Certifying the Weighted Path Order

René Thiemann
University of Innsbruck, Austria

Jonas Schöpf
University of Innsbruck, Austria

Christian Sternagel
DVT, Innsbruck, Austria

Akihisa Yamada
National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Abstract

The weighted path order (WPO) unifies and extends several termination proving techniques that are known in term rewriting. Consequently, the first tool implementing WPO could prove termination of rewrite systems for which all previous tools failed. However, we should not blindly trust such results, since there might be problems with the implementation or the paper proof of WPO.

In this work, we increase the reliability of these automatically generated proofs. To this end, we first formally prove the properties of WPO in Isabelle/HOL, and then develop a verified algorithm to certify termination proofs that are generated by tools using WPO. We also include support for max-polynomial interpretations, an important ingredient in WPO. Here we establish a connection to an existing verified SMT solver. Moreover, we extend the termination tools NaTT and $\mathsf{T_TT_2}$, so that they can now generate certifiable WPO proofs.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Equational logic and rewriting

Keywords and phrases certification, Isabelle/HOL, reduction order, termination analysis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2020.4

Category Invited Talk

Supplementary Material Details on the experiments are provided at `http://cl-informatik.uibk.ac.at/isafor/experiments/wpo/`. This website also provides links to the formalization.

Funding René Thiemann was supported by the Austrian Science Fund (FWF) project Y757. Jonas Schöpf and Christian Sternagel were supported by FWF project P27502. A part of Akihisa Yamada's contribution was made while he was working in University of Innsbruck under support by FWF projects Y757 and P27502.

Acknowledgements We thank Sarah Winkler for her assistance and discussions on IsaFoR, $\mathsf{T_TT_2}$ and WPO.

1 Introduction

Automatically proving termination of *term rewrite systems (TRSs)* has been an active field of research for half a century. A number of *simplification orders* [13] are classic methods for proving termination, while more general pairs of orders called *reduction pairs* play a central role in the more modern *dependency pair framework* [19].

The *weighted path order (WPO)* was first [51] introduced as a simplification order that unifies and extends classical ones, and then generalized to a reduction pair to further subsume more recent techniques [53]. The ***Na****goya* ***T****ermination* ***T****ool* (NaTT) [52] was originally developed solely to demonstrate the power of WPO. It participated in the full run of the

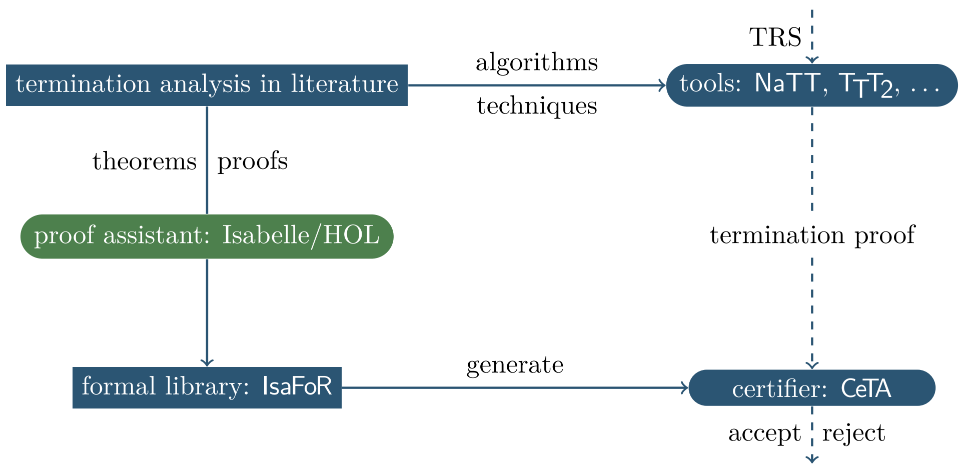

Figure 1 Procedure for Certification of Termination Proofs via IsaFoR/CeTA.

2013 edition of the Termination Competition [18] and won the second place, closing 34 of 159 then-open problems in the TRS Standard category. In 28 of them WPO was essential (the others are due to the efficiency of NaTT) [53].

Despite the significance of the result, two natural questions arise:

(1) "Is the theory of WPO correct?," and if yes

(2) "Is NaTT's implementation of the theory correct?".

So far, nobody investigated the 34 proofs found by NaTT; these benchmarks are obtained via automatic transformations from other systems, and hence hard to analyze by hand (they have up to a few hundred of rules). In this work, we answer the two questions.

To this end, we extend IsaFoR and CeTA [47]. The former, **Isa**belle **Fo**rmalization of **R**ewriting, is an Isabelle/HOL [35]-formalized library of correctness proofs of analysis techniques for term rewriting and transition systems, and the latter, **Ce**rtified **T**ool **A**ssertions, is a verified Haskell code generated from IsaFoR that takes machine-readable output from untrusted verifiers and checks whether techniques are applied correctly. This workflow is illustrated in Figure 1.

In this paper we describe two main extensions of IsaFoR and CeTA. After preliminaries we develop formal proofs of the properties of WPO being a reduction pair in Section 3. Here, we illustrate that one refinement of WPO provided in [53] breaks transitivity in a corner case, but we also show how to repair it by adding a mild precondition. Second, in Section 4 we formalize the max-polynomial interpretations that are used in [53] in a general manner. There we utilize our recently developed verified SMT solver for integer arithmetic [7, 8]. In Section 5 we give a short overview of our new certificate parser implementation in Isabelle/HOL and the format for certificates for WPO and max-polynomial interpretations. In Section 6, we experimentally evaluate our extensions of CeTA. To this end, we extend NaTT to be able to output certificates introduced in the preceding section, and we also integrate WPO in the **T**yrolean **T**ermination **T**ool **2** (T_TT_2) [27]. Details on the experiments are provided at:

`http://cl-informatik.uibk.ac.at/isafor/experiments/wpo/`

This website also provides links to the formalization.

Related Work

There is plenty of work on orders for proving termination of rewriting. The earliest such work we are aware of is Knuth and Bendix' order (KBO), introduced in their celebrated paper in 1970 along with the Knuth–Bendix completion [26]. In the same year, Manna and Ness [33] proposed a semantic approach, which nowadays is called *interpretation methods*. One instantiation of this approach is Lankford's *polynomial interpretation method* [30], which he also combined with KBO [31]. Dershowitz [14] initiated a purely syntactic approach called *recursive path orders (RPO)*, where he also discovered the notion of simplification orders.

The *dependency pair method* of Arts and Giesl [1] boosted the power of termination proving techniques, and around the same time many automated termination provers emerged: AProVE [17], TTT [21], CiME3 [10], Matchbox [49], muterm [32], TORPA [57], and so on, which have been evaluated annually in *the Termination Competition* [18] since 2004. Results of the competition regularly reveal that we cannot blindly trust such automated tools, when one tool claims a TRS terminating, while another claims the same TRS nonterminating.

Hence *certification* came into play. Besides our IsaFoR/CeTA, we are aware of at least two other systems for certifying termination proofs of TRSs: Coccinelle/CiME3 [11] and CoLoR/Rainbow [6]. Here, Coccinelle and CoLoR are similar to IsaFoR: they are all formal libraries on rewriting, though the former two are in Coq [5] instead of Isabelle. The choice of proof assistant aside, a significant difference to IsaFoR/CeTA is in the workflow when performing certification: CiME3 and Rainbow transform termination proofs into Coq files that reference their corresponding formal libraries, and then Coq does the final check, whereas in our case we just run the generated Haskell code CeTA outside of Isabelle.

Within IsaFoR, most closely related to the current work is the previous formalization [46] of RPO, since RPO and WPO are similar in its structure. We refer to Section 3 for more details on how we exploit this similarity.

We would also like to mention some related work outside of pure term rewriting. Recently a verified ordered resolution prover [36] has been developed as part of the IsaFoL project, the **Isa**belle **Fo**rmalization of **L**ogic. Currently the verified prover is based on KBO, which could be replaced by the stronger and more general WPO. In fact, WPO is already utilized in the E theorem prover [24].

In recent work [8] IsaFoR became capable of certifying termination proofs for *integer transition systems*. This work eventually led to a verified SMT solver for linear integer arithmetic [7], which we heavily reuse in our current work.

2 Preliminaries

2.1 Term Rewriting

We assume familiarity with term rewriting [2], but briefly recall notions that are used in the following. A *term* built from *signature* $\mathcal{F}$ and set $\mathcal{V}$ of variables is either $x \in \mathcal{V}$ or of form $f(t_1, \dots, t_n)$, where $f \in \mathcal{F}$ is n-ary and $t_1, \dots, t_n$ are terms. A *context* C is a term with one hole, and $C[t]$ is the term where the hole is replaced by t. The *subterm relation* $\trianglerighteq$ is defined by $C[t] \trianglerighteq t$. A *substitution* is a function σ from variables to terms, and we write $t\sigma$ for the *instance* of term t in which every variable x is replaced by $\sigma(x)$. A *term rewrite system (TRS)* is a set $\mathcal{R}$ of *rewrite rules*, which are pairs of terms ℓ and r indicating that an instance of ℓ in a term can be rewritten to the corresponding instance of r. $\mathcal{R}$ is *terminating* if no term can be rewritten infinitely often.

A *reduction pair* is a pair $(\succ, \succsim)$ of two relations on terms that satisfies the following requirements: $\succ$ is well-founded, $\succsim$ and $\succ$ are compatible (i.e., $\succsim \circ \succ \circ \succsim \subseteq \succ$), both are closed under substitutions, and $\succsim$ is closed under contexts. If $\succ$ is also closed under contexts, then we call $(\succ, \succsim)$ a *monotone* reduction pair. If the first component $\succ$ of a monotone reduction pair is transitive, it is called a *reduction order*. While reduction orders are used to directly prove termination by $\mathcal{R} \subseteq \succ$, reduction pairs are usually employed for termination proofs with dependency pairs. We write $\succ^{\mathsf{lex}}$ and $\succ^{\mathsf{mul}}$ for the lexicographic and multiset extension induced by $(\succ, \succsim)$, respectively.

A *weakly monotone ($\mathcal{F}$-)algebra* $\mathcal{A}$ is a well-founded ordered set $(A, >)$ equipped with an interpretation $f_\mathcal{A} : A^n \to A$ for every n-ary $f \in \mathcal{F}$, such that $f_\mathcal{A}(\dots, a, \dots) \geq f_\mathcal{A}(\dots, b, \dots)$ whenever $a \geq b$. Any weakly monotone algebra $\mathcal{A}$ induces a reduction pair $(>_\mathcal{A}, \geq_\mathcal{A})$ defined by $s \,{}_{(}\!\geq_{)\mathcal{A}}\, t$ iff $[\![s]\!]^\alpha_\mathcal{A} \,{}_{(}\!\geq_{)}\, [\![t]\!]^\alpha_\mathcal{A}$ for all assignments α. Here, $[\![t]\!]^\alpha_\mathcal{A}$ denotes term evaluation in the algebra with respect to an assignment $\alpha : \mathcal{V} \to A$.

A *(partial) status* is a mapping π which assigns to each n-ary symbol f a list $\pi(f) = [i_1, \dots, i_m]$ of indices in $\{1, \dots, n\}$. Abusing notation, we also use $\pi(f)$ as the set $\{i_1, \dots, i_m\}$, and as an operation on n-ary lists defined by $\pi(f)[t_1, \dots, t_n] = [t_{i_1}, \dots, t_{i_m}]$.

A binary relation $\succ$ over terms is *simple with respect to* status π, if $f(t_1, \dots, t_n) \succ t_i$ for all $i \in \pi(f)$. It is *simple*, if it is simple independent of the status. In particular, a simple reduction order is called a *simplification order*.

A *precedence* is a preorder $\succsim$ on $\mathcal{F}$, such that $\succ := \succsim \setminus \precsim$ is well-founded.

▶ **Definition 1** (WPO [53, Def. 10, incl. Refinements (2c) and (2d) of Sect. 4.2]). *Let $\mathcal{A}$ be a weakly monotone algebra, $\succsim$ a precedence, and π be a status. Let $\geq_\mathcal{A}$ be simple with respect to π. The WPO reduction pair $(\succ_{\mathsf{WPO}}, \succsim_{\mathsf{WPO}})$ is defined as follows: $s \succ_{\mathsf{WPO}} t$ iff*

1. *$s >_\mathcal{A} t$, or*
2. *$s \geq_\mathcal{A} t$ and*
  a. *$s = f(s_1, \dots, s_n)$ and $\exists i \in \pi(f).\ s_i \succsim_{\mathsf{WPO}} t$, or*
  b. *$s = f(s_1, \dots, s_n)$, $t = g(t_1, \dots, t_m)$, $\forall j \in \pi(g).\ s \succ_{\mathsf{WPO}} t_j$ and*
    i. *$f \succ g$ or*
    ii. *$f \succsim g$ and $\pi(f)[s_1, \dots, s_n] \succ^{\mathsf{lex}}_{\mathsf{WPO}} \pi(g)[t_1, \dots, t_m]$.*

The relation $s \succsim_{\mathsf{WPO}} t$ is defined in the same way, where $\succ^{\mathsf{lex}}_{\mathsf{WPO}}$ in the last line is replaced by $\succsim^{\mathsf{lex}}_{\mathsf{WPO}}$, and there are the following additional subcases in case **2***:*

  c. *$s \in \mathcal{V}$ and either $s = t$ or $t = g(t_1, \dots, t_m)$, $\pi(g) = \emptyset$ and g is least in precedence,*
  d. *$s = f(s_1, \dots, s_n)$, $t \in \mathcal{V}$, $>_\mathcal{A}$ is simple w.r.t. π, and $\forall g.\ f \succ g \lor (f \succsim g \land \pi(g) = \emptyset)$.*

▶ **Theorem 2** ([53]). *WPO forms a reduction pair.* ◀

For the certification purpose it suffices to formalize Theorem 2 and to provide a verified implementation to check WPO constraints of the form $s \,{}_{(}\!\succsim_{)}\, t$ for a concrete instance of WPO. In [53] it is further shown that a number of existing methods are obtained as instances of WPO, namely: the Knuth–Bendix order (KBO) [26], interpretation methods [15, 30], polynomial KBO [31], lexicographic path orders (LPO) [25], and non-collapsing argument filters [1, 29]. This means that, by having a WPO certifier, one can also certify these existing methods.

2.2 Isabelle/HOL and IsaFoR

We do not assume familiarity with Isabelle/HOL, since most of the illustrated formal statements are close to mathematical text. We give some brief explanations by illustrating certain term rewriting concepts via their counterparts in IsaFoR. For instance, IsaFoR contains a datatype for terms, ('f,'v)term, where 'f and 'v are type-variables representing the signature $\mathcal{F}$ and the set of variables $\mathcal{V}$, respectively. A typing judgment is of the form *term* :: *type*. As an example, R :: ('f,'v)term rel states that R has type ('f,'v)term rel, i.e., R is a binary relation over terms.

An Isabelle *locale* [3] is a named context where certain elements can be fixed and properties can be assumed. Locales are frequently used in IsaFoR. For instance, reduction pairs in IsaFoR are formulated as a locale redpair.[1] Here, O is relation composition, and SN is a predicate for well-foundedness (strong normalization).

```
locale redpair =
  fixes S NS :: "('f,'v)term rel"
  assumes "SN S"
    and "ctxt.closed NS"
    and "subst.closed S" and "subst.closed NS"
    and "NS O S ⊆ S" and "S O NS ⊆ S"
```

Locales are also useful to model hierarchical structures. For instance, whereas redpair does not require that the relations are orders, this is required in the upcoming locale redpair_order which is an extension of redpair.

```
locale redpair_order = redpair S NS +
  assumes "trans S" and "trans NS" and "refl NS"
```

Beside the abstract definitions for reduction pairs, IsaFoR also provides several instances of them, e.g., one for RPO, one for KBO [40], etc. These instances can then be used in termination techniques like the reduction pair processor to validate concrete termination proofs. However, often the requirements of a reduction pair are not yet enough. As an example, the usable rules refinement [20, 48] requires $\mathcal{C}_e$-compatible reduction pairs and argument filters. To this end IsaFoR contains the locale ce_af_redpair_order. It extends redpair_order by a new parameter π for the argument filter, and demands the additional requirements.

```
locale ce_af_redpair_order = redpair_order S NS +
  fixes π :: "'f af"
  assumes "af_compatible π NS"
    and "ce_compatible NS"
```

There are further locales for monotone reduction pairs, for reduction pairs which can be used in complexity proofs, etc.

3 Formalization of WPO

In this section we present our formalization of WPO. It starts by formalizing the properties of WPO in Section 3.1, so that we can add WPO as a new instance of a reduction pair to IsaFoR. Afterwards we illustrate our verified implementation for checking WPO constraints in Section 3.2.

[1] In IsaFoR, there is a more general locale for reduction *triples* (redtriple), which we simplify to reduction *pairs* in the presentation of this paper.

3.1 Properties of WPO

As we have seen in Section 2.2, IsaFoR already contains several formalized results about reduction pairs, including general results, instances, and termination techniques based on reduction pairs. In contrast, at the start of this formalization of WPO, IsaFoR did not contain a single locale about generic weakly monotone algebras. In particular, the formalization of matrix interpretations and polynomial interpretations [42] directly refers to redpair and its variants. So, the question arises, how the generic version of WPO in Definition 1 can be formalized, which is based on arbitrary weakly monotone algebras.

The obvious approach would have been to just add the missing pieces. To be more precise, we could have formalized weakly monotone algebras in IsaFoR and then on top have formally verified the properties of WPO. However, this approach has the disadvantage that we would have had to adjust also existing instances of weakly monotone algebras (like polynomial interpretations, arctic interpretations, and matrix interpretations) to the new interface.

Therefore, we choose a different approach, namely to reformulate the definition of WPO such that it does no longer depend on the notion of weakly monotone algebra, but instead directly refers to reduction pairs (cf. Definition 3).

▶ **Definition 3** (WPO based on Reduction Pairs). *Let $(>_{\mathcal{A}}, \geq_{\mathcal{A}})$ be a reduction pair, $\succsim$ a precedence, ... and continue as in Definition 1 to define the relations $\succ_{\mathsf{WPO}}$ and $\succsim_{\mathsf{WPO}}$.*

In this way, all instances of reduction pairs in IsaFoR immediately become available as parameters to WPO. On the one hand, we can parameterize WPO with (max-)polynomial interpretations and matrix interpretations as is already done in the literature. On the other hand, it is also possible to use KBO or RPO as parameter to WPO, or even to nest WPOs recursively.

Of course the question is, how easy it is to formally prove properties of this WPO based on reduction pairs. At this point we profit from the fact that the structure of WPO is quite close to other path orders like RPO, and that the latter has already been fully formalized in IsaFoR.

▶ **Definition 4** (RPO as formalized in IsaFoR). *Let $\succsim$ be a precedence and σ be a function of type $\mathcal{F} \to \{\mathsf{lex}, \mathsf{mul}\}$. We define the RPO reduction pair $(\succ_{\mathsf{RPO}}, \succsim_{\mathsf{RPO}})$ as follows: $s \succ_{\mathsf{RPO}} t$ iff*

- **a.** *$s = f(s_1, \ldots, s_n)$ and $\exists i \in \{1, \ldots, n\}$. $s_i \succsim_{\mathsf{RPO}} t$, or*
- **b.** *$s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, $\forall j \in \{1, \ldots, m\}$. $s \succ_{\mathsf{RPO}} t_j$ and*
  - **i.** *$f \succ g$ or*
  - **ii.** *$f \succsim g$ and $\sigma(f) = \sigma(g)$ and $[s_1, \ldots, s_n] \succ^{\sigma(f)}_{\mathsf{RPO}} [t_1, \ldots, t_m]$.*
  - **iii.** *$f \succsim g$ and $\sigma(f) \neq \sigma(g)$ and $n > 0$ and $m = 0$.*

The relation $s \succsim_{\mathsf{RPO}} t$ is defined in the same way, where $\succ^{\sigma(f)}_{\mathsf{RPO}}$ in case bii is replaced by $\succsim^{\sigma(f)}_{\mathsf{RPO}}$, the condition $n > 0$ in case biii is dropped, and there is one additional subcase:

- **c.** *$s \in \mathcal{V}$ and either $s = t$ or $t = c$ where c is a constant in $\mathcal{F}$ that is least in precedence.*

So, we start our formalization of WPO by copy-and-pasting the definitions and proofs about RPO, and renaming every occurrence of "RPO" to "WPO." At this point we have a fully verifiable Isabelle theory which defines WPO as a copy of RPO.

Next, we modify a couple of definitions, such that eventually, we arrive at a formalized variant of the WPO in Definition 3. For each modification, we immediately adjust the formal proofs. Such adjustments are mostly straight-forward, not least due to the valuable support by the proof assistant: we are immediately pointed to those parts of proofs which are broken by a modification, without having to manually recheck the remaining proofs that were not affected.

To be more precise, we perform the following sequence of modifications.

- We delete σ from RPO and replace it by lex, as the choice between multiset and lexicographic comparison via σ is not present in WPO. As a result, case biii is dropped, case bii always uses lexicographic comparison, and the formal proofs become shorter.
- We add the two tests $s \geq_{\mathcal{A}} t$ and $s >_{\mathcal{A}} t$ that are present in WPO, but not in RPO. Moreover, we add the requirement of WPO, that $\geq_{\mathcal{A}}$ must be simple, in order to adjust all the proofs of the defined relations.
- We include the status π, which is present in the WPO definition, but not in RPO. In this step we also weaken the requirement of $\geq_{\mathcal{A}}$ being simple to the requirement that $\geq_{\mathcal{A}}$ is simple with respect to π.
- We generalize case c of RPO in such a way that not only for constants c we permit $x \succsim_{\mathsf{WPO}} c$, but also $x \succsim_{\mathsf{WPO}} g(t_1, \ldots, t_n)$ is possible if $\pi(g) = \emptyset$.
- We finally add refinement 2d under the premise that $>_{\mathcal{A}}$ is simple with respect to π. At this point we have precisely a formalized version of WPO as defined in Definition 3.

Interestingly, after the final refinement we were no longer able to show all properties of $(\succ_{\mathsf{WPO}}, \succsim_{\mathsf{WPO}})$. For example, the transitivity proof of $\succsim_{\mathsf{WPO}}$ was broken and we were not able to repair it. Indeed, it turns out that $\succsim_{\mathsf{WPO}}$ is no longer transitive with the refinement as illustrated by Example 5. This example was constructed with the help of Isabelle, since it directly pointed us to the case where the transitivity proof got broken.

▶ **Example 5.** Consider $\mathcal{F} = \{\mathsf{a}\}$, $\pi(\mathsf{a}) = []$, and a reduction pair (or algebra) where $\geq_{\mathcal{A}}$ relates all terms and $>_{\mathcal{A}}$ is empty. Then $x \succsim_{\mathsf{WPO}} \mathsf{a} \succsim_{\mathsf{WPO}} y$, but $x \succsim_{\mathsf{WPO}} y$ does not hold.

The reduction pair (or algebra) in Example 5 is obviously a degenerate case. In fact, by excluding this degenerate case, we can formally prove that WPO including refinement 2d is a reduction pair.

To this end, we gather all parameters of WPO in a locale and assume relevant properties of these parameters, either via other locales or as explicit assumptions. The precedence $\succsim$ is specified by way of three functions prc, pr_least, and pr_large: prc takes two symbols f and g and returns a pair of Booleans $(f \succ g, f \succsim g)$; pr_least is a predicate telling whether a symbol is least in $\succsim$ or not; and pr_large states whether a symbol is largest in $\succsim$ with respect to π or not, as required in rule 2d of Definition 1. Whereas most of the properties of the precedence are encoded via an existing locale precedence, for a symbol being of *largest* precedence we add two new assumptions explicitly. In the locale we further use a Boolean ssimple to indicate whether $>_{\mathcal{A}}$ is simple with respect to π, i.e., whether it is allowed to apply rule 2d or not. Only then, the properties of pr_large must be satisfied and the degenerate case must be excluded. Being simple with respect to π is enforced via the predicate simple_arg_pos: for any relation R the property simple_arg_pos R f i ensures that $f(t_1, \ldots, t_n)$ R t_i holds for all $t_1, \ldots, t_n$.

```
locale wpo_params = redpair_order S NS + precedence prc pr_least
  for S NS :: "('f, 'v) term rel"                    (* underlying reduction pair *)
    and prc :: "'f ⇒ 'f ⇒ bool × bool" and pr_least pr_large :: "'f ⇒ bool"(* precedence *)
    and ssimple :: bool                       (* flag whether rule (2d) is permitted *)
    and π :: "'f status" +                                                  (* status *)
  assumes "S ⊆ NS"
    and "i ∈ π f ⟹ simple_arg_pos NS f i"                    (* NS is simple w.r.t. π *)
    and "ssimple ⟹ i ∈ π f ⟹ simple_arg_pos S f i"            (* S is simple w.r.t. π *)
    and "ssimple ⟹ NS ≠ UNIV"                               (* exclude degenerate case *)
    and "ssimple ⟹ pr_large f ⟹ fst (prc f g) ∨ snd (prc f g) ∧ π g = []"
    and "ssimple ⟹ pr_large f ⟹ snd (prc g f) ⟹ pr_large g"
```

Within the locale we define the relations WPO_S and WPO_NS ($\succ_{\mathsf{WPO}}$ and $\succsim_{\mathsf{WPO}}$ of Definition 3) with the help of a recursive function, and prove the main theorem:

theorem "redpair_order WPO_S WPO_NS"

Moreover, we prove that whenever the non-strict relation is compatible with an argument filter μ then also the WPO is compatible with $\pi \cup \mu$, defined as $(\pi \cup \mu)(f) = \pi(f) \cup \mu(f)$.

lemma assumes "af_compatible μ NS"
  shows "af_compatible $(\pi \cup \mu)$ WPO_NS"

We further prove that WPO is also $\mathcal{C}_e$-compatible under mild preconditions, namely whenever $\pi(f)$ includes the first two positions of some symbol f. Finally, we formalize that WPO can be used in combination with usable rules, since it is an instance of the corresponding locale:

lemma assumes "$\exists$f. $\{0,1\} \subseteq \pi$ f" (* positions in IsaFoR start from 0 *)
  and "af_compatible μ NS"
  shows "ce_af_redpair_order WPO_S WPO_NS $(\pi \cup \mu)$"

At the moment, our formalization does not cover any comparison to other term orders. There is, for example, no formal statement that each polynomial KBO can be formulated as an instance of WPO. The simple reason is that such a formalization will not increase the power of the certifier, and the support for polynomial KBO can much easier be added by just translating an instance of polynomial KBO into a corresponding WPO within a certificate, e.g., when generating certificates in a termination tool or when parsing certificates in CeTA.

3.2 Checking WPO Constraints

Recall that our formalization of WPO in Section 3.1 has largely been developed by adjusting the existing formal proofs for RPO. When implementing an executable function to check constraints of a particular WPO instance, where precedence, status, etc. are provided, there is however one fundamental difference to RPO: in WPO we need several tests $s >_{\mathcal{A}} t$ and $s \geq_{\mathcal{A}} t$ of the underlying reduction pair. And in general, these tests are just *approximations*, e.g., since testing positiveness of non-linear polynomials is undecidable.

In order to cover approximations, the implementations of reduction pairs in IsaFoR adhere to the following interface, which is a record named redpair that contains five components:

- One component is for checking validity of the input. For instance, for polynomial interpretations here one would check that each interpretation of an n-ary function symbol is a polynomial which only uses variables $x_1, \ldots, x_n$.
- There are two functions check_S and check_NS of type ('f,'v)term $\Rightarrow$ ('f,'v)term $\Rightarrow$ bool for approximating whether two terms are strictly and weakly oriented, respectively.
- There is a flag mono which indicates whether the reduction pair is monotone. An enabled mono-flag is required for checking termination proofs without dependency pairs.
- The implicit argument filter of the reduction pair can be queried, a feature that is essential for usable rules.

The generic interface is instantiated by all reduction pair (approximations) in IsaFoR, and they satisfy the common soundness property, that for a given approximation of a reduction pair rp and for given finite sets of strict- and non-strict-constraints, represented as two lists S_list and NS_list, there exists a corresponding reduction pair that orients all constraints in

S_list strictly and in NS_list weakly. In order to simplify the presentation, in the upcoming formal sources we uniformly use set comprehensions instead of list comprehensions, and we omit all conversions between lists and sets.

```
assumes "redpair.valid rp"                    (∗ generic_reduction_pair ∗)
  and "∀ (s,t) ∈ S_list. redpair.check_S rp s t"
  and "∀ (s,t) ∈ NS_list. redpair.check_NS rp s t"
shows "∃ S NS.
      ce_af_redpair_order S NS (redpair.af rp) ∧
      S_list ⊆ S ∧ NS_list ⊆ NS ∧
      (redpair.mono rp ⟶ ctxt.closed S)"
```

We next explain how to instantiate this interface by WPO. To be more precise, we are given a status π, a precedence, and an approximated reduction pair rp and have to implement the interface for WPO such that generic_reduction_pair is satisfied.

For checking validity of WPO, we assert redpair.valid rp and in addition perform checks that the status π is well-defined, i.e., $\pi(f) \subseteq \{1, \dots, n\}$ must hold for each n-ary symbol f. Moreover, we globally compute symbols of largest and least precedence, i.e., the functions pr_least and pr_large of the wpo_params-locale. We further set the argument filter of WPO to $\pi \cup$ redpair.af af.

For determining the ssimple parameter of the wpo_params-locale, there is the problem, that we do not know whether the generated strict relation S will be simple with respect to π. Moreover, to instantiate the locale, we always must ensure that NS is simple with respect to π. Unfortunately, the formal statement of generic_reduction_pair does not include any such information.

We solve this problem by enlarging the record redpair by two new entries for strict and weak simplicity, and require in generic_reduction_pair that if these flags are enabled, then the relations S and NS must be simple with respect to π, respectively. Whereas now all required information for WPO is accessible via the interface, the change of the interface requires to adapt all existing reduction pairs in IsaFoR, like polynomial interpretations, to provide the new information. To be more precise, we formalize two sufficient criteria for each reduction pair in IsaFoR, that ensure simplicity of the weak and strict relation, respectively.

At this point all parameters of WPO are fixed, except for S and NS. We now define the approximation of WPO as the WPO where S and NS are replaced by redpair.check_S rp and redpair.check_NS rp, respectively.

Next, we are given two lists of constraints wpo_S_list and wpo_NS_list that are oriented by the approximation of WPO. Out of these we extract the lists S_list and NS_list that contain all invocations of the underlying approximated reduction pair rp within the recursive definition of WPO, for instance:

$$\mathsf{S\_list} = \{(s_i, t_i) \mid (s,t) \in \mathsf{wpo\_S\_list} \cup \mathsf{wpo\_NS\_list}, s \trianglerighteq s_i, t \trianglerighteq t_i, \mathsf{redpair.check\_S\ rp}\ s_i\ t_i\}$$

After these lists have been defined, we apply generic_reduction_pair to get access to the (non-approximated) reduction pair in the form of relations S and NS. With these we are able to instantiate the wpo_params-locale and get access to the reduction pair WPO_S and WPO_NS. We further know that the approximations in S_list and NS_list are correct, e.g., whenever $(s,t) \in$ wpo_S_list $\cup$ wpo_NS_list, $s \trianglerighteq s_i$, $t \trianglerighteq t_i$ and redpair.check_S rp s_i t_i then $(s_i, t_i) \in$ S. With this auxiliary statement we finally prove that the approximated WPO corresponds to the actual WPO for all constraints in wpo_S_list $\cup$ wpo_NS_list. So, we have a reduction pair WPO_S and WPO_NS and an approximation statement, as required by generic_reduction_pair.

In total, we get an interpretation of the generic interface for WPO, and thus can use WPO in every termination technique of IsaFoR which is based on reduction pairs.

4 Integration of Max-Polynomial Interpretation

As already mentioned in the previous section, various kinds of interpretation methods have been formalized in IsaFoR and supported by CeTA. However, max-polynomial interpretations [16] were not yet supported. Hence we extend IsaFoR and CeTA to incorporate them, in particular those over natural numbers as required by WPO instances introduced in [53].

In order for CeTA to certify proofs using max-polynomial interpretations, we must formally prove that the pair of relations $(>_{\mathcal{A}}, \geq_{\mathcal{A}})$ forms a reduction pair, and implement a verifier to check $s >_{\mathcal{A}} t$ and $s \geq_{\mathcal{A}} t$. The former is easy, it is clearly weakly monotone and well-founded. For a verified comparison of max-polynomials, instead of implementing a dedicated checker from scratch, we chose to reduce the comparison of max-polynomials to the validity of an integer arithmetic formula without max, for which we already have a formalized validity checker [7, 8]. This checker is essentially an SMT-solver for linear integer arithmetic that we utilize to ensure unsatisfiability of negated formulas.

We formalize max-polynomials in IsaFoR as terms over the following signature.

```
datatype sig = ConstF nat | SumF | ProdF | MaxF
```

The interpretation of these symbols is as expected:

```
primrec I where
  "I (ConstF n) = (λx. n)"
| "I SumF = sum_list"
| "I ProdF = prod_list"
| "I MaxF = max_list"
```

In order to compare max-polynomials, we first normalize them according to the following four distribution rules:

$$\max(x,y)+z \to \max(x+z, y+z) \qquad x+\max(y,z) \to \max(x+y, x+z)$$
$$\max(x,y)\cdot z \to \max(x\cdot z, y\cdot z) \qquad x\cdot\max(y,z) \to \max(x\cdot y, x\cdot z)$$

Note that the distribution of multiplication over max is admissible because we are only considering natural numbers. This way, the max-polynomials s and t are normalized to $\max_{i=1}^{n} s_i$ and $\max_{i=1}^{m} t_i$, where $s_1, \dots, s_n$ and $t_1, \dots, t_m$ are polynomials (without max). In IsaFoR we define the mapping from s to $s_1, \dots, s_n$ as to_IA. Then the comparison of two such normal forms is easily translated to an arithmetic formula without max [4]:

$$s \;{}_{(}\!\leq\!{}_{)}\; t \iff \max_{i=1}^{n} s_i \;{}_{(}\!\leq\!{}_{)}\; \max_{j=1}^{m} t_j \iff \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} s_i \;{}_{(}\!\leq\!{}_{)}\; t_j$$

This reduction is formalized in Isabelle as follows. Here, operators with subscript "$_{\mathsf{f}}$" build syntactic formulas, and those with prefix "IA." or subscript "$_{\mathsf{IA}}$" come from the formalization of integer arithmetic; e.g., "$\bigwedge_{\mathsf{f}}$ x ← xs. IA.const 0 $\leq_{\mathsf{IA}}$ IA.var x" denotes an integer arithmetic formula representing "$0 \leq x_1 \wedge \cdots \wedge 0 \leq x_n$", where xs $= [x_1, \dots, x_n]$. Since we are originally concerned about natural numbers, in the following definitions we insert such assumptions for the list of variables occurring in s and t. Initially we did not impose these assumptions and consequently, several valid termination proofs could not be certified. We thank Sarah Winkler for spotting and fixing this omission.

```
definition le_via_IA where "le_via_IA s t ≡
  (⋀f x ← vars_term_list s @ vars_term_list t. IA.const 0 ≤IA IA.var x) ⟶f
  (⋀f si ← to_IA s. ⋁f tj ← to_IA t. si ≤IA tj)"

definition less_via_IA where "less_via_IA s t ≡
  (⋀f x ← vars_term_list s @ vars_term_list t. IA.const 0 ≤IA IA.var x) ⟶f
  (⋀f si ← to_IA s. ⋁f tj ← to_IA t. si <IA tj)"
```

The soundness of the reduction is formally proved as follows.

```
lemma le_via_IA:
  assumes "⊨IA le_via_IA s t" shows "s ≤𝒜 t"

lemma less_via_IA:
  assumes "⊨IA less_via_IA s t" shows "s <𝒜 t"
```

Because of lemmas le_via_IA and less_via_IA it is now possible to invoke the validity checker for integer arithmetic on the formulas le_via_IA t s and less_via_IA t s in order to soundly validate the comparisons $s \geq_{\mathcal{A}} t$ and $s >_{\mathcal{A}} t$, respectively.

Finally all results are put together to form an instance of an generic_reduction_pair of Section 3.2, namely a verified implementation for max-polynomial interpretations.

5 Certificate Format and Parser

The *Certification Problem Format (CPF)* [41] is a machine-readable XML format, which was codeveloped by several research groups of the term rewriting community to serve as the standard communication language between automated provers and certifiers.

Here we present the additions to CPF that are part of the current work, i.e., the certificate format for WPO and max-polynomial interpretations. Moreover, we comment on our complete overhaul of CeTA's certificate parser. Before this overhaul, the certificate parser was built on top of an Isabelle/HOL formalized XML transformer library [43] that has several limitations. In this context an XML transformer is a parser that consumes an XML element and produces results represented by arbitrary (custom) data types. (In the remainder, we will use "parser" and "transformer" synonymously.) In the current work we develop a more concise and flexible XML transformer library, which allows for syntax similar to Haskell's do-notation.

In our formalization, the type of XML transformers is 'a xmlt2, which is a function that takes the internal representation of an XML element and returns, in form of direct sums, either the result of a successful parse (type 'a) or an error state.

The notation "XMLdo s {...}" yields a transformer for an XML element whose root tag is s. Within an XMLdo block, we can parse child elements by the binding "x ← *inner*;" or one of its variants such as "xs $\leftarrow\hat{}\{\ell..u\}$ *inner*;" which binds xs to the list of values resulting from transforming at least ℓ and at most u inner elements using the XML transformer *inner*. Here u is of type enat (extended naturals), so that it can be ∞. The frequent instance $\leftarrow\hat{}\{0..\infty\}$ is also written $\leftarrow *$. Typically a parser block should end with "xml_return r", where r is a return value that may rely on previously bound variables. This invocation also checks that after parsing, there are no child elements left in the current XML element. This ensures that the transformer defines a grammar.

Given two parsers p_1 and p_2, we allow a choice between them by "p_1 XMLor p_2". This works as follows: if parser p_1 returns a *recoverable* error state, then we continue with p_2. Otherwise, the result of "p_1 XMLor p_2" is the result of p_1. Here recoverable means that the tag of the root element is not consumed by p_1. If p_1 consumed the root node but failed for some child element, then it yields an unrecoverable error state containing an appropriate error message.

In the following we illustrate our approach by some parsers from our formalization. The new notation should make it fairly easy to translate between such parsers and corresponding specifications for the CPF format. Until a certain moment in the development we stated all parsers using Isabelle's **function** command, which specifies a recursive function along with a proof that the function is totally defined. For humans, proving XML transformers well-defined is rather easy, only requiring a dedicated measure for the internal XML representations. For Isabelle, however, accepting the proofs turned out to be excessively slow. Especially for the big parser that covers the entire CPF. Therefore, we now define our transformers via the **partial_function** [28] command, which does not perform such proofs and therefore is much faster.

A first concrete example is a parser for expressions occurring in max-polynomial interpretations. Here notions defined in Section 4 are accessed via prefix "max_poly.", and (STR "...") is the notation for target-language strings in Isabelle/HOL.

```
partial_function (sum_bot) exp_parser :: "(max_poly.sig, nat) term xmlt2" where
  [code]: "exp_parser xml = (
  XMLdo (STR "product") {
    exps ←* exp_parser; xml_return (Fun max_poly.ProdF exps)
  } XMLor XMLdo (STR "sum") {
    exps ←* exp_parser; xml_return (Fun max_poly.SumF exps)
  } XMLor XMLdo (STR "max") {
    exps ←^{1..∞} exp_parser; xml_return (Fun max_poly.MaxF exps)
  } XMLor XMLdo (STR "constant") {
    n ←nat; xml_return (max_poly.const n)
  } XMLor XMLdo (STR "variable") {
    n ←nat; xml_return (Var (n − 1))
  }) xml"
```

The parser recursively defines the grammar of max-polynomial expressions (as a *complex type* in XML schema terminology). It is a choice among the elements `<product>`, `<sum>`, `<max>`, `<constant>` and `<variable>`. Elements `<product>` and `<sum>` recursively contain an arbitrary number of subexpressions and construct corresponding terms over signature max_poly.sig. Element `<max>` is similar, except that it demands at least one subexpression. Element `<constant>` contains just a natural number, which is parsed as a constant. Element `<variable>` also contains a natural number, which indicates the i-th variable (using zero-based indexing).

The extended format for reduction pairs (triples) is as follows:

```
partial_function (sum_bot) redtriple :: "'a redtriple_impl xmlt2" where
  [code]: "redtriple xml = ( ...                        (* existing reduction pairs *)
    XMLor XMLdo (STR "maxPoly") {          (* max−polynomial interpretations *)
      inters ←* XMLdo (STR "interpret") {
        f ← xml2name;
        a ← XMLdo (STR "arity") { a ←nat; xml_return a };
        e ← exp_parser;
        xml_return ((f, a), e)
      };
      xml_return (Max_poly inters)
    } XMLor XMLdo (STR "weightedPathOrder") {      (* new alternative for WPO *)
      a ← wpo_params;
      b ← redtriple;
      xml_return (WPO a b)
    }
    XMLor XMLdo (STR "filteredRedPair") {...}     (* collapsing argument filter *)
  ) xml"
```

It is extended from the previous reduction pairs with three new alternatives. Element `<maxPoly>` is the reduction pair induced by max-polynomial interpretations, which is a list of elements `<interpret>`, each assigning a function symbol f of arity a its interpretation as expression e. The `<weightedPathOrder>` element characterizes a concrete WPO reduction pair. It consists of WPO specific parameters wpo_params that fixes status and precedences, and another reduction pair in a recursive manner, which specifies the "algebra" $\mathcal{A}$ in terms of $(>_{\mathcal{A}}, \geq_{\mathcal{A}})$. The `<filteredRedPair>` element is newly added specially for *collapsing* argument filters. Since partial status subsumes non-collapsing argument filters [50], only dedicated collapsing ones have to be specially supported.

6 Implementations and Experiments

In order to evaluate the relevance of our extension of CeTA by WPO and max-polynomial interpretations, we implement certificate output for WPO in two termination analyzers: NaTT and T$_\mathsf{T}$T$_2$.

NaTT. originates as an experimental implementation of WPO [51]. From its early design NaTT followed the trend [54, 55, 37, 9] of reducing termination problems into SMT problems and employ an external SMT solver, by default, Z3 [12]. Further, NaTT utilizes incremental SMT solving, and implements some tricks for efficiency [52]. In the current work, its output is adjusted to conform to the newly defined XML certificate format for WPO, max-polynomials, and collapsing argument filters. These are essentially the central techniques implemented in NaTT, but a few techniques implemented later on in NaTT had to be deactivated to be able to be certified by CeTA; some of them, such as nontermination proofs, are actually supported but NaTT is not yet adjusted to produce certificates for them.

T$_\mathsf{T}$T$_2$. succeeded the automated termination analyzer T$_\mathsf{T}$T in 2007. It implements numerous (non-)termination techniques. For searching reduction pairs it uses a SAT/SMT-based approach and the SMT solver MiniSMT [56]. We extend T$_\mathsf{T}$T$_2$ by an implementation of WPO, following mostly the presented encodings in [53]. A notable difference in the search space for max-polynomials: while NaTT heuristically chooses between max and sum, T$_\mathsf{T}$T$_2$ embeds this choice into the SMT encoding.

Besides the integration of the full WPO search engine, we would also like to mention an additional feature of T$_\mathsf{T}$T$_2$ regarding WPO. Usual termination tools just try to find *any* proof. Even if users want a specific shape of proofs, they cannot impose constraints on proofs that termination tools find. T$_\mathsf{T}$T$_2$ provides *termination templates* [38] where users can fix parts of proofs via parameters when invoking T$_\mathsf{T}$T$_2$. We also added support for termination templates for WPO, i.e., if one wants to find a specific proof with WPO then (some) values can be fixed with T$_\mathsf{T}$T$_2$ and afterwards CeTA can validate if the proof is correct.

▶ **Example 6.** Consider the following TRS (`Zantema_05/z10.xml` of TPDB):

$$\begin{aligned} \mathsf{a}(\mathsf{lambda}(x), y) &\to \mathsf{lambda}(\mathsf{a}(x, \mathsf{p}(1, \mathsf{a}(y, \mathsf{t})))) & \mathsf{a}(\mathsf{a}(x, y), z) &\to \mathsf{a}(x, \mathsf{a}(y, z)) \\ \mathsf{a}(\mathsf{p}(x, y), z) &\to \mathsf{p}(\mathsf{a}(x, z), \mathsf{a}(y, z)) & \mathsf{a}(\mathsf{id}, x) &\to x \\ \mathsf{a}(1, \mathsf{id}) &\to 1 & \mathsf{a}(\mathsf{t}, \mathsf{id}) &\to \mathsf{t} \\ \mathsf{a}(1, \mathsf{p}(x, y)) &\to x & \mathsf{a}(\mathsf{t}, \mathsf{p}(x, y)) &\to y \end{aligned}$$

If we just call TTT2 with WPO (☑[2]) on this TRS then we get a termination proof consisting of arbitrary values. However, e.g., we might want a specific WPO proof with the precedence $\mathsf{id} > \mathsf{a} > \mathsf{lambda} > \mathsf{t} > 1 > \mathsf{p}$ and a status reversing the arguments of p for the lexicographic comparison. For this we can use the following call (☑):

```
./ttt2 -s "wpo -msum -st \"p = [1;0]\" -prec \"id > a > lambda > t > 1 >
                         p\"" Zantema_05/z10.xml
```

The flag `-msum` activates $\mathcal{MS}um$ (from [53]) as interpretation for WPO, the flag `-st` fixes statuses and the flag `-prec` fixes a (part of a) precedence. Also all other WPO parameters, for the standard instances of [53], can be fixed via flags. In order to be sure that the proof is correct we can call CeTA on the certificate.

As a result we obtain a proof with the stated preconditions and in a broader sense TTT2 can be used to find specific WPO proofs. For some applications, it even makes sense to fix all parameters of WPO, so that there is no search at all. This option is useful for validating WPO-based termination proofs in papers, since writing XML-files in CPF by hand is tedious, but it is easy to invoke TTT2 on an ASCII representation of both the TRS and the WPO parameters. Then one automatically gets the corresponding proof in XML so that validation by CeTA is possible afterwards.

Evaluation. We now evaluate CeTA over the certifiable proofs generated by NaTT and TTT2. Experiments are run on StarExec [45], a computation resource service for evaluating logic solvers and program analyzers. The environment offers an Intel® Xeon® CPU E5-2609 running at 2.40GHz and 128GB main memory for each pair of a solver and problem. We set 300s timeout for each pair, as in the Termination Competition 2019.

We compare six configurations: NaTT, TTT2 without WPO and with WPO, and their variants that restrict to certifiable techniques. The results are summarized in Table 1. We remark that all the proofs generated by certifiable configurations are successfully certified by CeTA. Most notably, the termination proofs for the 34 examples mentioned in the introduction that reportedly only NaTT could prove terminating are verified.

The impact of WPO in TTT2, unfortunately, appears marginal. It only brings two additional termination proofs in the certifiable setting; and in the other setting the small difference would vanish if one slightly modifies the timeout. It is most likely that the proof search heuristic of TTT2 is not optimal, and more engineering effort is necessary in order to maximize the effect of WPO for TTT2.

There are still significant gaps between full and certifiable versions of each tool, since the certifiable versions must disable techniques that are not (fully) supported by CeTA. Among them, both NaTT and TTT2 had to disable or restrict:

- max-polynomial interpretations with negative constants [22, 16];

[2] The link in this icon directs to the web interface of TTT2, preloaded with this example.

Table 1 Certification Experiments.

Tool	Yes	No	Time (tool)	Time (CeTA)
NaTT certifiable	751	7	02:32:39	00:13:46
$\mathsf{T_TT_2}$ w/ WPO certifiable	754	194	1d 10:31:29	00:08:18
$\mathsf{T_TT_2}$ w/o WPO certifiable	752	194	1d 06:26:32	00:03:55
NaTT	864	169	02:42:48	–
$\mathsf{T_TT_2}$ w/ WPO	827	205	13:48:29	–
$\mathsf{T_TT_2}$ w/o WPO	826	205	13:46:57	–

- reachability analysis techniques: for NaTT satisfiability-oriented ones [44], and for $\mathsf{T_TT_2}$ ones based on tree automata [34];
- uncurrying [23]: although the technique itself is fully supported [39], both NaTT and $\mathsf{T_TT_2}$ have their own variants which exceed the capabilities of CeTA.

These observations lead to promising directions of future work. For instance, negative constants seem essentially within reach in light of certified SMT solving.

7 Summary

We have presented an extension of the IsaFoR library and the certifier CeTA with a formalization of WPO. First, we discussed how we obtained WPO as a new reduction pair in IsaFoR while relying on the already existing formalization of RPO and adapting its proofs for the requirements of WPO. Second, we described how max-polynomial interpretations were added to IsaFoR as these are often used in combination with WPO. Afterwards we gave a brief overview of the CPF format and the corresponding parser in CeTA. For this parser we define and employ a notation similar to the do-notation of Haskell, which makes the parser implementation more concise and easier to understand. Finally, we tested the new version of CeTA with the termination provers NaTT and $\mathsf{T_TT_2}$, which both have been extended to generate CPF certificates for WPO. All generated proofs have been validated, including those for the 34 TRSs that reportedly only NaTT could prove terminating.

The main formal developments in this paper consist of only 3669 lines of Isabelle source code, since several concepts were already available in IsaFoR, e.g., lexicographic comparisons and precedences for WPO and the integer arithmetic solver for max-polynomial interpretations.

References

1 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000. `doi:10.1016/S0304-3975(99)00207-8`.

2 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998. `doi:10.1017/CBO9781139172752`.

3 Clemens Ballarin. Locales: A module system for mathematical theories. *J. Autom. Reasoning*, 52(2):123–153, 2014. `doi:10.1007/s10817-013-9284-7`.

4 Amir M. Ben-Amram and Michael Codish. A SAT-based approach to size change termination with global ranking functions. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume

4963 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2008. doi:10.1007/978-3-540-78800-3_16.

5 Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07964-5.

6 Frédéric Blanqui and Adam Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Math. Struct. Comput. Sci.*, 21(4):827–859, 2011. doi:10.1017/S0960129511000120.

7 Ralph Bottesch, Max W. Haslbeck, Alban Reynaud, and René Thiemann. Verifying a solver for linear mixed integer arithmetic in Isabelle/HOL. In *NASA Formal Methods Symposium, 12th International Conference, Proceedings*, 2020. To appear.

8 Marc Brockschmidt, Sebastiaan J. C. Joosten, René Thiemann, and Akihisa Yamada. Certifying safety and termination proofs for integer transition systems. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 454–471. Springer, 2017. doi:10.1007/978-3-319-63046-5_28.

9 Michael Codish, Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann. SAT solving for termination proofs with recursive path orders and dependency pairs. *J. Autom. Reasoning*, 49(1):53–93, 2012. doi:10.1007/s10817-010-9211-0.

10 Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of automated termination proofs. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2007. doi:10.1007/978-3-540-74621-8_10.

11 Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Automated certified proofs with CiME3. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*, pages 21–30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.RTA.2011.21.

12 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

13 Nachum Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17:279–301, 1982. doi:10.1016/0304-3975(82)90026-3.

14 Nachum Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1/2):69–116, 1987. doi:10.1016/S0747-7171(87)80022-6.

15 Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reasoning*, 40(2-3):195–220, 2008. doi:10.1007/s10817-007-9087-9.

16 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. Maximal termination. In Andrei Voronkov, editor, *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings*, volume 5117 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2008. doi:10.1007/978-3-540-70590-1_8.

17 Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with aprove. *J. Autom. Reasoning*, 58(1):3–31, 2017. doi:10.1007/s10817-016-9388-y.

18 Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The Termination and Complexity Competition. In Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*, volume 11429 of *Lecture Notes in Computer Science*, pages 156–166. Springer, 2019. doi:10.1007/978-3-030-17502-3_10.

19 Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331. Springer, 2004. doi:10.1007/978-3-540-32275-7_21.

20 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *J. Autom. Reasoning*, 37(3):155–203, 2006. doi:10.1007/s10817-006-9057-7.

21 Nao Hirokawa and Aart Middeldorp. Tsukuba Termination Tool. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings*, volume 2706 of *Lecture Notes in Computer Science*, pages 311–320. Springer, 2003. doi:10.1007/3-540-44881-0_22.

22 Nao Hirokawa and Aart Middeldorp. Polynomial interpretations with negative coefficients. In Bruno Buchberger and John A. Campbell, editors, *Artificial Intelligence and Symbolic Computation, 7th International Conference, AISC 2004, Linz, Austria, September 22-24, 2004, Proceedings*, volume 3249 of *Lecture Notes in Computer Science*, pages 185–198. Springer, 2004. doi:10.1007/978-3-540-30210-0_16.

23 Nao Hirokawa, Aart Middeldorp, and Harald Zankl. Uncurrying for termination and complexity. *J. Autom. Reasoning*, 50(3):279–315, 2013. doi:10.1007/s10817-012-9248-3.

24 Jan Jakubuv and Cezary Kaliszyk. Towards a unified ordering for superposition-based automated reasoning. In James H. Davenport, Manuel Kauers, George Labahn, and Josef Urban, editors, *Mathematical Software - ICMS 2018 - 6th International Conference, South Bend, IN, USA, July 24-27, 2018, Proceedings*, volume 10931 of *Lecture Notes in Computer Science*, pages 245–254. Springer, 2018. doi:10.1007/978-3-319-96418-8_29.

25 Sam Kamin and Jean-Jacques Lévy. Two generalizations of the recursive path ordering, 1980. Unpublished note.

26 Donald E. Knuth and Peter Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, New York, 1970. doi:10.1016/B978-0-08-012975-4.50028-X.

27 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In Ralf Treinen, editor, *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304. Springer, 2009. doi:10.1007/978-3-642-02348-4_21.

28 Alexander Krauss. Recursive definitions of monadic functions. In Ekaterina Komendantskaya, Ana Bove, and Milad Niqui, editors, *Partiality and Recursion in Interactive Theorem Provers, PAR@ITP 2010, Edinburgh, UK, July 15, 2010*, volume 5 of *EPiC Series*, pages 1–13. EasyChair, 2010. URL: http://www.easychair.org/publications/paper/52847.

29 Keiichirou Kusakari, Masaki Nakamura, and Yoshihito Toyama. Argument filtering transformation. In Gopalan Nadathur, editor, *Principles and Practice of Declarative Programming, International Conference PPDP'99, Paris, France, September 29 - October 1, 1999, Proceedings*, volume 1702 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 1999. doi:10.1007/10704567_3.

30 Dallas Lankford. Canonical algebraic simplification in computational logic. Technical Report ATP-25, University of Texas, 1975.

31 Dallas Lankford. On proving term rewrite systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

32 Salvador Lucas. MU-TERM: A tool for proving termination of context-sensitive rewriting. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2004. doi:10.1007/978-3-540-25979-4_14.

33 Zohar Manna and Stephen Ness. On the termination of Markov algorithms. In *3rd Hawaii International Conference on System Science*, pages 789–792, 1970.

34 Aart Middeldorp. Approximating dependency graphs using tree automata techniques. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 593–610. Springer, 2001. doi:10.1007/3-540-45744-5_49.

35 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-45949-9.

36 Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel. A verified prover based on ordered resolution. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 152–165. ACM, 2019. doi:10.1145/3293880.3294100.

37 Peter Schneider-Kamp, René Thiemann, Elena Annov, Michael Codish, and Jürgen Giesl. Proving termination using recursive path orders and SAT solving. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2007. doi:10.1007/978-3-540-74621-8_18.

38 Jonas Schöpf and Christian Sternagel. $\mathsf{T_TT_2}$ with termination templates for teaching. In *Proc. of the 16th International Workshop on Termination*, 2018. URL: http://arxiv.org/abs/1806.05040.

39 Christian Sternagel and René Thiemann. Generalized and formalized uncurrying. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *Frontiers of Combining Systems, 8th International Symposium, FroCoS 2011, Saarbrücken, Germany, October 5-7, 2011. Proceedings*, volume 6989 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2011. doi:10.1007/978-3-642-24364-6_17.

40 Christian Sternagel and René Thiemann. Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands*, volume 21 of *LIPIcs*, pages 287–302. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.RTA.2013.287.

41 Christian Sternagel and René Thiemann. The certification problem format. In Christoph Benzmüller and Bruno Woltzenlogel Paleo, editors, *Proceedings Eleventh Workshop on User Interfaces for Theorem Provers, UITP 2014, Vienna, Austria, 17th July 2014*, volume 167 of *EPTCS*, pages 61–72, 2014. doi:10.4204/EPTCS.167.8.

42 Christian Sternagel and René Thiemann. Formalizing monotone algebras for certification of termination and complexity proofs. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8560 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2014. doi:10.1007/978-3-319-08918-8_30.

43 Christian Sternagel and René Thiemann. XML. *Archive of Formal Proofs*, October 2014. , Formal proof development. URL: http://isa-afp.org/entries/XML.html.

44 Christian Sternagel and Akihisa Yamada. Reachability analysis for termination and confluence of rewriting. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 262–278. Springer, 2019. `doi:10.1007/978-3-030-17462-0_15`.

45 Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A cross-community infrastructure for logic solving. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 367–373. Springer, 2014. `doi:10.1007/978-3-319-08587-6_28`.

46 René Thiemann, Guillaume Allais, and Julian Nagele. On the Formalization of Termination Techniques based on Multiset Orderings. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPIcs*, pages 339–354. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. `doi:10.4230/LIPIcs.RTA.2012.339`.

47 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009. `doi:10.1007/978-3-642-03359-9_31`.

48 Xavier Urbain. Modular and incremental automated termination proofs. *J. Autom. Reasoning*, 32(4):315–355, 2004. `doi:10.1007/BF03177743`.

49 Johannes Waldmann. Matchbox: A tool for match-bounded string rewriting. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94. Springer, 2004. `doi:10.1007/978-3-540-25979-4_6`.

50 Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Partial status for KBO. In Johannes Waldmann, editor, *13th International Workshop on Termination (WST 2013)*, pages 74–78, 2013.

51 Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Unifying the Knuth-Bendix, recursive path and polynomial orders. In Ricardo Peña and Tom Schrijvers, editors, *15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013*, pages 181–192. ACM, 2013. `doi:10.1145/2505879.2505885`.

52 Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Nagoya Termination Tool. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8560 of *Lecture Notes in Computer Science*, pages 466–475. Springer, 2014. `doi:10.1007/978-3-319-08918-8_32`.

53 Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. A unified ordering for termination proving. *Sci. Comput. Program.*, 111:110–134, 2015. `doi:10.1016/j.scico.2014.07.009`.

54 Harald Zankl, Nao Hirokawa, and Aart Middeldorp. Constraints for argument filterings. In Jan van Leeuwen, Giuseppe F. Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and Frantisek Plasil, editors, *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings*, volume 4362 of *Lecture Notes in Computer Science*, pages 579–590. Springer, 2007. `doi:10.1007/978-3-540-69507-3_50`.

55 Harald Zankl, Nao Hirokawa, and Aart Middeldorp. KBO orientability. *J. Autom. Reasoning*, 43(2):173–201, 2009. `doi:10.1007/s10817-009-9131-z`.

56 Harald Zankl and Aart Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 481–500. Springer, 2010. `doi:10.1007/978-3-642-17511-4_27`.

57 Hans Zantema. Termination of string rewriting proved automatically. *J. Autom. Reasoning*, 34(2):105–139, 2005. `doi:10.1007/s10817-005-6545-0`.