting and Narrowing Dependency Pairs

In this section we consider two techniques of [7] that allow to ignore the strategy for one step. Given a DP problem $(\mathcal{P}, \mathcal{R})$, they replace one of the pairs $s \to t$ in $\mathcal{P}$ by new ones which result from rewriting or narrowing $s \to t$.

One advantage over the result from the previous subsection is that we only need unique normal forms for the *usable rules* while previously we had to consider the whole TRS. Here, the usable rules of a term t are any subset $\mathcal{U}(t)$ of $\mathcal{R}$ such that whenever $t\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s$ for some σ which instantiates all variables by normal forms, then in this derivation all applied rules must be from $\mathcal{U}(t)$. There are various estimations of usable rules where the simplest one is provided in [1]. The following theorem already generalizes [7, Thm. 31] which requires non-overlappingness instead of unique normal forms.

Theorem 14. *Let $(\mathcal{P} \uplus \{s \to t\}, \mathcal{R})$ be a DP problem and suppose $t \to_{\mathcal{R},p} t'$ with rule $\ell \to r \in \mathcal{R}$ and substitution μ. If for $\mathcal{U} = \mathcal{U}(t|_p)$ the rewrite relation $\xrightarrow{\mathsf{i}}_{\mathcal{U}}$ has unique normal forms and there are only trivial critical pairs between $\ell \to r$ and $\mathcal{U}$ then the following holds: if $(\mathcal{P} \uplus \{s \to t'\}, \mathcal{R})$ is innermost nonterminating then $(\mathcal{P} \uplus \{s \to t\}, \mathcal{R})$ is also innermost nonterminating.*

In the formalization we closely followed the original paper proof where we were able to slightly relax the preconditions: it is sufficient to consider the usable rules with respect to all arguments of $t|_p$ instead of $t|_p$ itself. To check that $\mathcal{U}$ has unique normal forms we use the following easy but sufficient criterion: if all critical pairs of $\mathcal{U}$ at the root level are trivial then $\xrightarrow{\mathsf{i}}_{\mathcal{U}}$ is confluent and thus has unique normal forms. The following TRS can be shown innermost nonterminating via Theorem 14, but it requires the more relaxed preconditions.

Example 15. Consider the TRS $\mathcal{R}$ consisting of $\mathcal{R}'$ of Example 11 and the rules:

$$\mathsf{c}(x, y) \to x \qquad \mathsf{c}(x, y) \to y \qquad \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{c}(\mathsf{a}, \mathsf{tm}_{\mathsf{init}}))$$

Note that the result from the previous subsection is not applicable, since the system is not locally confluent. However, since $\mathcal{U}(\mathsf{a}) = \varnothing$ and $\mathcal{U}(\mathsf{tm}_{\mathsf{init}}) = \mathcal{R}'$ is confluent, we can rewrite the dependency pair $\mathsf{f}^\sharp(\mathsf{a}) \to \mathsf{f}^\sharp(\mathsf{c}(\mathsf{a}, \mathsf{tm}_{\mathsf{init}}))$ to $\mathsf{f}^\sharp(\mathsf{a}) \to \mathsf{f}^\sharp(\mathsf{a})$ and obtain an obvious loop.

To certify such a nontermination proof, one only has to provide the rewrite step that is performed and a nontermination proof for the modified problem. All preconditions are automatically checked by CeTA.

The second technique considers narrowing of dependency pairs, where a rule $s \to t \in \mathcal{P}$ is first instantiated to $s\sigma \to t\sigma$ and subsequently $t\sigma$ gets rewritten to

u, yielding a new rule $s\sigma \to u$. Since instantiation is obviously correct for nontermination analysis, completeness of narrowing is a straightforward consequence of the completeness result for rewriting, cf. `Rewriting.thy`, `Instantiation.thy`, and `Narrowing.thy`.

4.3 Nonterminating Derivations

To finally detect nontermination, one requires a technique which actually finds infinite derivations. As stated before, one can consider loops $t \to^+ C[t\mu]$, however, there are also techniques which are able to detect a larger class of nonterminating derivations [5,17] which are both available in CeTA.

The idea in [5] is to derive *pattern rules* of the form $s\,\sigma^n\,\tau \hookrightarrow t\,\delta^n\,\mu$ which state that for each n there is a rewrite sequence $s\sigma^n\tau \to^+ t\delta^n\mu$. To this end, there are several inference rules which allow to derive pattern rules, and there is a sufficient criterion when a pattern rule implies nontermination.

Example 16. Consider the following nonterminating TRS.

$$\mathsf{s}(x) > \mathsf{0} \to \mathsf{true} \qquad\qquad \mathsf{0} > y \to \mathsf{false}$$
$$\mathsf{s}(x) > \mathsf{s}(y) \to x > y \qquad\qquad \mathsf{f}(\mathsf{true}, x, y) \to \mathsf{f}(x > y, \mathsf{s}(x), \mathsf{s}(y))$$

It is nonlooping, as in the infinite derivation

$$\begin{aligned} &\mathsf{f}(\mathsf{true}, \mathsf{s}^2(\mathsf{0}), \mathsf{s}^1(\mathsf{0})) \to \mathsf{f}(\mathsf{s}^2(\mathsf{0}) > \mathsf{s}^1(\mathsf{0}), \mathsf{s}^3(\mathsf{0}), \mathsf{s}^2(\mathsf{0}))\\ \to^2\ &\mathsf{f}(\mathsf{true}, \mathsf{s}^3(\mathsf{0}), \mathsf{s}^2(\mathsf{0})) \to \mathsf{f}(\mathsf{s}^3(\mathsf{0}) > \mathsf{s}^2(\mathsf{0}), \mathsf{s}^4(\mathsf{0}), \mathsf{s}^3(\mathsf{0}))\\ \to^3\ &\mathsf{f}(\mathsf{true}, \mathsf{s}^4(\mathsf{0}), \mathsf{s}^3(\mathsf{0})) \to \dots \end{aligned}$$

it takes more and more steps to rewrite $\mathsf{s}^{n+1}(\mathsf{0}) > \mathsf{s}^n(\mathsf{0})$ to true when n is increased. However, using the inference rules, one can first derive the pattern rule $(\mathsf{s}(x) > \mathsf{s}(y))\,\{x/\mathsf{s}(x), y/\mathsf{s}(y)\}^n\,\{x/\mathsf{s}(x), y/\mathsf{0}\} \hookrightarrow \mathsf{true}\,\varnothing^n\,\varnothing$ which states that it is possible to rewrite each term $\mathsf{s}^{n+2}(x) > \mathsf{s}^{n+1}(\mathsf{0})$ to true ($\varnothing$ denotes the empty substitution). And afterwards, it is easy to combine this pattern rule with the rule for f to detect nontermination, again using the methods of [5].

To be able to certify this kind of nontermination proofs, in `Nonloop.thy` we first proved correctness of all inference rules on an abstract level, e.g., where substitutions are modeled as functions from variables to terms. In order to check concrete proofs, in `Nonloop_Impl.thy` we then introduced a datatype to represent proofs, i.e., sequences of inference steps, where also the type of substitutions was changed from the abstract type to a list based representation.

Using this approach, most of the paper proofs have been easily integrated into Isabelle. We here only report on some issues we had to solve during the formalization. To this end, consider the following two inference rules of [5].

$$\frac{s\,\varnothing^n\,\varnothing \hookrightarrow t\,\varnothing^n\,\varnothing}{s\,\sigma^n\,\varnothing \hookrightarrow t[z]_p\,(\sigma \cup \{z/t[z]_p\})^n\,\{z/t|_p\}}\text{if } p \in \mathcal{P}\mathsf{os}(t),\ s = t|_p\sigma,\ z \text{ is fresh} \quad \text{(III)}$$

$$\frac{s\,\sigma_s^n\,\mu_s \hookrightarrow t\,\sigma_t^n\,\mu_t}{s\,(\sigma_s\rho)^n\,\mu_s \hookrightarrow t\,(\sigma_t\rho)^n\,\mu_t}\text{if } \delta\rho = \rho\delta \text{ for each } \delta \in \{\sigma_s, \mu_s, \sigma_t, \mu_t\} \quad \text{(VII)}$$

One of the small problems we encountered is the underspecification in Rule (III): the condition "z is fresh" does not contain the information w.r.t. which other variables z has to be fresh—in the formalization this is clarified, namely $\mathcal{V}(s) \cup \mathcal{V}(t) \cup \bigcup_{x:\sigma(x)\neq x}(\{x\} \cup \mathcal{V}(\sigma(x)))$.

Moreover, there have been several operations on substitutions which first had to be defined, e.g. for domain renamings [5, Def. 3], one defines substitutions like $\{x\rho/s\rho \mid x/s \in \sigma\}$ where ρ has some further properties. Before showing properties of this substitution, in the formalization we first had to prove that this substitution is well-defined, i.e., that the properties of ρ ensure that $x\rho$ is always a variable, and that there are no conflicting assignments.

Further operations on substitutions became necessary for certification. For example, in Rule (VII) one has to check equality of substitutions. Here, it turned out that checking equality of the lists which represent the substitutions was not sufficient, as some correct proofs have been rejected by our certifier, e.g., since $[(x,t),(y,s)] \neq [(y,s),(x,t),(x,t)]$, but both lists represent the same substitution $\{x/t, y/s\}$. Instead, we had to implement a function *subst-eq* which decides whether two substitutions which are represented by lists are identical.

We finally remark on an extension of the original approach that was required in the formalization: while the technique in [5] is presented on the level of TRSs, the implementation in AProVE also applies the method on DP problems, where in the inference rules one has to distinguish between $\mathcal{P}$- and $\mathcal{R}$-steps. Moreover, AProVE also uses the following inference rule, which was not described in [5].

$$\frac{s\,\sigma_s^n\,\mu_s \hookrightarrow t\,\sigma_t^n\,\mu_t}{s\sigma_s^k\,\sigma_s^n\,\mu_s \hookrightarrow t\sigma_t^k\,\sigma_t^n\,\mu_t} \tag{X}$$

All these extensions have been integrated in IsaFoR and CeTA.

The technique of [17] is quite similar to [5] in the sense that there are also derivation patterns which can be derived via some inference rules, until some pattern is detected which immediately implies nontermination. In fact, [5] is an extension of [17] as the latter only considers string rewrite systems, i.e., TRSs with only unary function symbols. But since it is currently unknown whether [5] can fully simulate [17], we also formalized the technique of [17] directly, which was a relatively easy task: since everything in [17] works on strings, there was no tedious reasoning on substitutions and renamings of variables required, cf. `Nonloop_SRS.thy`.

For certification we require the full inference tree that derives the final pattern, where in each inference rule all parameters have to be specified. For example, for (III) we explicitly require σ, p, and z; and for (VII) the substitution ρ has to be provided. Moreover, for pretty-printing and early error detection we require that every derived pattern is explicitly stated within the certificate.

5 Experiments and Partial Nontermination Proofs

We tested our certifier using the TRSs from the termination problem database (TPDB 8.0.7). To be more precise, we considered all 596 first-order TRSs where at least one tool in 2013 has generated a nontermination proof. In our experiments, we tested the following termination tools which all print their proofs in a structured proof format (CPF).

- AProVE'13 and TTT2'13 are the versions AProVE and TTT2 which participated in the certified category of the termination competition in 2013. Both tools are restricted to nontermination techniques of [23].
- AProVE'14 is the current version of AProVE. It can even apply nontermination techniques that are not supported by CeTA.
- TTT2'14 is the current version of TTT2.

	AProVE'13	AProVE'14	TTT2'13	TTT2'14
# successful nontermination proofs	276	575	221	417
# certified proofs	276	563	221	417
# partially certified proofs	–	12	–	–

Table 1. Experimental data.

Table 1 clearly shows the significance of our formalizations: we doubled the number of certifiable nontermination proofs for AProVE, and can now certify 98 % of the generated proofs.

Since AProVE'13, TTT2'13, and TTT2'14 use only techniques supported by CeTA, it comes as no surprise that all these proofs were certified. In contrast, 12 proofs by AProVE'14 were refused as the applied nontermination techniques are not available in CeTA, e.g., proofs for equational rewrite systems (modulo AC).

To still increase the reliability for these proofs, we added support for partial proofs in CeTA. To be more precise, we added a proof technique called "unknown proof" to CPF which logically states that the certifier may assume the implication $\neg SN(a_1) \wedge \cdots \wedge \neg SN(a_n) \implies \neg SN(a_0)$ where each a_i may be some arbitrary binary relation, including textual descriptions like "equational rewrite relation of ..." which are not formally specified. As a consequence, every technique that is not supported by CeTA can be exported as an unknown proof, and then CeTA can still check all the proofs for the subgoals $\neg SN(a_i)$ with $i > 0$.

Using partial certification, CeTA can check in average 70 % of the proof steps within each of the 12 partial proofs. Interestingly, due to the partial certification capabilities of CeTA, we could even spot and fix one real soundness bug within AProVE. In one example a terminating TRS $\mathcal{R}_1$ was transformed into a nonterminating TRS $\mathcal{R}_2$ although it was claimed that the termination behavior of $\mathcal{R}_1$ and $\mathcal{R}_2$ is equivalent. Since AProVE was not able to finally disprove termination of $\mathcal{R}_2$—and hence there was no complete nontermination proof of $\mathcal{R}_1$—this bug was only discovered due to partial certification, where even for incomplete proofs every single nontermination technique could be checked by CeTA.

To support partial certification in CeTA, major restructuring was required. Previously, there was a hierarchical structure of nontermination proofs where the hierarchy was given by the input: nontermination proofs for DP problems have been a leaf, proofs for TRSs have been been the next layer, and proofs for relative termination have been at the top of the hierarchy. However, now for every input there is the "unknown proof" which may contain subproofs for all other inputs. Therefore, the proof types for every input are modeled via one large mutual recursive datatype (it is the datatype definition *. . . -nontermination-proof* at the beginning of `Check_Nontermination.thy`), which takes considerably more time to process by Isabelle than the hierarchical sequence of non-mutual recursive datatypes that we had before. Similarly, also all functions and proofs for the overall certification procedure had to be defined and proven simultaneously for all inputs. Whereas most of this adaptation was straightforward, we also encountered problems, that some packages in Isabelle do not support mutual recursion. For example, in order to define our parser for CPF, we first had to add support for mutual recursion to the partial functions package of [14]. We refer to [19] for further details on this extension.

In order to obtain input examples for CeTA's forbidden pattern loop check, we integrated support for forbidden pattern loops into $\mathsf{T_TT_2}$. More precisely, we added a forbidden pattern loop check to the already present `unfold` strategy which searches for loops. To that end, we exported IsaFoR's loop checking procedure to OCaml using Isabelle's code generator. Though interfacing IsaFoR's data structures required some overhead, this proved to be a fast way to integrate a reliable implementation in $\mathsf{T_TT_2}$. Support of forbidden pattern loops allows $\mathsf{T_TT_2}$'14 to show nontermination of all those TRSs in our test set of 596 problems that feature an innermost, outermost, or context-sensitive strategy (197 problems in total), as well as Example 3. In total, by just integrating CeTA's forbidden pattern loop check, we could nearly double the number of nontermination proofs of $\mathsf{T_TT_2}$'13: from 221 to 417, cf. Table 1.

6 Conclusion

In summary, we formalized several new nontermination techniques which cover nonlooping derivations and looping derivations under strategies. In total this formalization increased the size of IsaFoR by around 10k lines. Due to our work, CeTA is now able to certify the vast majority of nontermination proofs that are generated by automated tools for TRSs.

Acknowledgments The authors are listed in alphabetical order regardless of individual contributions or seniority.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

2. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
3. F. Blanqui and A. Koprowski. CoLoR: A Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 4:827–859, 2011.
4. E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Automated certified proofs with CiME3. In *Proc. RTA '11*, volume 10 of *LIPIcs*, pages 21–30, 2011.
5. F. Emmes, T. Enger, and J. Giesl. Proving non-looping non-termination automatically. In *Proc. IJCAR '12*, volume 7364 of *LNAI*, pages 225–240, 2012.
6. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. In *Proc. IJCAR '06*, volume 4130 of *LNAI*, pages 281–286, 2006.
7. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
8. B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
9. B. Gramlich and F. Schernhammer. Extending context-sensitivity in term rewriting. In *Proc. WRS '09*, volume 15 of *EPTCS*, pages 56–68, 2010.
10. F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In *Proc. FLOPS 2010*, volume 6009 of *LNCS*, pages 103–117, 2010.
11. B. Huffman and O. Kuncar. Lifting and transfer: A modular design for quotients in Isabelle/HOL. In *Proc. CPP '13*, volume 8307 of *LNCS*, pages 131–146, 2013.
12. M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *Proc. RTA '09*, volume 5595 of *LNCS*, pages 295–304, 2009.
13. A. Krauss. Partial and nested recursive function definitions in higher-order logic. *Journal of Automated Reasoning*, 44(4):303–336, 2010.
14. A. Krauss. Recursive definitions of monadic functions. In *Proc. PAR '10*, volume 43 of *EPTCS*, pages 1–13, 2010.
15. S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1:1–61, 1998.
16. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
17. M. Oppelt. Automatische Erkennung von Ableitungsmustern in nichtterminierenden Wortersetzungssystemen. Diploma thesis, HTWK Leipzik, Germany, 2008.
18. C. Sternagel and R. Thiemann. Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In *Proc. RTA '13*, volume 21 of *LIPIcs*, pages 287–302, 2013.
19. R. Thiemann. Mutually recursive partial functions. *Archive of Formal Proofs*, February 2014. http://afp.sf.net/entries/Partial_Function_MR.shtml, Formal proof development.
20. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Deciding innermost loops. In *Proc. RTA '08*, volume 5117 of *LNCS*, pages 366–380, 2008.
21. R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs 2009*, volume 5674 of *LNCS*, pages 452–468, 2009.
22. R. Thiemann and C. Sternagel. Loops under strategies. In *Proc. RTA '09*, volume 5595 of *LNCS*, pages 17–31, 2009.
23. R. Thiemann and C. Sternagel. Certification of nontermination proofs. In *Proc. ITP '12*, volume 7406 of *LNCS*, pages 266–282, 2012.
24. R. Thiemann, C. Sternagel, J. Giesl, and P. Schneider-Kamp. Loops under strategies ... continued. *Proc. IWS '10*, 44:51–65, 2010.