


1 WANDA – a Higher Order Termination Tool

2 Cynthia Kop 

3 Radboud University, Netherlands

4 <https://www.cs.ru.nl/~cynthiakop/>

5 c.kop@cs.ru.nl

6 — Abstract —

7 **Wanda** is a fully automatic termination analysis tool for higher-order term rewriting. In this paper,
8 we will discuss the methodology used in **Wanda**. Most pertinently, this includes a higher-order
9 dependency pair framework and a variation of the higher-order recursive path ordering, as well as
10 some non-termination analysis techniques and delegation to a first-order tool. Additionally, we will
11 discuss **Wanda**'s internal rewriting formalism, and how to use **Wanda** in practice for systems in two
12 different formalisms. We also present experimental results that consider both formalisms.

13 **2012 ACM Subject Classification** 500 – Theory of Computation (Rewrite systems / Equational
14 logic and rewriting)

15 **Keywords and phrases** higher-order term rewriting, termination, automatic analysis, dependency
16 pair framework, higher-order recursive path ordering

17 **Digital Object Identifier** 10.4230/LIPIcs.FSCD.2020.15

18 **Category** System Description

19 **Funding** The author is supported by the NWO TOP project “ICHOR”, NWO 612.001.803/7571.

20 **Acknowledgements** Thanks go to Carsten Fuhs both for proof-reading and for creating a customised
21 version of AProVE which gives an explicit example term for non-termination; to Julian Nagele for
22 using CSI^{ho} to translate the pattern HRSs in COPS to AFSMs; and to the anonymous reviewers
23 of FSCD 2020 whose thorough feedback helped to improve the paper.

24 **1** Introduction

25 Termination of term rewriting systems has been an area of active research for several
26 decades. This concerns not only the analysis of pure term rewriting, but also many variants,
27 such as context-sensitive [52], conditional [41] and higher-order [8] term rewriting. Since the
28 introduction of the annual *International Termination Competition* [13], automated techniques
29 in particular have flourished, with many strong provers competing against each other.

30 Compared to the core area of first-order term rewriting, *higher-order* term rewriting
31 provides some unique challenges, for example due to bound variables. Nevertheless, several
32 tools have participated in the higher-order category of the termination competition (Hot [5],
33 THOR [9], Sol [29], SizeChangeTool [24], Wanda), each using different methods; these include
34 both extensions of first-order techniques like recursive and semantic path orderings [31, 15,
35 30, 10] and dependency pairs [4, 39, 38], and also dedicated methods such as sized types [6].

36 **Wanda**, a tool built primarily around dependency pairs, has participated in this category
37 since 2010 and won most years, including 2019. **Wanda** was also used as a termination
38 back-end in the higher-order category of the 2019 *International Confluence Competition* [12],
39 with both participants (ACPH [45] and CSI^{ho} [43]) delegating termination analysis to **Wanda**.

40 Despite this history, **Wanda** is not well-documented: no tool description has ever been
41 formally published. Implementation choices *are* outlined in the author's PhD thesis [36]
42 alongside termination techniques, but are not easily accessible as understanding these parts
43 requires an understanding of the whole document. This has led to problems, as critical
44 details—such as the rewriting formalism **Wanda** employs, what **Wanda** actually does and how



© Cynthia Kop;

licensed under Creative Commons License CC-BY

5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020).

Editor: Zena M. Ariola; Article No. 15; pp. 15:1–15:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

45 to use Wanda in different configurations or for different styles of rewriting—are hard to find.
46 In addition, there have been substantial updates in recent years.

47 The present work will address this issue by presenting the usage of and the most important
48 techniques used in Wanda. To start, the formalism of higher-order rewriting Wanda uses,
49 AFSMs, is explained in §2, as well as its relation to other popular higher-order formalisms.
50 Then we will discuss the non-termination and termination techniques used in Wanda (§3–5),
51 The paper ends with experimental results, practical information and conclusions (§6–8).

52 Wanda is open-source, and is available at: <http://wandahot.sourceforge.net>. The
53 snapshot that was used in the present paper, including all back-ends, is available from:
54 <https://www.cs.ru.nl/~cynthiakop/experiments/fscd20/wanda2020.zip>.

55 **Theoretical contribution.** Although the focus is on Wanda, this paper also presents some
56 theoretical results that were previously only published in the author’s PhD thesis:

- 57 ■ a transformation from pattern HRSs [44] to Wanda’s internal format, AFSMs;
- 58 ■ two simple non-termination techniques (§3.1–3.2);
- 59 ■ a new variation of the higher-order recursive path ordering suited to AFSMs (§4.2).

60 In addition, the results of §2.3 and §4.1, and the “dynamic” part of §5, were previously
61 presented for a more restricted formalism and are here generalised to AFSMs. The remaining
62 results in this paper connect and discuss existing work, and explain how it is used in Wanda.

63 **2 Higher-order term rewriting using AFSMs**

64 There is no single, unified approach to higher-order term rewriting; rather, there are several
65 similar but not fully compatible systems. This is a problem, since users of various kinds of
66 higher-order TRSs may be interested in termination, and it would be frustrating to adapt
67 techniques and write different tools for each style. Therefore, Wanda uses a custom format,
68 AFSMs, which several popular kinds of rewriting systems can be translated into. AFSMs
69 (Algebraic Functional Systems with Meta-variables) are essentially simply-typed CRSs [33]
70 and also largely correspond to the formalism in [7]. AFSMs are fully presented in [23].

71 **2.1 Preliminaries: the AFSM formalism**

72 Wanda operates on typed expressions, defined by Definitions 1 and 2.

73 ► **Definition 1 (Simple types).** *We fix a set \mathcal{S} of sorts. All sorts are simple types, and if*
74 *σ, τ are simple types, then so is $\sigma \rightarrow \tau$. Here, \rightarrow is right-associative.*

75 Denoting ι, κ for a sort, all types have a unique form $\sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$.

76 This definition does not have type variables, which occur in polymorphic styles of rewriting.
77 Wanda *does* allow them as input, but since the implementation of most termination techniques
78 does not support polymorphism, we will here consider only the simple types above.

79 ► **Definition 2 (Terms and meta-terms).** *We fix disjoint sets \mathcal{F} of function symbols, \mathcal{V} of*
80 *variables and \mathcal{M} of meta-variables, each symbol equipped with a type. Each meta-variable*
81 *is additionally equipped with a natural number (its arity). We assume that both \mathcal{V} and \mathcal{M}*
82 *contain infinitely many symbols of all types. The set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of terms over \mathcal{F}, \mathcal{V} consists of*
83 *expressions s where $s : \sigma$ can be derived for some type σ by the following clauses:*

- 84 (V) $x : \sigma$ if $x : \sigma \in \mathcal{V}$ (@) $s t : \tau$ if $s : \sigma \rightarrow \tau$ and $t : \sigma$
- (F) $f : \sigma$ if $f : \sigma \in \mathcal{F}$ (Λ) $\lambda x. s : \sigma \rightarrow \tau$ if $x : \sigma \in \mathcal{V}$ and $s : \tau$

85 Meta-terms are expressions whose type can be derived by the clauses above along with:

(M) $Z[s_1, \dots, s_k] : \sigma_{k+1} \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota$

if $Z : (\sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \dots \rightarrow \sigma_m \rightarrow \iota, k) \in \mathcal{M}$ and $s_1 : \sigma_1, \dots, s_k : \sigma_k$

The λ binds variables as in the λ -calculus; unbound variables are called free, and $FV(s)$ is the set of free variables in s . Meta-variables cannot be bound; we write $FMV(s)$ for the set of meta-variables occurring in s . A meta-term s is called closed if $FV(s) = \emptyset$ (even if $FMV(s) \neq \emptyset$). Meta-terms are considered modulo α -conversion. Application ($@$) is left-associative; abstractions (Λ) extend as far to the right as possible. A meta-term s has type σ if $s : \sigma$; it has base type if $\sigma \in \mathcal{S}$. Let $\text{head}(s) = \text{head}(s_1)$ if $s = s_1 s_2$; otherwise $\text{head}(s) = s$.

A (meta-)term s has a sub-(meta-)term t , notation $s \triangleright t$, if either $s = t$ or $s \triangleright t$, where $s \triangleright t$ if (a) $s = \lambda x.s'$ and $s' \triangleright t$, (b) $s = s_1 s_2$ and $s_2 \triangleright t$ or (c) $s = s_1 s_2$ and $s_1 \triangleright t$.

Note that every term s has a form $t s_1 \dots s_n$ with $n \geq 0$ and $t = \text{head}(s)$ a variable, function symbol, or abstraction; in meta-terms t may also be a meta-variable application $Z[s_1, \dots, s_k]$. Terms are the objects that we will rewrite; meta-terms are used to define rewrite rules. Note that all our terms (and meta-terms) are, by definition, well-typed. An example of a meta-term is $\lambda x.\lambda y.\text{sin } Z[x]$. In the left-hand side of a rule, this meta-term stands for an arbitrary term of the form $\lambda x.\lambda y.\text{sin } t$ where t may contain the bound variable x , but not the bound variable y . This is more fully defined in Definitions 4 and 5.

For rewriting, we will additionally employ *patterns*:

► **Definition 3 (Patterns).** A meta-term is a pattern if it has one of the forms $Z[x_1, \dots, x_k]$ with all x_i distinct variables and $Z : (\sigma, k) \in \mathcal{M}$ for some σ ; $\lambda x.l$ with $x \in \mathcal{V}$ and l a pattern; or a $\ell_1 \dots \ell_n$ with $a \in \mathcal{F} \cup \mathcal{V}$ and all ℓ_i patterns ($n \geq 0$).

In rewrite rules, meta-variables are used for *matching* and variables are only used with *binders*. In terms, variables can occur both free and bound, and meta-variables cannot occur. Meta-variables originate in early forms of higher-order rewriting (e.g., [1, 33]), but have also been used in later formalisms (e.g., [7]). They strike a balance between matching modulo β and syntactic matching. By using meta-variables, we obtain the same expressive power as with Miller patterns [42], but without including a reversed β -reduction as part of matching.

In Wanda, function symbols are identified by their name, and variables and meta-variables by an integer index; using integers makes it very easy to allocate fresh variables when needed. The indexes are not shown to the user; instead a unique name is generated for printing.

► **Definition 4 (Substitution).** A substitution γ is a type-preserving mapping from a subset of $\mathcal{V} \cup \mathcal{M}$ (the domain of γ) to terms, typically denoted in a form $\gamma = [b_1 := s_1, \dots, b_n := s_n]$ (here, the domain is $\{b_1, \dots, b_n\}$). Substitutions may have infinite domain, but—denoting $\text{dom}(\gamma)$ for the domain of γ —we require that there are infinitely many variables x of all types such that (a) $x \notin \text{dom}(\gamma)$ and (b) for all $b \in \text{dom}(\gamma)$: $x \notin FV(\gamma(b))$.

A substitution is extended to a function from meta-terms to meta-terms as follows:

$$\begin{array}{lll}
 x\gamma & = & \gamma(x) & \text{if } x \in \mathcal{V} \cap \text{dom}(\gamma) \\
 x\gamma & = & x & \text{if } x \in \mathcal{V} \setminus \text{dom}(\gamma) \\
 \mathbf{f}\gamma & = & \mathbf{f} & \text{if } \mathbf{f} \in \mathcal{F} \\
 (s t)\gamma & = & (s\gamma) (t\gamma) & \\
 (\lambda x.s)\gamma & = & \lambda x.(s\gamma) & \text{if } x \notin \text{dom}(\gamma) \wedge x \notin \\
 & & & \bigcup_{y \in \text{dom}(\gamma)} FV(\gamma(y)) \\
 Z[s_1, \dots, s_k]\gamma & = & Z[s_1\gamma, \dots, s_k\gamma] & \text{if } Z \notin \text{dom}(\gamma) \\
 Z[s_1, \dots, s_k]\gamma & = & t[x_1 := s_1\gamma, \dots, x_k := s_k\gamma] & \text{if } \gamma(Z) = \lambda x_1 \dots x_k.t \\
 Z[s_1, \dots, s_k]\gamma & = & t[x_1 := s_1\gamma, \dots, x_n := s_n\gamma] (s_{n+1}\gamma) \dots (s_k\gamma) & \text{if } \gamma(Z) = \lambda x_1 \dots x_n.t \\
 & & & \wedge n < k
 \end{array}$$

15:4 WANDA – a Higher Order Termination Tool

122 Note that substituting an abstraction is fully defined due to α -conversion and the requi-
 123 rement that there are infinitely many variables not occurring in the domain or range of γ .
 124 Moreover, for fixed k , any meta-term $\gamma(Z)$ can be written in the form $\lambda x_1 \dots x_n. t$ with either
 125 $n < k$ and t not an abstraction, or $n = k$ (and t unrestricted). Thus, this is well-defined.

126 Essentially, applying a substitution with meta-variables in its domain combines a substi-
 127 tution with a β -development. For example, `deriv ($\lambda x.$ sin ($F[x]$))[$F := \lambda y.$ plus $y x$]` equals
 128 `deriv ($\lambda z.$ sin (plus $z x$))`, and `$X[0, \text{nil}][X := \lambda x.$ map ($\lambda y.$ x)]` equals `map ($\lambda y.$ 0) nil`. If
 129 $\text{dom}(\gamma)$ contains all meta-variables in $FMV(s)$, then $s\gamma$ is a term.

130 ► **Definition 5 (Rules and Rewriting).** A rule is a pair $\ell \Rightarrow r$ of closed meta-terms of the
 131 same type, where ℓ is a pattern of the form $\mathbf{f} \ell_1 \dots \ell_n$ with $\mathbf{f} \in \mathcal{F}$, and $FMV(r) \subseteq FMV(\ell)$.
 132 For a set of rules \mathcal{R} , reduction is the smallest monotonic relation $\Rightarrow_{\mathcal{R}}$ on terms that includes:

(Rule) $\ell\gamma \Rightarrow_{\mathcal{R}} r\gamma$ for $\ell \Rightarrow r \in \mathcal{R}$, and γ a substitution with $\text{dom}(\gamma) = FMV(\ell)$
 133 (Beta) $(\lambda x.s) t \Rightarrow_{\mathcal{R}} s[x := t]$

134 Note that we can reduce at any position of a term, even below a λ . We write $s \Rightarrow_{\beta} t$ if $s \Rightarrow_{\mathcal{R}} t$
 135 is derived using (Beta). A term s is terminating under \mathcal{R} if there is no infinite reduction
 136 $s = s_0 \Rightarrow_{\mathcal{R}} s_1 \Rightarrow_{\mathcal{R}} \dots$, is in normal form if there is no t with $s \Rightarrow_{\mathcal{R}} t$, and is β -normal if
 137 there is no t with $s \Rightarrow_{\beta} t$. The relation $\Rightarrow_{\mathcal{R}}$ is terminating if all terms are terminating.

138 Although the theory in [36] allows for \mathcal{R} to be infinite (mostly with an eye on polymor-
 139 phism), Wanda does not fully support this yet, so we will here limit interest to finite \mathcal{R} .

140 ► **Example 6.** Let $\mathcal{F} \supseteq \{0 : \text{nat}, \mathbf{s} : \text{nat} \rightarrow \text{nat}, \text{nil} : \text{list}, \text{cons} : \text{nat} \rightarrow \text{list} \rightarrow$
 141 $\text{list}, \text{map} : (\text{nat} \rightarrow \text{nat}) \rightarrow \text{list} \rightarrow \text{list}\}$ and consider the following rules \mathcal{R}_1 :

142
$$\begin{aligned} \text{map } (\lambda x.Z[x]) \text{ nil} &\Rightarrow \text{nil} \\ \text{map } (\lambda x.Z[x]) (\text{cons } H T) &\Rightarrow \text{cons } Z[H] (\text{map } (\lambda x.Z[x]) T) \end{aligned}$$

143 Then `map ($\lambda y.$ 0) (cons (\mathbf{s} 0) nil) $\Rightarrow_{\mathcal{R}_1}$ cons 0 (map ($\lambda y.$ 0) nil) $\Rightarrow_{\mathcal{R}_1}$ cons 0 nil`. Note that
 144 the bound variable y does not need to occur in the body of $\lambda y.0$ to be matched by $\lambda x.Z[x]$.
 145 However, note also that a term like `map \mathbf{s} (cons 0 nil)` cannot be reduced, because \mathbf{s} does
 146 not match $\lambda x.Z[x]$. We could alternatively consider the rules \mathcal{R}_2 :

147
$$\begin{aligned} \text{map } Z \text{ nil} &\Rightarrow \text{nil} \\ \text{map } Z (\text{cons } H T) &\Rightarrow \text{cons } (Z H) (\text{map } Z T) \end{aligned}$$

148 In the previous example, we had $Z : (\text{nat} \rightarrow \text{nat}, 1) \in \mathcal{M}$; here, we have $Z : (\text{nat} \rightarrow \text{nat}, 0) \in$
 149 \mathcal{M} (we will typically leave this implicit since the arity of meta-variables can be read off
 150 from the left-hand sides of the rules). Instead of meta-variable application $Z[x]$, we use
 151 explicit application $Z x$. Now we do have `map \mathbf{s} (cons 0 nil) $\Rightarrow_{\mathcal{R}_2}$ cons (\mathbf{s} 0) (map \mathbf{s} nil)`.
 152 However, now we will often need explicit β -reductions; e.g., `map ($\lambda y.$ 0) (cons (\mathbf{s} 0) nil) $\Rightarrow_{\mathcal{R}_2}$`
 153 `cons (($\lambda y.$ 0) (\mathbf{s} 0)) (map ($\lambda y.$ 0) nil) \Rightarrow_{β} cons 0 (map ($\lambda y.$ 0) nil)`.

154 Thus, AFSMs allow us to define essentially the same rules in multiple ways. This flexibility
 155 may seem redundant, but is necessary to enable the analysis of different styles of higher-order
 156 term rewriting, as we will see in §2.2. An AFSM is a pair $(\mathcal{T}(\mathcal{F}, \mathcal{V}), \Rightarrow_{\mathcal{R}})$ of a set of terms
 157 and a reduction relation on that set. To define an AFSM, it suffices to supply \mathcal{F} and \mathcal{R} ;
 158 types of (meta-)variables can be derived from context. This is what Wanda takes as input.

159 ► **Example 7.** The first `map` rules from Example 6 can be given to Wanda, in a file `map.afsm`,
 160 which provides first the signature and then the rules:

```

161 nil : list
162 cons : nat -> list -> list
163 map : (nat -> nat) -> list -> list
164
165 map (/ \x.Z[x]) nil => nil
166 map (/ \x.Z[x]) (cons H T) => cons Z[H] (map (/ \x.Z[x]) T)

```

167 Note: all identifiers (function symbols, variables and meta-variables) in `.afsm` files are
 168 expected to be alphanumeric. Characters such as `+`, `-` and `_` are not allowed in names. The
 169 only exceptions are the exclamation mark symbol (`!`) and the percentage symbol (`%`); the
 170 latter may only be used at the start of (meta-)variables.

171 2.2 Transformations

172 AFSMs are not meant to be interesting in their own right. Rather, they are defined to
 173 support termination proofs in multiple formalisms. Let us consider the two most relevant.

174 **Higher-order Rewriting Systems** (HRSs) [44] are one of the oldest styles of higher-order
 175 term rewriting. Here, rewriting is *modulo* \Rightarrow_β : for terms s, t in η -long β -normal form we
 176 have $s \Rightarrow_{\mathcal{R}} t$ if there exist a rule $\ell \Rightarrow r$ and a substitution γ such that $\ell\gamma \Rightarrow_\beta^* s$ and $r\gamma \Rightarrow_\beta^* t$.
 177 All terms are presented in η -long β -normal form, and rules are pairs of such terms (there are
 178 no meta-variables). The η -long form of a term s is obtained by repeatedly applying the step
 179 “ $s \Rightarrow_\eta \lambda x.(s\ x)$ ” on all subterms of s where this can be done without creating a β -redex.

180 In general, the reduction relation $\Rightarrow_{\mathcal{R}}$ in an HRS is not computable, but practical
 181 examples typically consider *pattern HRSs* (PRSs), where for all rules $\ell \Rightarrow r$ and for all
 182 subterms $x\ \ell_1 \cdots \ell_m$ of the left-hand side with x a variable: each ℓ_i is the η -long form of a
 183 distinct bound variable. Pattern HRSs are translated to AFSMs in a natural way, by replacing
 184 free variables in the rules by meta-variables, and their applications by meta-applications.

185 ► **Example 8.** Let us consider an example of a pattern HRS:

```

186   bind (return x) (\y.f y)  => f x
      bind x (\y.return y)   => x
  bind (bind x (\y.f y)) (\z.g z) => bind x (\u.bind (f u) (\v.g v))

```

187 It is translated to the following AFSM (meta-variables are indicated with capitals):

```

188   bind (return X) (\y.F[y]) => F[X]
      bind X (\y.return y)   => X
  bind (bind X (\y.F[y])) (\z.G[z]) => bind X (\u.bind (F[u]) (\v.G[v]))

```

189 This translated system has very similar behaviour to the original PRS, but there is a
 190 critical difference: the PRS is a relation on η -long β -normal terms, while the AFSM is
 191 generally considered as a relation on all terms. It turns out that the restriction to η -long
 192 terms does not affect termination, but the β -normalisation does ([36, Theorem 3.5]):

193 ► **Lemma 9.** *The original PRS $(\mathcal{F}, \mathcal{R})$ is terminating if and only if the translated AFSM*
 194 *$(\mathcal{F}, \mathcal{R}')$ is terminating using a reduction strategy where \Rightarrow_β is preferred to other steps.*

195 That is, we need to β -normalise terms after every reduction step. Wanda can test the
 196 property of termination with a \Rightarrow_β -first strategy by being invoked with a runtime argument
 197 `--betafirst` (e.g., `./wanda.exe --betafirst system.afsm`). As a side note, however, the
 198 examples where this requirement makes a difference are rare and typically artificial.

199 ► **Example 10.** Let $\mathcal{F} = \{\mathbf{a} : \mathbf{o}, \mathbf{f} : \mathbf{o} \rightarrow \mathbf{o}, \mathbf{g} : ((\mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o}\}$ and \mathcal{R} given by:

$$200 \quad \mathbf{f} \ \mathbf{a} \ \Rightarrow \ \mathbf{g} \ (\lambda x. \lambda y. x \ (\mathbf{f} \ y)) \quad \mathbf{g} \ (\lambda x. \lambda y. h \ (\lambda z. x \ z) \ y) \ \Rightarrow \ h \ (\lambda z. \mathbf{a}) \ (\mathbf{a})$$

201 This PRS is translated to an AFSM with the following rules \mathcal{R}' :

$$202 \quad \mathbf{f} \ \mathbf{a} \ \Rightarrow \ \mathbf{g} \ (\lambda x. \lambda y. x \ (\mathbf{f} \ y)) \quad \mathbf{g} \ (\lambda x. \lambda y. H[x, y]) \ \Rightarrow \ H[\lambda z. \mathbf{a}, \ \mathbf{a}]$$

203 While the original PRS is terminating, the same does not hold for the translated AFSM: we
 204 have $\mathbf{f} \ \mathbf{a} \Rightarrow_{\mathcal{R}'} \mathbf{g} \ (\lambda x. \lambda y. x \ (\mathbf{f} \ y)) \Rightarrow_{\mathcal{R}'} (\lambda z. \mathbf{a}) \ (\mathbf{f} \ \mathbf{a})$, where the last term has $\mathbf{f} \ \mathbf{a}$ as a subterm.
 205 In an AFSM, it is not mandatory to reduce the β -redex. *Wanda* concludes non-termination
 206 normally, but cannot find a proof or disproof if the `--betafirst` argument is provided.

207 *Remark:* *Wanda* has not been optimised for HRSs, and does not take advantage of the
 208 `--betafirst` argument other than avoiding false claims of non-termination. This is primarily
 209 due to a lack of motivating examples: the annual Termination Competition does not consider
 210 HRSs. However, since the *International Confluence Competition* [12] *does* consider HRSs,
 211 and comes with its own benchmark set, this situation is likely to change in the future.

212 **Algebraic Functional Systems** (AFSs) [30] are higher-order term rewriting systems with
 213 \Rightarrow_{β} as a separate step (i.e., $\Rightarrow_{\beta} \subseteq \Rightarrow_{\mathcal{R}}$; unlike HRSs, β -steps are not implicitly done as part
 214 of other steps); this is the format used in the higher-order category of the International
 215 Termination Competition [13]. Rules in an AFS are pairs of *terms*, not *meta-terms*, and
 216 there is no pattern Another difference with AFSMs is that AFSMs use applicative (curried)
 217 notation while AFSs use a mixture of functional and applicative term formation; however, this
 218 difference is not significant, since—following [35, 36]—currying does not affect termination.

219 Using variables rather than meta-variables for matching is not important either: just
 220 replace all free variables by meta-variables. This gives rules like the “alternative” rules \mathcal{R}_2 in
 221 Example 6. However, the lack of a pattern restriction is very significant.

222 ► **Example 11.** Let us consider an example of an AFS that cannot be naturally translated
 223 without violating pattern restrictions. We let $\mathcal{F} = \{\mathbf{new} : (\mathbf{N} \rightarrow \mathbf{A}) \rightarrow \mathbf{A}\}$ and \mathcal{R} consist of:

$$224 \quad \mathbf{new} \ (\lambda x. y) \ \Rightarrow \ y \quad \mathbf{new} \ (\lambda x. \mathbf{new} \ (\lambda y. f \ x \ y)) \ \Rightarrow \ \mathbf{new} \ (\lambda x. \mathbf{new} \ (\lambda y. f \ y \ x))$$

225 Now, the left-hand sides *look* like patterns. Indeed, they satisfy the requirements for an
 226 HRS-pattern: the free variable f in the second rule is only applied to distinct bound variables.
 227 So if this was an HRS, we could translate it to the following AFSM:

$$228 \quad \mathbf{new} \ (\lambda x. Y) \ \Rightarrow \ Y \quad \mathbf{new} \ (\lambda x. \mathbf{new} \ (\lambda y. F[x, y])) \ \Rightarrow \ \mathbf{new} \ (\lambda x. \mathbf{new} \ (\lambda y. F[y, x]))$$

229 However, since the original system was an AFS, this is *not* equivalent. Unlike in HRSs,
 230 matching in AFSs is not modulo beta: like in AFSMs, s rewrites to t by rule $\ell \Rightarrow r$ if there
 231 exists a substitution γ such that $s = \ell\gamma$ and $t = r\gamma$. So, in the AFS, the subterm $f \ x \ y$ can
 232 *only* be instantiated by terms of the form $s \ x \ y$. An accurate translation of the second rule
 233 to AFSMs would simply replace $f \ x \ y$ by $F[] \ x \ y$, resulting in a non-pattern.

234 This is important because the AFSM above is non-terminating: $\mathbf{new} \ (\lambda x. \mathbf{new} \ (\lambda y. z))$
 235 reduces to itself in one step because the meta-variable F can be instantiated by a substitution
 236 $\lambda x. \lambda y. z$. On the other hand, the original AFS is terminating, as we will see below.

237 A final difference is that, following [30], AFSs use polymorphic types. *Wanda* limits
 238 interest to simply-typed AFSs, which is what the Termination Competition uses. Polymorphic
 239 AFSs can be translated to polymorphic AFSMs, but this is not yet well-supported in *Wanda*.

240 Wanda accepts AFSs as input directly (using the xml format of the Termination Com-
 241 petition or a custom human-readable format). *Most* AFSs can be naturally translated into
 242 AFSMs just by replacing free variables by meta-variables; typically counterexamples look like
 243 they were meant as HRSs, but translated poorly into AFSs. For the examples that cannot be
 244 naturally translated, Wanda first applies the transformations in [35] to create patterns. This
 245 involves introducing fresh symbols app_i to replace some of the applications $s t$ by terms of
 246 the form $\text{app}_i s t$. New rules may also be introduced, as for instance $\mathbf{f} (X Y) \mathbf{a} \Rightarrow \mathbf{f} (X \mathbf{b}) Y$
 247 is replaced by not only $\mathbf{f} (\text{app}_i X Y) \mathbf{a} \Rightarrow \mathbf{f} (\text{app}_i X \mathbf{b}) Y$, but in addition potentially many
 248 rules of the form $\mathbf{f} (\mathbf{g} X_1 \cdots X_n Y) \mathbf{a} \Rightarrow \mathbf{f} (\mathbf{g} X_1 \cdots X_n \mathbf{a}) Y$. This is exacerbated when the
 249 AFS is presented in curried (applicative) form rather than functional notation.

250 ► **Example 12.** The AFS of Example 11 is translated to an AFSM with the following rules:

$$\begin{aligned}
 & \text{new } (\lambda x.Y) \Rightarrow Y \\
 251 \quad & \text{new } (\lambda x.\text{new } (\lambda y.\text{app } F x y)) \Rightarrow \text{new } (\lambda x.\text{new } (\lambda y.\text{app } F u z)) \\
 & \text{app } F X \Rightarrow F X
 \end{aligned}$$

252 This AFSM can be proved terminating by Wanda’s recursive path ordering (§4.2) in combin-
 253 ation with dependency pairs (§5).

254 It is worth noting that the transformations needed to translate an AFS to an AFSM
 255 with equivalent behaviour can sometimes cause the system to become much more difficult
 256 to analyse, both due to the inclusion of explicit “application” symbols in the rules and the
 257 addition of potentially many new rules. For this reason, Wanda uses the following approach:

- 258 ■ create both an accurate translation and an *overestimation* of the AFS (so that termination
 259 of the overestimation implies termination of the original system, but not the reverse);
 260 this results in translations like those given in Example 11;
- 261 ■ try to prove non-termination using the accurate translation;
- 262 ■ try to prove termination using the overestimation;
- 263 ■ if this fails, try to prove termination using the accurate translation.

264 2.3 Uncurrying

265 Following [36, §2.3.1] and [35, §7], *uncurrying* does not affect termination provided the rules
 266 are (essentially) unchanged. That is, we can denote both rules and terms in a functional
 267 notation, but only if the number of arguments is respected in each rule. To be exact:

268 ► **Lemma 13.** *Let $(\mathcal{F}, \mathcal{R})$ be an AFSM, and let $\text{minar}(\mathbf{f})$ denote the largest number k such
 269 that (1) the type of \mathbf{f} allows \mathbf{f} to be applied to at least k arguments, and (2) every occurrence
 270 of \mathbf{f} in \mathcal{R} is applied to at least k arguments. Then $\Rightarrow_{\mathcal{R}}$ is non-terminating if and only if there
 271 is an infinite reduction $s_1 \Rightarrow_{\mathcal{R}} s_2 \Rightarrow_{\mathcal{R}} \dots$ where, in every term s_i , each symbol \mathbf{f} always
 272 occurs with at least $\text{minar}(\mathbf{f})$ arguments.*

273 For example, in Example 6, $\text{minar}(\mathbf{s}) = 1$ and $\text{minar}(\mathbf{cons}) = \text{minar}(\mathbf{map}) = 2$; thus, we
 274 do not need to consider terms such as $\mathbf{map} \mathbf{s} (\mathbf{cons} 0 \mathbf{nil})$ or $\mathbf{map} (\lambda x.\mathbf{s} x)$ for termination.
 275 Wanda indicates this by showing terms in functional notation; e.g., $\mathbf{map}(\lambda x.\mathbf{s}(x), \mathbf{cons}(0, \mathbf{nil}))$.

276 ► **Example 14.** Consider the toy system with $\mathcal{F} = \{\mathbf{a}, \mathbf{b} : \mathbf{o}, \mathbf{f} : \mathbf{o} \rightarrow \mathbf{o} \rightarrow \mathbf{o}, \mathbf{g} : \mathbf{o} \rightarrow (\mathbf{o} \rightarrow$
 277 $\mathbf{o}) \rightarrow \mathbf{o}\}$ and $\mathcal{R} = \{\mathbf{f} \mathbf{a} X \Rightarrow \mathbf{g} X (\mathbf{f} \mathbf{a}), \mathbf{g} \mathbf{a} F \Rightarrow F \mathbf{b}\}$. Then $\text{minar}(\mathbf{a}) = \text{minar}(\mathbf{b}) = 0$,
 278 $\text{minar}(\mathbf{g}) = 2$ and $\text{minar}(\mathbf{f}) = 1$ (since \mathbf{f} occurs both with 1 or 2 arguments, we must choose
 279 the smaller value). Wanda prints these rules as $\mathbf{f}(\mathbf{a}) X \Rightarrow \mathbf{g}(X, \mathbf{f}(\mathbf{a}))$ and $\mathbf{g}(\mathbf{a}, F) \Rightarrow F \mathbf{b}$.

280 We do *not* η -expand as part of uncurrying. To illustrate why not, note that the above
 281 system is terminating, but its η -long variant, which has a rule $\mathbf{f} \mathbf{a} X \Rightarrow \mathbf{g} (\lambda z.\mathbf{f} \mathbf{a} z)$, is not.

282 3 Non-termination

283 As Wanda’s focus is on proving termination, the available non-termination techniques are
 284 currently quite minimal. There are three methods. The first two are very quick, and are
 285 applied at the start of the analysis, before termination is considered. The last one is employed
 286 when dependency pairs are initiated, as it is combined with the simplification given in §5.2.

287 3.1 Detecting obvious loops

288 An AFSM is clearly non-terminating if there is a reduction $s \Rightarrow_{\mathcal{R}}^* t$ such that $t \succeq s\gamma$ for some
 289 γ . To discover such loops, Wanda takes the left-hand side of a rule, replaces meta-variable
 290 applications $Z[x_1, \dots, x_k]$ by variable applications $y x_1 \cdots x_k$, and performs a breadth-first
 291 search on reducts to see whether any instances of the original term appear, not going beyond
 292 the first 1000. If the `betafirst` runtime argument is given, then reducts are β -normalised
 293 before this test is done. This simple method will not find any sophisticated counterexamples
 294 for termination, but is quick and easy, and often catches mistakes in a recursive call.

295 In the future, it would be natural to extend this module to use semi-unification [32]
 296 instead of matching, as done for first-order rewriting in [27]. However, this would require the
 297 design of a higher-order semi-unification algorithm. Similarly, Wanda could be strengthened
 298 by creating higher-order variants of existing first-order non-termination techniques (e.g.,
 299 [18, 46, 47]), but this would require substantial new work to develop the theory.

300 3.2 The $\omega\omega$ counterexample

301 Wanda also has one truly higher-order non-termination technique, which does not build
 302 on first-order methods. This technique recognises a particular kind of rule that leads to
 303 non-termination in a non-obvious way. The idea is to build a variation of the λ -term $\omega\omega$ in
 304 the untyped λ -calculus, where $\omega = \lambda x.xx$. Note that $\omega\omega$ reduces to itself in one \Rightarrow_{β} -step.

305 Let a *context* be a meta-term $\underline{C}[\square_1, \dots, \square_n]$ containing n typed holes \square_i , and denote
 306 $\underline{C}[s_1, \dots, s_n]$ for the same meta-term with each \square_i replaced by s_i . Wanda identifies rules
 307 $\ell \Rightarrow r$ where ℓ has the form $\underline{C}[\underline{D}[Z], X]$ such that:

- 308 ■ $Z : \sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \tau \in \mathcal{M}$, where τ is the type of ℓ ;
- 309 ■ there is some i with $X : (\sigma_i, 0) \in \mathcal{M}$ and also $\underline{D}[Z]$ has type σ_i ;
- 310 ■ r can be written as $\underline{E}[Z s_1 \cdots s_{i-1} X s_{i+1} \cdots s_n]$
- 311 ■ X and Z do not appear at other positions in \underline{C} or \underline{D} .

312 If this is satisfied, Wanda concludes non-termination with the following justification. Let γ
 313 be the substitution that maps each $Y : \pi_1 \rightarrow \dots \rightarrow \pi_m \rightarrow \iota$ in the rule, aside from Z and X , to
 314 a term $\lambda x_1 \dots x_m.y x_1 \cdots x_m$ (with y a variable), and let $\omega := \underline{D}\gamma[\lambda x_1 \dots x_n.\underline{C}\gamma[x_i, x_i]]$. Let
 315 $\delta := \gamma \cup [X := \omega, Z := \lambda x_1 \dots x_n.\underline{C}\gamma[x_i, x_i]]$. Then $\underline{C}\gamma[\omega, \omega] = \ell\delta \Rightarrow_{\mathcal{R}} \underline{E}[(\lambda x_1 \dots x_n.\underline{C}\gamma[x_i,$
 316 $x_i]) s_1 \cdots \omega \cdots s_n]\delta \Rightarrow_{\beta}^* \underline{E}[\underline{C}\gamma[\omega, \omega]]\delta \succeq \underline{C}\gamma[\omega, \omega]$, a loop.

317 The method above is specialised for AFSMs that originate from AFSs (as used in the
 318 Termination Competition): it is designed for meta-variables that do not take any arguments.
 319 If meta-variables do take arguments, and for instance $\lambda x_1 \dots x_n.Z[x_1, \dots, x_n]$ is used instead
 320 of Z , we *probably* have a similar counter-example—depending on how Z and X are used in
 321 \underline{E} (it is possible that $\underline{E}[\delta]$ does not contain any copies of \square_1). Wanda tries to recognise such
 322 variations of the meta-variables, and tests whether the counterexample still applies.

3.3 Using a first-order tool

Finally, it is clear that an AFSM $(\mathcal{F}, \mathcal{R})$ is non-terminating if there is a subset $\mathcal{R}' \subseteq \mathcal{R}$ such that $\Rightarrow_{\mathcal{R}'}$ is non-terminating. An interesting subset is the set of rules that can be viewed as first-order (i.e., rules that do not use λ , that only use function symbols with a type declaration $\iota_1 \rightarrow \dots \rightarrow \iota_m \rightarrow \iota_0$ with all $\iota_i \in \mathcal{S}$, and where function symbols only occur fully applied). This subset is easier to analyse, as known methods for first-order rewriting apply.

Thus, Wanda extracts this first-order part, to pass it to a dedicated first-order (non-) termination tool. The main problem with this approach is that existing tools do not consider types. This can make a difference, as shown by an example due to Toyama [51]:

► **Example 15.** Let $\mathcal{F} = \{0 : \mathbf{a}, 1 : \mathbf{a}, f : \mathbf{a} \rightarrow \mathbf{a} \rightarrow \mathbf{a} \rightarrow \mathbf{a}, g : \mathbf{b} \rightarrow \mathbf{b} \rightarrow \mathbf{b}\}$ and $\mathcal{R} = \{f(X, X, X) \Rightarrow f(0, 1, X), g(X, Y) \Rightarrow X, g(X, Y) \Rightarrow Y\}$. This system is terminating, because there is no term of type \mathbf{a} that reduces to both 0 and 1. However, there is an *untypable* term that loops by these rules: $f(0, 1, g(0, 1)) \Rightarrow_{\mathcal{R}} f(g(0, 1), g(0, 1), g(0, 1)) \Rightarrow_{\mathcal{R}} f(0, g(0, 1), g(0, 1)) \Rightarrow_{\mathcal{R}} f(0, 1, g(0, 1))$. Thus, a first-order termination tool (which does not consider types) would conclude non-termination.

Now, if the first-order subset is *orthogonal*, then it is terminating if and only if it is terminating without regarding types as observed in [21] (using a combination of results in [20] and [28]). Thus, in this case Wanda can use an arbitrary first-order tool without inhibitions. The same is true if the set of first-order rules uses only one sort. If neither of those cases holds, Wanda investigates the output of the first-order tool to see whether a non-terminating term is given, and if so, tests whether it is well-sorted.

Comment: unfortunately, the standard output format for the Termination Competition does not require tools to output a non-terminating term if NO is answered. Thus, any common first-order tool can be used if \mathcal{R}' is orthogonal or has only one sort, but otherwise a specialised tool with the right output format is needed. For this, Wanda uses a custom adaptation of AProVE [25]. As AProVE is currently not open-source, this is not included in Wanda's release.

4 Orderings

At the heart of Wanda's termination techniques are *reduction pairs*. These are orderings on terms—generated by an ordering on meta-terms—which can be used both as part of the dependency pair framework (§5) and on their own to simplify a termination proof.

► **Definition 16.** A reduction pair is a pair (\succsim, \succ) of a quasi-ordering and a well-founded ordering on meta-terms of the same type, such that:

- \succsim and \succ are compatible: $\succ \cdot \succsim$ is included in \succ ;
- \succsim and \succ are meta-stable: if $s \succsim t$ and γ is a substitution on domain $FMV(s) \cup FMV(t)$, then $s\gamma \succsim t\gamma$ (and similar for \succ);
- \succsim is monotonic: if $s \succsim t$, then $s u \succsim t u$ and $u s \succsim u t$ and $\lambda x.s \succsim \lambda x.t$
- \succsim contains beta: $(\lambda x.s) t \succsim s[x := t]$ if s and t are terms.

A reduction pair is strongly monotonic if moreover \succ is monotonic.

Strongly monotonic reduction pairs can be used in *rule removal*: if $\ell \succsim r$ for some rules, and $\ell \succ r$ for the remainder, then the rules in the remainder cannot occur infinitely often in a reduction sequence, and thus can be “removed” (they no longer need to be considered for the termination argument). Reduction pairs are also used—without the strong monotonicity requirement—in the dependency pair framework. It *would* be possible to also include rule

366 removal with strongly monotonic reduction pairs in the framework rather than using it as a
 367 separate step; however, using it as a separate step often gives simpler termination proofs,
 368 and makes it possible to assess the strength of these reduction pairs in isolation.

369 Wanda has two ways to generate reduction pairs: *weakly monotonic interpretations* and
 370 *recursive path orderings*. Both ideas extend first-order methods, and use *functional notation*.
 371 This is an extension of uncurrying, where the remaining applications are replaced by function
 372 application, as follows: in every rule, every subterm of the left- or right-hand side of the form
 373 $s\ t$ is replaced by $@^{\sigma,\tau}(s, t)$, where $s : \sigma \rightarrow \tau$. The set of all symbols $@^{\sigma,\tau}$ that are used in the
 374 rules is added to \mathcal{F} , and the corresponding rules $@^{\sigma,\tau}(\lambda x.Z[x], Y) \Rightarrow Z[Y]$ are added to \mathcal{R} .

375 ► **Example 17.** The AFSM of Example 14 is functionalised by replacing $\mathbf{f}(\mathbf{a})\ X$ in the
 376 uncurried rules by $@^{\circ,\circ}(\mathbf{f}(\mathbf{a}), X)$ and $F\ \mathbf{b}$ by $@^{\circ,\circ}(F, \mathbf{b})$. Thus, we obtain the rules:

$$377 \quad \begin{array}{ll} @^{\circ,\circ}(\mathbf{f}(\mathbf{a}), X) & \Rightarrow \mathbf{g}(X, \mathbf{f}(\mathbf{a})) & @^{\circ,\circ}(\lambda x.Z[x], Y) & \Rightarrow Z[Y] \\ \mathbf{g}(\mathbf{a}, F) & \Rightarrow @^{\circ,\circ}(F, \mathbf{b}) \end{array}$$

378 4.1 Weakly monotonic algebras

379 The idea of van de Pol’s *weakly monotonic algebras* [48] is to assign valuations which map
 380 all function symbols \mathbf{f} of type σ to a *weakly monotonic functional* $\mathcal{J}_{\mathbf{f}}$: an element of $\llbracket \sigma \rrbracket$,
 381 where $\llbracket \iota \rrbracket$ is the set of natural numbers for a sort ι and $\llbracket \sigma \rightarrow \tau \rrbracket$ is the set of those functions
 382 from $\llbracket \sigma \rrbracket$ to $\llbracket \tau \rrbracket$ that are weakly monotonic (i.e., if $a, b \in \llbracket \sigma \rrbracket$ and $a \geq b$, then $f(a) \geq f(b)$ for
 383 $f \in \llbracket \sigma \rightarrow \tau \rrbracket$, where \geq is a point-wise comparison). This induces a value on closed terms,
 384 which can be extended to a reduction pair, as explained below.

385 Given a meta-term s in functional notation and a function α which maps each variable $x : \sigma$
 386 occurring freely in s to an element of $\llbracket \sigma \rrbracket$ and each meta-variable $Z : (\sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \tau, k)$
 387 to an element of $\llbracket \sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \tau \rrbracket$, we let $[s]_{\alpha}^{\mathcal{J}}$ be recursively defined as follows:

$$388 \quad \begin{array}{ll} [x]_{\alpha}^{\mathcal{J}} & = \alpha(x) & [\mathbf{f}(s_1, \dots, s_k)]_{\alpha}^{\mathcal{J}} & = \mathcal{J}_{\mathbf{f}}([s_1]_{\alpha}^{\mathcal{J}}, \dots, [s_k]_{\alpha}^{\mathcal{J}}) \\ [\lambda x.s]_{\alpha}^{\mathcal{J}} & = u \mapsto [s]_{\alpha \cup [x:=u]}^{\mathcal{J}} & [Z[s_1, \dots, s_k]]_{\alpha}^{\mathcal{J}} & = \alpha(Z)([s_1]_{\alpha}^{\mathcal{J}}, \dots, [s_k]_{\alpha}^{\mathcal{J}}) \end{array}$$

389 (This follows the definition of $[\cdot]_{\alpha}^{\mathcal{J}}$ for functionalised AFSs in [22], but extends it with a case
 390 for meta-variable applications.) For closed meta-terms ℓ, r , let $\ell \succ r$ if $[\ell]_{\alpha}^{\mathcal{J}} > [r]_{\alpha}^{\mathcal{J}}$ for all α ,
 391 and $\ell \succsim r$ if $[\ell]_{\alpha}^{\mathcal{J}} \geq [r]_{\alpha}^{\mathcal{J}}$ for all α . Then (\succ, \succsim) is a reduction pair if the valuations $\mathcal{J}_{@(\sigma,\tau)}$
 392 are chosen to have $\mathcal{J}_{@(\sigma,\tau)}(F, X) \geq F(X)$. It is a strongly monotonic pair if each function
 393 $\mathcal{J}_{\mathbf{f}}$ (including each $\mathcal{J}_{@(\sigma,\tau)}$) is monotonic over $>$ in the first *minar*(\mathbf{f}) arguments.

394 In [22], a strategy is discussed to find interpretations based on *higher-order polynomials*
 395 for AFSs, and an automation using encodings of the ordering requirements into SAT. Wanda
 396 implements this methodology, only slightly adapted to take meta-variables into account.

397 ► **Example 18.** We consider \mathcal{R}_2 in Example 6. Let $\mathcal{J}_{\text{nil}} = 0$ and $\mathcal{J}_{\text{cons}} = (n, m) \mapsto n + m + 1$
 398 and $\mathcal{J}_{\text{map}} = (f, n) \mapsto nf(n) + 2n + f(0)$ and $\mathcal{J}_{@^{\text{nat},\text{nat}}} = (f, n) \mapsto f(n) + n$. Then, writing
 399 $F := \alpha(Z)$, $n := \alpha(H)$, $m := \alpha(T)$, we have:

$$400 \quad \begin{array}{l} \blacksquare [\text{map}(Z, \text{nil})]_{\alpha}^{\mathcal{J}} = F(0) \geq 0 = [\text{nil}]_{\alpha}^{\mathcal{J}} \\ 401 \blacksquare [\text{map}(Z, \text{cons}(H, T))]_{\alpha}^{\mathcal{J}} = (n + m + 1) \cdot F(n + m + 1) + 2 \cdot (n + m + 1) + F(0) > (F(n) + \\ 402 n) + (m \cdot F(m) + 2 \cdot m + F(0)) + 1 = [\text{cons}(@^{\text{nat},\text{nat}}(Z, H), \text{map}(Z, T))]_{\alpha}^{\mathcal{J}} \\ 403 \blacksquare [@^{\text{nat},\text{nat}}(\lambda x.Z[x], H)]_{\alpha}^{\mathcal{J}} = F(H) + H \geq F(H) = [F[H]]_{\alpha}^{\mathcal{J}}. \end{array}$$

404 As all $\mathcal{J}_{\mathbf{f}}$ are strictly monotonic in all arguments, we may remove the second rule.

4.2 StarHorpo

The recursive path ordering [15] is a syntactic method to extend an ordering on function symbols to an ordering on first-order terms. There are various extensions (e.g. [19, 31]) including several higher-order variations (e.g. [8, 30]). However, these are mostly designed for rewriting with plain matching, and adapting them to work well with meta-variables is non-trivial. Instead, **Wanda** uses a specialised definition, built using the same ideas as [30] but using *iterative* path orderings [34, 37] as a starting point. This is discussed in detail in [36, Ch. 5]; here, we note only the end result: a reduction pair that can be used on functionalised AFSMs and (unlike other higher-order recursive path orderings) is natively transitive.

Following [34, 37], **StarHorpo** employs a star mark \star to indicate an *intent to decrease*; practically, $\mathbf{f}_\sigma^\star(s_1, \dots, s_k)$ should be seen as an upper bound for all functional meta-terms of type σ which are *strictly smaller* than $\mathbf{f}(s_1, \dots, s_k)$. Let s^\star denote $\lambda x_1 \dots x_n. \mathbf{f}_\sigma^\star(s_1, \dots, s_k)$ if $s = \lambda x_1 \dots x_n. \mathbf{f}(s_1, \dots, s_k)$. If s has any other form, then s^\star is undefined.

StarHorpo assumes given a *precedence* \blacktriangleright : a quasi-ordering on all symbols, whose strict part \blacktriangleright is well-founded; we let \approx denote the equivalence relation $\blacktriangleright \cap \blacktriangleleft$. We assume that there is a special symbol \perp_σ for each type σ , which is minimal for \blacktriangleright (i.e., $\mathbf{f} \blacktriangleright \perp_\sigma$ for all \mathbf{f}); \perp_σ^\star is undefined. All symbols are assigned a *status* in $\{Lex, Mul\}$, such that $status(\mathbf{f}) = status(\mathbf{g})$ whenever $\mathbf{f} \approx \mathbf{g}$. Let $\succ_\star^\mathbf{f}$ denote either the lexicographic or multiset extension of \succ_\star , depending on the status of \mathbf{f} . Now the reduction pair $(\succeq_\star, \succ_\star)$ is given by the rules in Figure 1.

(\succ)	s	\succ_\star	t	if	$s^\star \succeq_\star t$	
(Var)	x	\succeq_\star	x	if	$x \in \mathcal{V}$	
(Abs)	$\lambda x. s$	\succeq_\star	$\lambda x. t$	if	$s \succeq_\star t$	
(Meta)	$Z[s_1, \dots, s_k]$	\succeq_\star	$Z[t_1, \dots, t_k]$	if	each $s_i \succeq_\star t_i$	
(Fun)	$\mathbf{f}(s_1, \dots, s_n)$	\succeq_\star	$\mathbf{g}(t_1, \dots, t_k)$	if	$\mathbf{f} \approx \mathbf{g}$ and $[s_1, \dots, s_n] \succeq_\star^\mathbf{f} [t_1, \dots, t_k]$	
(Put)	$\mathbf{f}(s_1, \dots, s_n)$	\succeq_\star	t	if	$\mathbf{f}_\sigma^\star(s_1, \dots, s_n) \succeq_\star t$ (for $\mathbf{f}(\vec{s}) : \sigma$)	
(Select)	$\mathbf{f}_\sigma^\star(s_1, \dots, s_n)$	\succeq_\star	t	if	$s_i(\mathbf{f}_{\tau_1}^\star(\vec{s}), \dots, \mathbf{f}_{\tau_j}^\star(\vec{s})) \succeq_\star t$ (**)	(**)
					where $s_i : \tau_1 \rightarrow \dots \rightarrow \tau_j \rightarrow \sigma$	
(FAbs)	$\mathbf{f}_{\sigma \rightarrow \tau}^\star(s_1, \dots, s_n)$	\succeq_\star	$\lambda x. t$	if	$\mathbf{f}_\tau^\star(s_1, \dots, s_n, x) \succeq_\star t$	
(Copy)	$\mathbf{f}_\sigma^\star(s_1, \dots, s_n)$	\succeq_\star	$\mathbf{g}(t_1, \dots, t_k)$	if	$\mathbf{f} \blacktriangleright \mathbf{g}$ and $\mathbf{f}_{\tau_i}^\star(\vec{s}) \succeq_\star t_i$ for $1 \leq i \leq k$	
(Stat)	$\mathbf{f}_\sigma^\star(s_1, \dots, s_n)$	\succeq_\star	$\mathbf{g}(t_1, \dots, t_k)$	if	$\mathbf{f} \approx \mathbf{g}$ and $\mathbf{f}_{\tau_i}^\star(\vec{s}) \succeq_\star t_i$ for $1 \leq i \leq k$ and $[s_1, \dots, s_n] \succ_\star^\mathbf{f} [t_1, \dots, t_k]$	
(Bot)	s	\succeq_\star	\perp_σ	if	$s : \sigma$	

The notation $s\langle t_1, \dots, t_n \rangle$ applies s to t_1, \dots, t_n in the following sense: $s\langle \rangle = s$ and $(\lambda x. s)\langle t, \vec{u} \rangle = s[x := t]\langle \vec{u} \rangle$ and $\mathbf{f}(\vec{s})\langle t, \vec{u} \rangle = \mathbf{f}_\tau^\star(\vec{s}, t)\langle \vec{u} \rangle$ and also $\mathbf{f}_{\sigma \rightarrow \tau}^\star(\vec{s})\langle t, \vec{u} \rangle = \mathbf{f}_\tau^\star(\vec{s}, t)\langle \vec{u} \rangle$.

■ **Figure 1** Rules of StarHorpo

Note that \succeq_\star and \succ_\star only compare terms of the same type, and that marked symbols \mathbf{f}^\star may occur with different types (indicated as subscripts) within a term. Symbols \mathbf{f}^\star may also have varying numbers of arguments, but must always have at least $minar(\mathbf{f})$.

► **Example 19.** Given a function symbol $@ : (\sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \tau$ (with σ and τ arbitrary types), we can prove $@(\lambda x. Z[x], Y) \succ Z[Y]$ as follows:

- by (\succ), because $@_\tau^\star(\lambda x. Z[x], Y) \succeq_\star Z[Y]$
- by (Select), because $Z[@_\sigma^\star(\lambda x. Z[x], Y)] \succeq_\star Z[Y]$
- by (Meta), because $@_\sigma^\star(\lambda x. Z[x], Y) \succeq_\star Y$
- by (Select) because $Y \succeq_\star Y$ by (Meta).

433 Wanda uses `StarHorpo` in combination with *argument functions*: each function symbol
 434 \mathbf{f} with $\mathit{minar}(\mathbf{f}) = k$ is mapped to a functionalised term $\lambda x_1 \dots x_k. s$, and in a given
 435 functionalised meta-term, all occurrences of $\mathbf{f}(t_1, \dots, t_k)$ are replaced by $s[x_1 := t_1, \dots, x_k :=$
 436 $t_k]$. If the reduction pair is required to be strongly monotonic (as is the case for rule removal),
 437 then $FV(s)$ must be $\{x_1, \dots, x_k\}$. Argument functions are a generalisation of *argument*
 438 *filterings* [40], and were introduced in [38]. In Wanda, they are not restricted to being used
 439 with dependency pairs (unlike [40, 38]), and s is limited to one of three forms: (1) x_i , (2)
 440 $\mathbf{f}'(x_{i_1}, \dots, x_{i_n})$ (with $n \leq k$, all x_{i_j} distinct), or (3) \perp_σ . This effectively extends argument
 441 filterings with argument permutations and a mapping to one of the minimal constants \perp_σ .

442 Wanda combines the search for a suitable precedence and status function with the search
 443 for an argument function, using a SAT encoding following [36, Chapter 8.6].

444 ► **Example 20.** Consider the (first-order) AFSM with just one sort `o` and the following rules:

$$445 \quad \mathbf{f} X (\mathbf{s} Y) \Rightarrow \mathbf{g} Y (\mathbf{s} (\mathbf{s} X)) \quad \mathbf{f} X Y \Rightarrow \mathbf{g} \mathbf{a} \mathbf{b} \quad \mathbf{g} X (\mathbf{s} Y) \Rightarrow \mathbf{f} Y X$$

446 Then $\mathit{minar}(\mathbf{f}) = \mathit{minar}(\mathbf{g}) = 2$. We use the argument functions $\pi(\mathbf{f}) = \lambda x. \lambda y. \mathbf{f}'(y, x)$ and
 447 $\pi(\mathbf{g}) = \lambda x. \lambda y. \mathbf{g}'(x, y)$ and $\pi(\mathbf{s}) = \lambda x. \mathbf{s}'(x)$ and $\pi(\mathbf{a}) = \pi(\mathbf{b}) = \perp_{\mathit{nat}}$ to get the requirements:

$$448 \quad \mathbf{f}'(\mathbf{s}'(Y), X) \succ \mathbf{g}'(Y, \mathbf{s}'(\mathbf{s}'(X))) \quad \mathbf{f}'(Y, X) \succsim \mathbf{g}'(\perp_{\mathit{nat}}, \perp_{\mathit{nat}})$$

$$\mathbf{g}'(X, \mathbf{s}'(Y)) \succ \mathbf{f}'(X, Y)$$

449 This is easily handled with $\mathbf{f}' \approx \mathbf{g}' \blacktriangleright \mathbf{s}'$, and $\mathit{status}(\mathbf{f}') = \mathit{status}(\mathbf{g}') = \mathit{Lex}$. This example
 450 relies on `a` and `b` being mapped to \perp_{nat} . Such use of a minimal constant originates in [49].

451 5 Dependency Pairs

452 After trying to prove non-termination using the methods in §3.1–3.2, and removing as many
 453 rules as possible with strongly monotonic reduction pairs, control is passed to the *dependency*
 454 *pair (DP) framework*. Like the first-order DP framework [26], this is an extendable framework
 455 for termination (and non-termination), which new termination methods can easily be plugged
 456 into in the form of “processors”. This framework encompasses all remaining termination
 457 techniques, but does not currently contain any processors for non-termination. The DP
 458 framework is detailed in [38, 23] and [36, Ch. 6–7]. Let us here consider a high-level overview.

459 5.1 The DP framework

460 The relatively simple form of the DP framework in Wanda operates on pairs $(\mathcal{P}, \mathcal{R})$ called *DP*
 461 *problems*. For a given AFSM, an initial pair is generated, which must be proved “finite” (also
 462 called “non-looping” in [2, 3, 38]). If this property applies, then the AFSM is terminating.

463 Now, a *processor* is a function that maps a DP problem ρ to a finite set of DP problems.
 464 Wanda has a list of processors M such that ρ is finite if and only if all elements of $M(\rho)$
 465 are finite; moreover, either $M(\rho) = \{\rho\}$, or all elements of $M(\rho)$ are strictly smaller than ρ
 466 (counting the number of elements in \mathcal{P} and \mathcal{R}). Wanda then applies the following algorithm:

- 467 1. Let A be the set containing just the initial DP problem $(\mathcal{P}, \mathcal{R})$.
- 468 2. If $A = \emptyset$ then return YES. Otherwise, choose an arbitrary element $\rho \in A$.
- 469 3. Find the first processor M in the list of processors such that $M(\rho) \neq \{\rho\}$.
- 470 4. If such a processor cannot be found, then the process has failed; return MAYBE.
- 471 5. Otherwise, let $A := (A \setminus \{\rho\}) \cup M(\rho)$, and go back to Item 2.

472 Note that, throughout the process, we retain the following property: the original AFSM
 473 is terminating if the initial DP problem is finite, which holds if and only if all elements in A
 474 are finite. This is why the conclusion in Item 2 is correct.

475 The processors used are, in order: the dependency graph, the subterm criterion, the
 476 computable subterm criterion, formative rules, and reduction pairs with usable rules (first
 477 polynomial interpretations, then **StarHorpo**). All processors are explained in [36, 23].

478 5.2 Delegation to a first-order prover

479 Following [21], the framework starts (as part of Item 1 in §5.1) by identifying the *first-order*
 480 rules in the AFSM. These are functionalised and passed to an external first-order termination
 481 tool; if the full AFSM is not orthogonal then additionally all rules in $C_\epsilon = \{c_\iota(X, Y) \Rightarrow$
 482 $X, c_\iota(X, Y) \Rightarrow Y \mid \iota \in \mathcal{S}\}$ are added (with $c_\iota : \iota \rightarrow \iota \rightarrow \iota$ fresh function symbols).¹

483 If the tool detects termination, then this is stored, as it allows all dependency pairs for
 484 these first-order rules to be omitted from the set of generated DPs. If the tool returns **NO** and
 485 no C_ϵ rules were added, then non-termination is concluded as explained in §3.3. Otherwise,
 486 the remaining cases of §3.3 are tested with a dedicated non-termination prover.

487 5.3 Static and Dynamic DPs

488 To complete item 1—so to generate the initial DP problem $(\mathcal{P}, \mathcal{R})$ —there are two different
 489 approaches, originating from distinct lines of work around the same period [38, 39]. In both
 490 cases, an AFSM $(\mathcal{F}, \mathcal{R})$ gives an initial DP problem $(\bigcup\{DP(\rho) \mid \rho \in \mathcal{R}\}, \mathcal{R} \cup \text{OptionalExtra})$,
 491 where the set $DP(\rho)$ of dependency pairs generated for a given rule varies between the two
 492 approaches. In both cases, the elements of $DP(\rho)$ with ρ a first-order rule may be omitted if
 493 the first-order part was proved terminating following §5.2. Unlike the name suggests (as this
 494 differs from the first-order definition), these dependency pairs are actually *triples* of a pattern
 495 of the form $\mathbf{f} \ell_1 \dots \ell_n$, a meta-term r and a set; this is discussed in more detail in [36, 23].

496 In the *dynamic* approach, each $DDP(\rho)$ contains triples whose second component r has
 497 a form $\mathbf{g} r_1 \dots r_m$ or $Z[r_1, \dots, r_m]$; the latter kind is called a “collapsing” DP. In the *static*
 498 approach, $SDP(\rho)$ contains no collapsing DPs, but may have DPs where $FMV(r) \not\subseteq FMV(\ell)$.
 499 Both fresh meta-variables in r and collapsing DPs are complications not present in the
 500 first-order setting, which make some of the processors weaker. The static approach for
 501 generating DPs can only be used if some restrictions on the AFSM are satisfied, but when
 502 applicable often gives an easier termination proof than the dynamic one.

503 The notion of a finite problem and the processors used in **Wanda** can all be defined generally
 504 enough to apply for both the static and dynamic approach. Hence, once the initial DP problem
 505 is generated, the same DP framework can be used for both. **Wanda** tries dynamic DPs first, and
 506 if this fails, falls back to static DPs. However, if $\bigcup\{SDP(\rho) \mid \rho \in \mathcal{R}\} \subseteq \bigcup\{DDP(\rho) \mid \rho \in \mathcal{R}\}$,
 507 this first step is omitted and only the static approach is tried.

508 ► **Example 21.** For \mathcal{R}_1 in Example 6, the dynamic approach generates $(\{(1), (2)\}, \mathcal{R}_1)$ with:

$$509 \begin{array}{ll} (1) & \text{map}^\sharp(\lambda x.Z[x]) (\text{cons } H T) \Rightarrow \text{map}^\sharp(\lambda x.Z[x]) T \quad (\emptyset) \\ (2) & \text{map}^\sharp(\lambda x.Z[x]) (\text{cons } H T) \Rightarrow Z[H] \quad (\emptyset) \end{array}$$

510 The static approach generates $(\{(1)\}, \mathcal{R}_1)$. Thus, **Wanda** does not try the dynamic approach.

¹ These rules allow for the construction of a term that can be reduced to all elements of an arbitrary finite set of terms with the same type. They are trivially discarded by many termination techniques, but may complicate analysis because they turn the system non-confluent.

	YES	NO	MAYBE	TIMEOUT	Avg. time
Full	188	16	25	32	1.14
Only rule removal	123	0	118	20	1.13
Only StarHorpo	111	0	141	9	0.24
Only interpretations	59	0	156	46	0.07
Only dependency pairs	186	0	42	33	1.02
only static DPs	152	0	86	23	0.55
only dynamic DPs	167	0	58	36	1.30
no first-order tool	183	9	47	22	0.90
no overestimation	155	16	25	65	0.75

■ **Figure 2** Experimental results on the TPDB (261 benchmarks).

511 6 Experimental results

512 To test the power of both *Wanda* as a whole, and individual techniques, various configurations
 513 of *Wanda* were tested on two data sets: (1) the “higher order union beta” benchmarks in
 514 the Termination Problem DataBase [14] (which are used in the International Termination
 515 Competition [13]), and (2) the pattern HRSs in the COPS (Confluence Problems) database
 516 [11] (which are used in the International Confluence Competition [12]), most of which were
 517 translated to AFSMs by the tool *CSI^{ho}* [43]. *Wanda* was executed with a timeout of 60
 518 seconds, on a Lenovo Thinkpad T420, using *AProVE* [25] as a first-order termination prover,
 519 and *MINISAT* [17] as a SAT-solver. The results are discussed below. Note that the average
 520 time only takes YES and NO results into account; in particular, TIMEOUTs are not considered.

521 An evaluation page with detailed results is available at:

522 <https://www.cs.ru.nl/~cynthiakop/experiments/fscd20/>

523 6.1 Benchmarks from the TPDB

524 The results on the termination problem database are given in Figure 2. The first test is
 525 *Wanda*’s default behaviour, the next three use only rule removal (with both techniques or
 526 only one), and the next three use only the DP framework (either full or with only one way
 527 of generating the initial DP problem). The final tests disable specific features in the full
 528 version: using a first-order termination tool, and overestimating AFSs as described in §2.2.
 529 The longest successful evaluation is 20.46 seconds, so not close to the 60 second timeout.

530 The tests show that rule removal is not as effective as dependency pairs, but does help
 531 a little: when it is disabled, *Wanda* loses two benchmarks (and does not gain any). This
 532 could be avoided by implementing rule removal as a processor in the DP framework, but this
 533 has thus far not been done (the implementation is not entirely straightforward due to the
 534 different requirements imposed by the DP framework). The effect of rule removal on speed is
 535 variable: rule removal often *succeeds* fast, but may take a long time to *fail*. Thus, when both
 536 are tried, the solution speed could go either way. Within rule removal, **StarHorpo** is much
 537 more powerful than polynomial interpretations, but the techniques are incomparable: there
 538 are 12 benchmarks that can be handled by interpretations but not **StarHorpo**.

539 Also the two styles of dependency pairs are incomparable: the dynamic approach seems
 540 to give a bit more power, but there are benchmarks that can be handled with static DPs
 541 and not with dynamic ones. Moreover, the static approach is significantly faster.

	YES	NO	MAYBE	TIMEOUT	Avg. time
Full	43	30	19	1	0.09
Only rule removal	37	0	56	0	0.11
Only StarHorus	33	0	60	0	0.17
Only interpretations	21	0	72	0	0.01
Only dependency pairs	43	0	49	1	0.51
only static DPs	37	0	56	0	0.26
only dynamic DPs	40	0	52	1	0.77
no first-order tool	43	30	19	1	0.07

■ **Figure 3** Experimental results on the COPS database (93 benchmarks).

542 Worth noting is that there are 16 benchmarks *Wanda* can prove non-terminating, of
 543 which 7 are found by *AProVE*. Of the remainder, manual checking shows that 7 have obvious
 544 loops, and 2 admit the $\omega\omega$ example. For termination, using *AProVE* gives a modest gain (five
 545 benchmarks). The last row deserves some further discussion. Due to unclear documentation
 546 on the competition’s format, the 85 newest benchmarks in this database are all “fake HRS”:
 547 like the system in Example 11, the left-hand sides often have subterms such as $F x y$ where
 548 F is a free variable. *Wanda* spends more time on these benchmarks than others, since not
 549 only the true translation to AFSM is considered, but also an overestimation that is often
 550 easier to handle. When overestimating is disabled, *Wanda* is faster, but significantly weaker.

551 **The first-order tool.** It is worth noting that more than fifty benchmarks in this database
 552 are actually first-order systems with one or two (typically trivial) higher-order rules. Indeed,
 553 about 25 of *Wanda*’s *TIMEOUT*s are due to *AProVE* timing out on a complicated first-order
 554 fragment. This raises the question whether the choice of first-order tool is significant.

555 The answer is ambiguous. For *non-termination*, *Wanda* relies on an explicit counter-
 556 example, which only the customised version of *AProVE* provides; without it, *Wanda* loses
 557 7 *NO*s. For *termination*, comparing *Wanda*’s performance when instead coupled with *NaTT*
 558 or *muterm*, we found that *NaTT* outperforms both *AProVE* and *muterm* by 13 benchmarks.
 559 However, this advantage is local: the “higher-order union beta” category of the TPDB has
 560 seven sub-directories, each representing a batch of (often similar) benchmarks that were
 561 added at the same time. On six of those seven, *Wanda* performs almost identically whether
 562 *AProVE*, *muterm* or *NaTT* is used: *muterm* and *AProVE* give one benchmark that *NaTT* fails,
 563 and all other answers are the same. In the last, *NaTT* wins 14 benchmarks over the others.

564 Looking at all benchmarks, we observe: the *only* cases where using a first-order tool helps,
 565 are combinations of a challenging first-order TRS and a quite simple higher-order part: it
 566 can be handled with static DPs and one of the subterm criterion processors [23]. Which
 567 first-order tool is the best for the job depends only on the form of the first-order part.

568 6.2 Benchmarks from COPS

569 Figure 3 shows the experimental results on AFSMs translated from the Confluence Problems
 570 database (COPS) [11]. Here, unlike the benchmarks from the TPDB, meta-variables are
 571 used with arguments. Even so, the comparative results between rule removal and full *Wanda*,
 572 and between static, dynamic and full dependency pairs, are similar to the TPDB results.
 573 There are relatively far more *NO* answers, which seems to be because COPS contains more
 574 non-terminating systems (and quite a few trivially so). This is explained by the purpose of
 575 the database: confluence is harder to prove for non-terminating than terminating systems.

576 **7** Practical use

577 Wanda is designed to run on a Linux terminal, and is invoked by supplying one or more input
 578 files, and zero or more runtime parameters that customise the behaviour. Runtime parameters
 579 range from purely aesthetical commands (e.g., to indicate that Wanda should use coloured
 580 output), to commands that make Wanda output properties of the given system (e.g., to
 581 indicate whether a system has η -long form) or that modify the termination checking behaviour
 582 (e.g., the previously mentioned `betafirst` parameter). Wanda has a fixed strategy—that is,
 583 techniques are always applied in the same order—but certain techniques can be disabled for
 584 practical experiments; this was done in §6. The full range of parameters is documented in
 585 the `README.txt` file included with the distribution. Some pertinent commands are:

- 586 ■ `-d <methods>` disables the given methods; for example, use `./wanda.exe -d nt,poly,dp`
 587 to disable non-termination analysis, algebra interpretations and dependency pairs; this
 588 forces Wanda to generate a proof using StarHOrpo, if one can be found;
- 589 ■ `-i <tool>` tells Wanda to use the given first-order termination tool as back-end, which must
 590 be located in the `resources/` sub-directory. If not given, Wanda uses the file “firstorder-
 591 prover”. Similarly, `-n <tool>` tells Wanda to use the given tool for *non-termination*
 592 analysis when this is done in a separate step.

593 In standard usage, Wanda takes an input file describing an AFSM or AFS, performs
 594 an analysis following §3–5 and then prints YES (a termination proof was found), NO (a
 595 non-termination proof was found) or MAYBE (neither could be proved). In the first two cases,
 596 this is followed by a human-readable proof. If more than one input file is supplied, Wanda
 597 prints the name of each file, followed by the answer and possibly proof.

598 **8** Conclusions and directions for future work

599 This paper has discussed the various techniques used in Wanda, and how they are applied.
 600 Wanda is only one of several higher-order tools, and interestingly, *incomparable* to others:
 601 there are benchmarks that Wanda can handle and other tools cannot, and vice versa. This is
 602 because all tools that have participated in the Termination Competition have focused on
 603 different techniques. For Wanda, the main termination approach is the DP framework.

604 There are many directions for improvement. Most pertinently, due to the presence of a
 605 large database of termination benchmarks in the competition format [14], Wanda has been
 606 optimised for AFSs and is decidedly weak in the presence of meta-variables with arguments.
 607 Moreover, non-termination analysis is very limited and does not take advantage of the DP
 608 framework. Other improvements could be to further extend first-order termination techniques,
 609 to build on primarily higher-order techniques like sized types [6], and to support AFSMs with
 610 polymorphic types. Automatic certification as has been done for first-order rewriting [50]
 611 would be a highly interesting direction to pursue, but would require a vast amount of work to
 612 build up the formalisation library. Finally, Wanda’s usability could be substantially improved
 613 by the addition of a web interface, for example using the EasyInterface toolkit [16].

614 A complete discussion of most techniques in Wanda and the technology behind automating
 615 them is available in the author’s PhD thesis [36]. Wanda is open-source and available from
 616 <http://wandahot.sourceforge.net/>. The snapshot that was used in the present paper
 617 (including both open- and closed-source back-ends) is available from the evaluation pages:

618 <https://www.cs.ru.nl/~cynthiakop/experiments/fscd20/>

619 — **References** —

- 620 1 P. Aczel. A general Church-Rosser theorem. Unpublished Manuscript, University of Manchester.
621 <http://www.ens-lyon.fr/LIP/REWRITING/MISC/AGeneralChurch-RosserTheorem.pdf>,
622 1978.
- 623 2 T. Aoto and Y. Yamada. Argument filterings and usable rules for simply typed dependency
624 pairs (extended abstract). In *Proceedings of HOR*, pages 21–27, 2007. Preliminary version of
625 [3].
- 626 3 T. Aoto and Y. Yamada. Argument filterings and usable rules for simply typed dependency
627 pairs. In *Proceedings of FroCoS*, volume 5749 of *LNAI*, pages 117–132. Springer, 2009.
- 628 4 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical
629 Computer Science*, 236(1-2):133–178, 2000.
- 630 5 F. Blanqui. HOT – an automated termination prover for higher-order rewriting. [http:
631 //rewriting.gforge.inria.fr/hot.html](http://rewriting.gforge.inria.fr/hot.html).
- 632 6 F. Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite
633 systems. In *Proceedings of RTA*, volume 3091 of *LNCS*, pages 24–39. Springer, 2004.
- 634 7 F. Blanqui, J. Jouannaud, and M. Okada. Inductive-data-type systems. *Theoretical Computer
635 Science*, 272(1-2):41–68, 2002.
- 636 8 F. Blanqui, J. Jouannaud, and A. Rubio. The computability path ordering: The end of a
637 quest. In *Proceedings of CSL*, volume 5213 of *LNCS*, pages 1–14. Springer, 2008.
- 638 9 C. Borralleras and A. Rubio. THOR – an automatic tool for proving termination of higher-order
639 rewriting. <https://www.cs.upc.edu/~albert/term.html>.
- 640 10 C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proceedings
641 of LPAR*, volume 2250 of *LNAI*, pages 531–547. Springer, 2001.
- 642 11 Community. Confluence Problems (COPS). <https://cops.uibk.ac.at/?q=prs>.
- 643 12 Community. The international Confluence Competition (CoCo). [http://coco.nue.riec.
tohoku.ac.jp/](http://coco.nue.riec.
644 tohoku.ac.jp/).
- 645 13 Community. Termination Portal. [http://www.termination-portal.org/wiki/Termination_
Competition](http://www.termination-portal.org/wiki/Termination_
646 Competition).
- 647 14 Community. Termination Problem DataBase (TPDB). [http://termination-portal.org/
wiki/TPDB](http://termination-portal.org/
648 wiki/TPDB).
- 649 15 N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–
650 301, 1982.
- 651 16 J. Doménech, S. Genaim, E.B. Johnsen, and R. Schlatte. EASYINTERFACE: A toolkit for rapid
652 development of GUIs for research prototype tools. In *Proceedings of FASE*, volume 10202 of
653 *LNCS*, pages 379–383. Springer, 2017.
- 654 17 N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International
655 Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, volume 2919 of
656 *LNCS*, pages 502–518. Springer, 2004. See also <http://minisat.se/>.
- 657 18 F. Emmes, T. Enger, and J. Giesl. Proving non-looping non-termination automatically. In
658 *Proceedings of IJCAR*, volume 7364 of *LNAI*, pages 225–240. Springer, 2012.
- 659 19 M. Ferreira and H. Zantema. Syntactical analysis of total termination. In *Proceedings of ALP*,
660 volume 850 of *LNCS*, pages 204–222. Springer, 1994.
- 661 20 C. Fuhs, J. Giesl, M. Parting, P. Schneider-Kamp, and S. Swiderski. Proving termination by
662 dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 47(2):133–
663 160, 2011.
- 664 21 C. Fuhs and C. Kop. Harnessing first order termination provers using higher order dependency
665 pairs. In *Proceedings of FroCoS*, volume 6989 of *LNAI*, pages 147–162. Springer, 2011.
- 666 22 C. Fuhs and C. Kop. Polynomial interpretations for higher-order rewriting. In *Proceedings of
667 RTA*, volume 15 of *LIPICs*, pages 176–192. Dagstuhl, 2012.
- 668 23 C. Fuhs and C. Kop. A static higher-order dependency pair framework. In *Proceedings of
669 ESOP*, volume 11423 of *LNCS*, pages 752–782, 2019.

- 670 24 G. Genestier. SizeChangeTool: A termination checker for rewriting dependent types. In
671 *Proceedings of HOR*, pages 14–19, 2019. [https://hal.archives-ouvertes.fr/hal-02442465/](https://hal.archives-ouvertes.fr/hal-02442465/document)
672 [document](https://hal.archives-ouvertes.fr/hal-02442465/document).
- 673 25 J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. frohn, C. Fuhs, J. Hensel, C. Otto,
674 M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Analyzing
675 program termination and complexity automatically with AProVE. *Journal of Automated*
676 *Reasoning*, 58(1):3–31, 2017.
- 677 26 J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining
678 techniques for automated termination proofs. In *Proceedings of LPAR*, volume 3452 of *LNAI*,
679 pages 301–331. Springer, 2005.
- 680 27 J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-
681 order functions. In *Proceedings of FroCoS*, volume 3717 of *LNAI*, pages 216–231. Springer,
682 2005.
- 683 28 B. Gramlich. Abstract relations between restricted termination and confluence properties of
684 rewrite systems. *Fundamenta Informaticae*, 24(1-2):3–23, 1995.
- 685 29 M. Hamana. PolySOL – an automatic tool for confluence and termination of polymorphic
686 second-order systems. <http://www.cs.gunma-u.ac.jp/hamana/polysol/>.
- 687 30 J. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of LICS*,
688 IEEE, pages 402–411, 1999.
- 689 31 S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Unpublished
690 Manuscript, University of Illinois, 1980.
- 691 32 D. Kapur, P. Musser, D. Narendran, and J. Stillman. Semi-unification. In *Proceedings of*
692 *FSTTCS*, volume 338 of *LNCS*, pages 435–454, 1988.
- 693 33 J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems:
694 introduction and survey. *Theoretical Computer Science*, 121(1-2):279 – 308, 1993.
- 695 34 J.W. Klop, V. van Oostrom, and R. de Vrijer. Iterative lexicographic path orders. In *Essays*
696 *dedicated to Joseph A. Goguen on the Occasion of his 65th Birthday*, volume 4060 of *LNCS*,
697 pages 541–554. Springer, 2006. Festschrift.
- 698 35 C. Kop. Simplifying algebraic functional systems. In *Proceedings of CAI*, volume 6742 of
699 *LNCS*, pages 201–215. Springer, 2011.
- 700 36 C. Kop. *Higher Order Termination*. PhD thesis, VU University Amsterdam, 2012.
- 701 37 C. Kop and F. van Raamsdonk. A higher-order iterative path ordering. In *Proceedings of*
702 *LPAR*, volume 5330 of *LNAI*, pages 697–711, 2008.
- 703 38 C. Kop and F. van Raamsdonk. Dynamic dependency pairs for algebraic functional systems.
704 *Logical Methods in Computer Science*, 8(2):10:1–10:51, 2012. Special Issue for RTA '11.
- 705 39 K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on
706 strong computability for higher-order rewrite systems. *IEICE Transactions on Information*
707 *and Systems*, 92(10):2007–2015, 2009.
- 708 40 K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proceedings*
709 *of PPDP*, volume 1702 of *LNCS*, pages 47–61. Springer, 1999.
- 710 41 S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting
711 systems. *Information Processing letters*, 95(4):446–453, 2005.
- 712 42 D. Miller. A logic programming language with lambda-abstraction, function variables, and
713 simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
- 714 43 J. Nagele. CoCo 2016 participant: CSI^{ho} 0.2. [http://coco.nue.riec.tohoku.ac.jp/2016/](http://coco.nue.riec.tohoku.ac.jp/2016/papers/csiho.pdf)
715 [papers/csiho.pdf](http://coco.nue.riec.tohoku.ac.jp/2016/papers/csiho.pdf); tool webpage: <http://cl-informatik.uibk.ac.at/software/csi/ho/>.
- 716 44 T. Nipkow. Higher-order critical pairs. In *Proceedings of LICS*, pages 342–349. IEEE, 1991.
- 717 45 K. Onozawa, K. Kikuchi, T. Aoto, and Y. Toyama. ACPH: System description for CoCo 2016.
718 <http://coco.nue.riec.tohoku.ac.jp/2016/papers/acph.pdf>.
- 719 46 É. Payet. Loop detection in term rewriting using the eliminating unfoldings. *Theoretical*
720 *Computer Science*, 403(2-3):307–327, 2008.

- 721 47 É. Payet. Guided unfoldings for finding loops in standard term rewriting. In *Proceedings of*
722 *LOPSTR*, volume 11408 of *LNCS*, pages 22–37, 2018.
- 723 48 J.C. van de Pol. *Termination of Higher-order Rewrite Systems*. PhD thesis, University of
724 Utrecht, 1996.
- 725 49 R. Thiemann, G. Allais, and J. Nagele. On the formalization of termination techniques based
726 on multiset orderings. In *Proceedings of RTA*, volume 15 of *LIPICs*, pages 339–354. Dagstuhl,
727 2012.
- 728 50 R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proceedings*
729 *of TPHOLs*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009.
- 730 51 Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems.
731 *Information Processing Letters*, 25(1):141–143, 1987.
- 732 52 H. Zantema. Termination of context-sensitive rewriting. In *Proceedings of RTA*, volume 1232
733 of *LNCS*, pages 172–186, 1997.