Certifying the Weighted Path Order

René Thiemann (©)
University of Innsbruck, Austria

Jonas Schöpf (©)
University of Innsbruck, Austria

Christian Sternagel (©)
DVT, Austria

Akihisa Yamada (©)
National Institute of Advanced Industrial Science and Technology, 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 TTT2, 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.CVIT.2016.23

Category invited paper

Funding René Thiemann: He was supported by the Austrian Science Fund (FWF) project Y757.
Jonas Schöpf: He was supported by FWF project P27502.
Christian Sternagel: He was supported by FWF project P27502.
Akihisa Yamada: A part of the work was carried out while he was working in Innsbruck under support by FWF projects Y757 and P27502.

Acknowledgements We thank Sarah Winkler for her assistance and discussions on TTT2 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 Nagoya Termination Tool (NaTT) [52] was originally developed solely to demonstrate the power of WPO. It participated in the full run of the 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, Isabelle Formalization of Rewriting, is an Isabelle/HOL [35]-formalized library of correctness proofs of analysis techniques for term rewriting and transition systems, and the latter, Certified Tool Assertions, 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 a new XML parser implemented in Isabelle/HOL and the format for certificates of 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 Tyrolean Termination Tool 2 (TTT2) [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 are plenty of work on orders for proving termination of rewriting. The earliest of such we know is the Knuth–Bendix order (KBO), introduced along with the Knuth–Bendix completion in their celebrated paper in 1970 [26]. In the same year, Manna and Ness [33] proposed a semantic approach, which nowadays is called interpretation methods. One instantiation of
the approach is Lankford’s polynomial interpretations [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. These
tools have been evaluated in the Termination Competition [18] since 2004. These develop-
ments, however, revealed that tool implementations are not blindly trustable: sometimes 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. Besides
the choice of proof assistants, a significant difference 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 a few related work outside pure term rewriting. Recently
a verified ordered resolution prover [36] has been developed as part of the IsaFoR project, the
Isabelle Formalization of Logic. Currently the verified prover is based on KBO, which can
be replaced by stronger and more general WPO. In fact, WPO is already utilized in the E
theorem prover [24].

In a 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 now heavily utilize in the 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 $f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}$ is $n$-ary and $t_1, \ldots, t_n$ 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 $\sqsupseteq$
is defined by $C[t] \sqsupseteq t$. A substitution is a function $\sigma$ from variables to terms, and we write
$\sigma t$ 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 $\ell$ and $r$ indicating that an
instance of $\ell$ 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 $(\succ, \succcurlyeq)$ of two relations on terms that satisfies the following
requirements: $\succ$ is well-founded, $\succcurlyeq$ and $\succ$ are compatible (i.e., $\preceq \circ \prec \circ \preceq \subseteq \succ$), both
are closed under substitutions, and $\preceq$ is closed under contexts. If $\succ$ is also closed under
context, then we call $(\succ, \preceq)$ a monotone reduction pair; a transitive relation $\succ$ of a monotone
reduction pair is called reduction order and used to directly prove termination by $\mathcal{R} \subseteq \succ$,
while reduction pairs are employed for termination proofs with dependency pairs. We write
$\succ_{\text{lex}}$ and $\succ_{\text{mul}}$ for the lexicographic and multiset extension induced by $(\succ, \preceq)$, respectively.
A weakly monotone \((F,\cdot)\)-algebra \(A\) is a well-founded ordered set \((A,\succ)\) equipped with an interpretation \(f_A : A^n \to A\) for every \(n\)-ary \(f \in F\), such that \(f_A(a_1,\ldots,a_n) \geq f_A(b_1,\ldots,b_n)\) whenever \(a \geq b\). Any weakly monotone algebra \(A\) induces a reduction pair \((\succ_A,\succeq_A)\) defined by \(s \succeq_A t\) if \(s,A \geq [t,A]\) for all assignments \(A\). Here, \([t,A]\) denotes term evaluation in the algebra with respect to an assignment \(A\).

A \((partial)\) status is a mapping \(\pi\) which assigns to each \(n\)-ary symbol \(f\) a list \(\pi(f) = [i_1,\ldots,i_m]\) of indices in \(\{1,\ldots,n\}\). Abusing notation, we also see \(\pi(f)\) as the set \(\{i_1,\ldots,i_m\}\), and as an operation on \(n\)-ary lists defined by \(\pi(f)[1_1,\ldots,1_n] = [1_1,\ldots,1_{im}]\).

A binary relation \(\succ\) over terms is simple with respect to status \(\pi\), if \(f(t_1,\ldots,t_n) \succ t_i\) 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 \(\succ\) on \(F\), such that \(\succ = \succ \setminus \preceq\) is well-founded.

**Definition 1** (WPO [53, Def. 10, incl. Refinements (2c) and (2d) of Sect. 4.2]). Let \(A\) be a weakly monotone algebra, \(\preceq\) a precedence, and \(\pi\) a status. Let \(\succeq_A\) be simple with respect to \(\pi\). The WPO reduction pair \((\succ_{\text{WPO}},\succeq_{\text{WPO}})\) is defined as follows: \(s \succ_{\text{WPO}} t\) iff

1. \(s \succ_A t\), or
2. \(s \succeq_A t\) and
   a. \(s = f(s_1,\ldots,s_n)\) and \(\exists i \in \pi(f)\). \(s_i \succ_{\text{WPO}} t_i\), or
   b. \(s = f(s_1,\ldots,s_n), t = g(t_1,\ldots,t_m)\), \(\forall j \in \pi(g)\). \(s \succ_{\text{WPO}} t_j\) and
      i. \(f \succ g\) or
      ii. \(f \succeq g\) and \(\pi(f)[s_1,\ldots,s_n] >_{\text{WPO}} \pi(g)[t_1,\ldots,t_m]\).

The relation \(s \succeq_{\text{WPO}} t\) is defined in the same way, where \(>_{\text{lex}}\) in the last line is replaced by \(>_{\text{lex}}\), and there are the following additional subcases in case 2:

3. \(s \in V\) and either \(s = t\) or \(t = g(t_1,\ldots,t_m)\), \(\pi(g) = \emptyset\) and \(g\) is least in precedence,
4. \(s = f(s_1,\ldots,s_n), t \in V, \succ_A\) is simple w.r.t. \(\pi\), and \(\forall g.f \succ g \lor (f \succeq g \land \pi(g) = \emptyset)\).

**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 \(s \succeq_{\text{WPO}} t\) 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, \('<f,\nu>'\)\text{term}, where \('f'\) and \('\nu'\) are type-variables representing the signature \(F\) and the set of variables \(V\), respectively. A typing judgement is of the form \(\text{term} :: \text{type}\). As an example, \(R :: ('f,\nu')\text{term rel}\) states that \(R\) has type \('<f,\nu>'\)\text{term rel}, i.e., \(R\) is a binary relation over terms.

An Isabelle \textit{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 \texttt{redpair}.\footnote{In \texttt{IsaFoR}, there is a more general locale for reduction \texttt{triples (redtriple)}, which we simplify to reduction \texttt{pairs} in the presentation of this paper.} Here, \(O\) is relation composition, and \(SN\) is a predicate for well-foundedness (strong normalization).

\texttt{locale redpair =}
\texttt{fixes \(S\ NS \:: \text{"(f.v)term rel"} \)}
\texttt{assumes \text{"SN \(S"\)}}
\texttt{\quad and \text{"ctx.closed \(NS\"}}
\texttt{\quad and \text{"subst.closed \(S\" and \text{"subst.closed \(NS\"}}
\texttt{\quad and \text{"\(NS\ O \(S\ \subseteq \(S\)}}\quad \text{and \text{"\(S\ O \(NS\ \subseteq \(S\)}}

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

\texttt{locale redpair_order = redpair \(S\ NS \)} +
\texttt{\quad assumes \text{"trans \(S" and \text{"trans \(NS" and \text{"refl \(NS\")}}}}

\begin{itemize}
  \item Beside the abstract definitions for reduction pairs, \texttt{IsaFoR} also provides several instances of them, e.g., one for RPO, one for KBO \cite{KBO}, 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 \cite{UsableRulesRefinement, UsableRulesRefinement2} requires \(C_e\)-compatible reduction pairs and argument filters. To this end \texttt{IsaFoR} contains the locale \texttt{ce_af_redpair_order}. It extends \texttt{redpair_order} by a new parameter \(\pi\) for the argument filter, and demands the additional requirements.

\texttt{locale ce_af_redpair_order = redpair_order \(S\ NS \)} +
\texttt{\quad fixes \(\pi \:: \text{"f af"} \)}
\texttt{\quad assumes \text{"af Compatible \(\pi \(NS\"}}
\texttt{\quad and \text{"ce Compatible \(NS\")}}
\end{itemize}

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

\section{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 \texttt{IsaFoR}. Afterwards we illustrate our verified implementation for checking WPO constraints in Section 3.2.

\subsection{Properties of WPO}

As we have seen in Section 2.2, \texttt{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, \texttt{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 is just adding the missing pieces. To be more precise, one could have formalized weakly monotone algebras in IsaFoR and then on top of that formally verify the properties of WPO. However, this has the disadvantage that also instances of weakly monotone algebras already formalized in IsaFoR would have to be adjusted to the new interface; this would be polynomial interpretations, arctic interpretations, and matrix interpretations.

Therefore, we choose a different approach, namely we only reformulate the definition of WPO, so that it does not depend on the notion of weakly monotone algebras anymore, but instead uses reduction pairs directly, cf. Definition 3.

► Definition 3 (WPO based on Reduction Pairs). Let \( (>_{A}, \succeq_{A}) \) be a reduction pair, \( \succeq \) a precedence, \( \ldots \) and continue as in Definition 1 to define the relations \( >_{WPO} \) and \( \succeq_{WPO} \).

In this way, all instances of reduction pairs in IsaFoR immediately become available as parameter to WPO, i.e., one can parametrize WPO with (max-)polynomial interpretations and matrix interpretations as it is already done in the literature, but it is also possible to use KBO or RPO as parameter to WPO, or one can even 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 it has been formalized in IsaFoR). Let \( \succeq \) be a precedence. Let \( \sigma \) be a function of type \( F \to \{ \text{lex, mul} \} \). We define the RPO reduction pair \( (>_{RPO}, \succeq_{RPO}) \) as follows: \( s >_{RPO} t \) iff

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

The relation \( s \succeq_{RPO} t \) is defined in the same way, where \( \succ_{RPO} \sigma(f) \) in case (ii) is replaced by \( \succeq_{RPO} \sigma(f) \), where \( n > 0 \) in case (iii) is dropped, and there is one additional subcase:

- c. \( s \in V \) and either \( s = t \) or \( t = c \) where \( c \) is a constant in \( F \) 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 compilable Isabelle file which defines RPO although everything is named WPO.

Next, we include a couple of modifications of the definition, so that eventually the WPO of Definition 3 is defined formally. For each modification, we immediately adjust the formal proofs. These adjustments have mostly been straight-forward, also because of the valuable support by the proof assistant: we were immediately pointed to those parts of the proofs which got broken by a modification, without the necessity of manual rechecking those proofs that did not require an adjustment.

To be more precise, we perform the following sequence of modifications.
We delete \( \pi \) from RPO and replace it by \( \text{lex} \), as the choice of multiset or lexicographic comparison via \( \sigma \) is not present in WPO. As a result, case (iii) is dropped, case (ii) always uses lexicographic comparison, and the formal proofs become shorter.

We add the two tests \( s \geq_A t \) and \( s >_A t \) that are present in WPO, but not in RPO. At this point we add the requirement of WPO, that \( \geq_A \) must be simple, in order to adjust all the proofs of the defined relations.

We include the status \( \pi \), which is present in the WPO definition, but not in RPO. In this step we also weaken the requirement of \( \geq_A \) being simple to the requirement that \( \geq_A \) is simple with respect to \( \pi \).

We generalize rule (2c) of RPO in such a way that not only for constants \( c \) we permit \( x \geq_{\text{WPO}} c \), but also \( x \geq_{\text{WPO}} g(t_1, \ldots, t_n) \) is possible if \( \pi(g) = \emptyset \).

We finally add refinement (2d) under the premise that \( >_A \) is simple w.r.t. \( \pi \). 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 \( (\geq_{\text{WPO}}, >_{\text{WPO}}) \), where for instance the transitivity proof of \( >_{\text{WPO}} \) got broken and could not be repaired. Actually, we figured out that \( >_{\text{WPO}} \) is no longer transitive with this refinement, cf. 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.

\[ \text{Example 5.} \] Consider \( F = \{ a \} \), \( \pi(a) = \emptyset \), and a reduction pair (or algebra) where \( \geq_A \) relates all terms and \( >_A \) is empty. Then \( x \geq_{\text{WPO}} a \geq_{\text{WPO}} y \), but \( x \geq_{\text{WPO}} y \) 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. Precedence \( \succsim \) is specified in form of three functions \( \text{prc}, \text{pr}_{\text{least}}, \text{pr}_{\text{large}} \): \( \text{prc} \) takes two symbols \( f \) and \( g \) and returns a pair of Booleans \( (f \succ g, f \succsim g) \); \( \text{pr}_{\text{least}} \) is a predicate telling a symbol is least \( \succsim \) or not; and \( \text{pr}_{\text{large}} \) states whether a symbol is largest \( \succsim \) with respect to \( \pi \) or not, as required in rule (2d) of Definition 1. Whereas most of the properties of the precedence are encoded via an existing locale \( \text{precedence} \), for a symbol being of \( \text{largest} \) precedence we add two new additional assumptions explicitly. In the locale we further use a Boolean \( \text{ssimple} \) to indicate whether \( >_A \) is simple with respect to \( \pi \), i.e., whether it is allowed to apply rule (2d) or not. Only then, the properties of \( \text{pr}_{\text{large}} \) must be satisfied and the degenerate case must be excluded. Being simple with respect to \( \pi \) is enforced via the predicate \( \text{simple}_{\text{arg_pos}} \) for any relation \( R \) the property \( \text{simple}_{\text{arg_pos}} \ R \ f \ i \) ensures that \( f(t_1, \ldots, t_n) R t_i \) must hold for all \( t_1, \ldots, t_n \).

\[ \text{locale wpo_params = redpair_order S NS + precedence prc pr}_{\text{least}} \]

\[ \text{for S NS :: "}(f, \_') \) \text{term rel"} \quad (* \text{underlying reduction pair } *) \]

\[ \text{and prc :: "}(f \Rightarrow f \Rightarrow \text{bool } \times \text{bool}) \quad \text{and pr}_{\text{least}} \text{ pr}_{\text{large}} :: "f \Rightarrow \text{bool}" (* \text{precedence } *) \]

\[ \text{and ssimple :: bool} \quad (* \text{flag whether rule (2d) is permitted } *) \]

\[ \text{and } \pi :: "f \text{ status }" \quad (* \text{status } *) \]

\[ \text{assumes "S } \subseteq \text{ NS"} \]

\[ \text{and } "i \in \pi f \Rightarrow \text{simple}_{\text{arg_pos}} \text{ NS } f \ i" \quad (* \text{NS is simple w.r.t. } \pi *) \]

\[ \text{and } "\text{ssimple } \Rightarrow i \in \pi f \Rightarrow \text{simple}_{\text{arg_pos}} \text{ S } f \ i" \quad (* \text{S is simple w.r.t. } \pi *) \]

\[ \text{and } "\text{ssimple } \Rightarrow \text{NS } \neq \text{UNIV}" \quad (* \text{exclude degenerate case } *) \]

\[ \text{and } "\text{ssimple } \Rightarrow \text{pr}_{\text{large }} f \Rightarrow \text{fst } (\text{prc } f \ g) \lor \text{snd } (\text{prc } f \ g) \wedge \pi g = \emptyset" \]

\[ \text{and } "\text{ssimple } \Rightarrow \text{pr}_{\text{large }} f \Rightarrow \text{snd } (\text{prc } g \ f) \Rightarrow \text{pr}_{\text{large }} g" \]
Within the locale we define the relations $\text{WPO}_S$ and $\text{WPO}_\text{NS}$ ($\succsim\text{WPO}$ and $\succeq\text{WPO}$ of Definition 3) with the help of a recursive function, and prove the main theorem:

**Theorem** "redpair_order $\text{WPO}_S\text{ WPO}_\text{NS}"

Moreover, we prove that whenever the non-strict relation is compatible with an argument filter $\mu$ 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 $\mu$ NS"

**Shows** "af_compatible $(\pi \cup \mu)$ WPO_NS"

We further prove that WPO is also $\mathcal{C}_c$-compatible under mild preconditions, namely we just require that $\pi(f)$ includes the first two positions of some symbol $f$. In summary, 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 $\mu$ NS"

**Shows** "ce_af_redpair_order $\text{WPO}_S\text{ WPO}_\text{NS} (\pi \cup \mu)$"

At the moment, our formalization does not cover any comparison to other term orders, e.g., there is no formal statement that each polynomial KBO can be formulated as an instance of a 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 $s \succ_\mathcal{A} t$ and $s \succeq_\mathcal{A} t$ 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)\text{term} \Rightarrow (f, v)\text{term} \Rightarrow \text{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 $\text{rp}$ and for given finite sets of strict- and non-strict-constraints, represented as two lists
\begin{verbatim}
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 the formal statement, set is Isabelle's function to 
convert a list into a set.

assumes "redpair.valid rp" (* generic_reduction_pair *)
    and "\forall (s,t) \in set S_list. redpair.check_S rp s t"
    and "\forall (s,t) \in set NS_list. redpair.check_NS rp s t"

shows "\exists S NS.
    ce_af_redpair_order S NS (redpair.af rp) \&
    set S_list \subseteq S \& set NS_list \subseteq NS \&
    (redpair.mono rp \longrightarrow ctxt.closed S)"

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

For checking validity of WPO, we assert \(\text{redpair.valid rp}\) and in addition perform checks 
that the status \(\pi\) is well-defined, i.e., \(\pi(f) \subseteq \{1, \ldots, n\}\) 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 \(\text{wpo_params}\)-locale. We further set the argument filter of WPO 
to \(\pi \cup \text{redpair.af af}\).

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

We solve this problem by enlarging the record \(\text{redpair}\) by two new entries for strict and 
weak simplicity, and require in \(\text{generic_reduction_pair}\) that if these flags are enabled, then 
the relations \(S\) and \(NS\) must be simple with respect to \(\pi\), 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 \(\text{IsaFoR}\), e.g., polynomial interpretations, etc., 
to provide the new information. To be more precise, we formalize two sufficient criteria 
for each reduction pair in \(\text{IsaFoR}\), 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 \(\text{redpair.check_S rp}\) and 
\(\text{redpair.check_NS rp}\), respectively.

Next, we are given two lists of constraints \(\text{wpo_S_list}\) and \(\text{wpo_NS_list}\) 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:

\[S\_list = \{(s, t_i) \mid (s, t) \in \text{wpo_S_list} \cup \text{wpo_NS_list}, s \succ s_i, t \succ t_i, \text{redpair.check_S rp } s_i t_i\}\]

After these lists have been defined, we apply \(\text{generic_reduction_pair}\) 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 \(\text{wpo_params}\)-locale and get access to the reduction pair \(\text{WPO}_S\) and 
\(\text{WPO}_NS\). We further know that the approximations in \(S\_list\) and \(NS\_list\) are correct, e.g., 
whenever \((s, t) \in \text{wpo_S_list} \cup \text{wpo_NS_list}, s \succ s_i, t \succ t_i\) and \(\text{redpair.check_S rp } s_i t_i\) 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 \(\text{wpo_S_list} \cup \text{wpo_NS_list}\). So, we have
\end{verbatim}
a reduction pair \( \text{WPO}_S \) and \( \text{WPO}_{\text{NS}} \) and an approximation statement, as required by \texttt{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 \texttt{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 \texttt{IsaFoR} and supported by \texttt{CeTA}. However, max-polynomial interpretations [16] were not yet supported. Hence we extend \texttt{IsaFoR} and \texttt{CeTA} to incorporate them, in particular those over natural numbers as required by WPO instances introduced in [53].

In order for \texttt{CeTA} to certify proofs using max-polynomial interpretations, we must formally prove that the pair of relations \((\succ_A, \succeq_A)\) forms a reduction pair, and implement a verifier to check \( s \succ_A t \) and \( s \succeq_A t \). 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 into the validity of an integer arithmetic formula without max, for which we have a formalized validity checker already [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 \texttt{IsaFoR} as terms of the following signature.

\[ \texttt{datatype sig = ConstF nat | SumF | ProdF | MaxF} \]

The interpretation of the symbols are as expected:

\[ \texttt{primrec I where} \]

\begin{align*}
&\texttt{"I (ConstF n) = (\lambda x. n)"} \\
&\texttt{"I SumF = sum_list"} \\
&\texttt{"I ProdF = prod_list"} \\
&\texttt{"I MaxF = max_list"} 
\end{align*}

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

\begin{align*}
\text{max}(x, y) + z & \rightarrow \text{max}(x + z, y + z) \\
\text{max}(x, y) \cdot z & \rightarrow \text{max}(x \cdot z, y \cdot z) \\
\text{max}(x, y, z) & \rightarrow \text{max}(x + y, x + z) \\
\text{max}(x, y, z) & \rightarrow \text{max}(x \cdot y, x \cdot z)
\end{align*}

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 \( \text{max}_{i=1}^{n} s_i \) and \( \text{max}_{i=1}^{m} t_i \), where \( s_1, \ldots, s_n \) and \( t_1, \ldots, t_m \) are polynomials (without max). In \texttt{IsaFoR} we define the mapping from \( s \) to \( s_1, \ldots, s_n \) as \texttt{to_IA}. Then the comparison of two such normal forms is easily translated to an arithmetic formula without max, cf. [4]:

\[ s \leq t \iff \max_{i=1}^{n} s_i \leq \max_{j=1}^{m} t_j \iff \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} s_i \leq_{j} t_j \]

This reduction is formalized in Isabelle as follows. Here, operators with subscript "\( \text{IA} \)" build syntactic formulas, and those with prefix "\( \text{IA} \)" or subscript "\( \text{IA} \)" come from the formalization of integer arithmetic; e.g., "\( \bigwedge_{x} \ exists \. \ const \ 0 \leq_{\text{IA}} \ text{var} \ x \)" denotes an integer arithmetic formula representing "\( 0 \leq x_1 \land \cdots \land 0 \leq x_n \)". 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 add these assumptions and consequently, several valid termination proofs could not be certified.
\begin{verbatim}
definition le_via_IA where "le_via_IA s t \equiv
  (\IA x \leftarrow vars_term_list s @ vars_term_list t. IA.const 0 \leq IA var x) \longrightarrow_{IA}
  (\IA s i \leftarrow to_IA s. \forall t j \leftarrow to_IA t. s_i \leq IA t_j)"

definition less_via_IA where "less_via_IA s t \equiv
  (\IA x \leftarrow vars_term_list s @ vars_term_list t. IA.const 0 \leq IA var x) \longrightarrow_{IA}
  (\IA s i \leftarrow to_IA s. \forall t j \leftarrow to_IA t. s_i < IA t_j)"

deﬁnition le_via_IA where "le_via_IA s t \equiv
  (\IA x \leftarrow vars_term_list s @ vars_term_list t. IA.const 0 \leq IA var x) \longrightarrow_{IA}
  (\IA s i \leftarrow to_IA s. \forall t j \leftarrow to_IA t. s_i \leq IA t_j)"

definition less_via_IA where "less_via_IA s t \equiv
  (\IA x \leftarrow vars_term_list s @ vars_term_list t. IA.const 0 \leq IA var x) \longrightarrow_{IA}
  (\IA s i \leftarrow to_IA s. \forall t j \leftarrow to_IA t. s_i < IA t_j)"

The soundness of the reduction is formally proved as follows.

lemma le_via_IA:
  assumes "|=\IA le_via_IA s t" shows "s \leq_{\IA} t"

lemma less_via_IA:
  assumes "|=\IA less_via_IA s t" shows "s <_{\IA} 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
securely validate the comparisons s \geq_{\IA} t and s >_{\IA} t, respectively.

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

5 Certificate Format and Parser

The Certification Problem Format (CPF) [41] is a machine-readable XML format, which was
developed in the term rewriting community to serve as the standard communication language
between verification tools and certification tools developed in various research groups.

Here we present the addition to the CPF made in the current work, namely the certificates
for WPO and max-polynomial interpretations. To this end, we also changed the structure of
the parser in CeTA, since it had been relying on an XML parser in Isabelle/HOL [43], which
had several limitations. In the current work we develop a more concise and flexible XML
parser library, which allows notations like Haskell’s do notation.

Notation "\XMLdo s {...}" constructs a parser for an XML element whose tag is s. An
element parser is of type 'a xmlt2, which is a function from internal representations of an
XML element to the direct sum of type 'a and an error state. Inside an XMLdo block, one can
parse inner elements by binding "x \leftarrow inner;" or its variants such as "xs \leftarrow{\ell.u} inner;"
which binds xs as the list of at least \ell and at most \ell repeated inner elements. Here u is of
type enat, so that it can be \infty; the frequent instance \leftrightarrow{\{0..\infty\}} is also written \leftrightarrow{\infty}. Typically
a parser block should end with "xml_return r;", where r is the return value expressed with
previously bound variables. This invocation also checks if there are no elements left to be
parsed, in order for the parser to precisely define a grammar.

Given parsers \(p_1\) and \(p_2\) for two kinds of elements, we allow choices between them by
"p1 XMLor p2". It works as follows: if parser \(p_1\) returns a recoverable error state, then it
tries \(p_2\). Here recoverable means that the tag of the root element is not handled by \(p_1\). If \(p_1\)
handled the root element but failed in inner elements, then it goes to an unrecoverable error
state.

In the following we present some important parsers from this work, and by that specify
XML grammars. Until a certain moment of the development we stated all parsers using
Isabelle’s command fun that specifies a terminating function. However, the automatic termina-
ation proving of fun turned out excessively slow for the parser of entire CPF. Therefore,
we now define our parsers via the `partial_function` command, which does not require termination proofs, so that processing is much faster.

A first concrete example is a parser for expressions 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.

```isar
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)}XMLdo (STR "sum") {exps ←* exp_parser; xml_return (Fun max_poly.SumF exps)}XMLdo (STR "max") {exps ←{1..∞} exp_parser; xml_return (Fun max_poly.MaxF exps)}XMLdo (STR "constant") {n ←nat; xml_return (max_poly.const n)}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 (turned into zero-based indexing).

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

```isar
partial_function (sum_bot) redtriple :: "a redtriple_impl xmlt2" where
[...]
```

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 \(<\text{weightedPathOrder}>\) element characterizes a concrete WPO reduction pair. It consists of WPO specific parameters \(\text{wpo}_\text{params}\) that fixes status and precedences, and another reduction pair in a recursive manner, which specifies the “algebra” \(A\) in terms of \((\succ_A, \succeq_A)\). The \(<\text{filteredRedPair}>\) 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 \(\text{CeTA}\) by WPO and max-polynomial interpretations, we implement certificate output for WPO in two termination analyzers: \(\text{NaTT}\) and \(\text{TTT}_2\).

\(\text{NaTT}\) originates as an experimental implementation of WPO [51]. From its early design \(\text{NaTT}\) 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, \(\text{NaTT}\) utilizes incremental SMT solving, and implements some tricks for efficiency [52]. In the current work, its output is adjusted to confine to the newly defined XML certificate format for WPO, max-polynomials, and collapsing argument filters. These are essentially the central techniques implemented in \(\text{NaTT}\), but a few techniques implemented later on in \(\text{NaTT}\) had to be deactivated to be able to be certified by \(\text{CeTA}\); some of them, such as nontermination proofs, are actually supported but \(\text{NaTT}\) is not yet adjusted to produce certificates for them.

\(\text{TTT}_2\) succeeded the automated termination analyzer \(\text{TTT}_2\) 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 \(\text{TTT}_2\) by an implementation of WPO, following mostly the presented encodings in [53]. A notable difference in the search space for max-polynomials: while \(\text{NaTT}\) heuristically chooses between max and sum, \(\text{TTT}_2\) 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 \(\text{TTT}_2\) 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. \(\text{TTT}_2\) provides termination templates [38] where users can fix parts of proofs via parameters when invoking \(\text{TTT}_2\). 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 \(\text{TTT}_2\) and afterwards \(\text{CeTA}\) can validate if the proof is correct.

Example 6. Consider the following TRS (\(\text{Zantema}_05/z10.xml\) of TPDB):

\[
\begin{align*}
  a(\lambda(x,y)) & \rightarrow \lambda(a(x,p(1,a(y,t)))) \\
  a(p(x,y),z) & \rightarrow p(a(x,z),a(y,z)) \\
  a(1,id) & \rightarrow 1 \\
  a(1,p(x,y)) & \rightarrow x \\
  a(t,id) & \rightarrow t \\
  a(t,p(x,y)) & \rightarrow y
\end{align*}
\]

If we just call \(\text{TTT}_2\) with WPO \((\checkmark^2)\) 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

\[2\] The link in this icon directs to the web interface of \(\text{TTT}_2\), preloaded with this example.
Table 1 Certification Experiments

<table>
<thead>
<tr>
<th>Tool</th>
<th>Yes</th>
<th>No</th>
<th>Time (tool)</th>
<th>Time (CeTA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaTT</td>
<td>751</td>
<td>0</td>
<td>02:32:01</td>
<td>00:13:31</td>
</tr>
<tr>
<td>NaTT w/ WPO certifiable</td>
<td>754</td>
<td>194</td>
<td>1d 10:32:00</td>
<td>00:07:43</td>
</tr>
<tr>
<td>NaTT w/o WPO certifiable</td>
<td>751</td>
<td>194</td>
<td>1d 06:31:19</td>
<td>00:03:44</td>
</tr>
<tr>
<td>TTT2</td>
<td>864</td>
<td>169</td>
<td>02:42:55</td>
<td>–</td>
</tr>
<tr>
<td>TTT2 w/ WPO</td>
<td>827</td>
<td>205</td>
<td>13:48:53</td>
<td>–</td>
</tr>
<tr>
<td>TTT2 w/o WPO</td>
<td>827</td>
<td>205</td>
<td>13:45:39</td>
<td>–</td>
</tr>
</tbody>
</table>

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

The flag -cpf enforces proof output via CPF, the flag -msum activates MSum (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 three additional termination proofs in the certifiable setting. 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];
- reachability analysis techniques: for NaTT satisfiability-oriented ones [44], and for TTT2 ones based on tree automata [34];
uncurrying [23]: although the technique itself is fully supported [39], both NaTT and TTT2 have their own variants which exceeds the capability of CeTA.

These observations lead to promising directions of future work. For instance, negative constants seems essentially within our reach in the light of the 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 with 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 its corresponding parser in CeTA. For this parser we have a similar notion as the do-notation in Haskell which makes the parser implementation concise and easy to understand.

The main formal developments in this paper consists 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.

We tested the new version of CeTA with the termination analysis tools NaTT and TTT2 which both have been extended to generate CPF proofs with WPO. All generated proofs have been validated, including those for the 34 TRSs that reportedly only NaTT could prove terminating.

References

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,


