First-Order Formative Rules*

Carsten Fuhs[1] and Cynthia Kop[2]

[1] University College London, Dept. of Computer Science, London WC1E 6BT, UK
[2] University of Innsbruck, Institute of Computer Science, 6020 Innsbruck, Austria

Abstract This paper discusses the method of *formative rules* for first-order term rewriting, which was previously defined for a higher-order setting. Dual to the well-known *usable rules*, formative rules allow dropping some of the term constraints that need to be solved during a termination proof. Compared to the higher-order definition, the first-order setting allows for significant improvements of the technique.

1 Introduction

In [12,13] C. Kop and F. van Raamsdonk introduce the notion of *formative rules*. The technique is similar to the method of *usable rules* [1,9,10], which is commonly used in termination proofs, but has different strengths and weaknesses.

Since, by [15], the more common *first-order* style of term rewriting, both with and without types, can be seen as a subclass of the formalism of [13], this result immediately applies to first-order rewriting. In an untyped setting, we will, however, lose some of its strength, as sorts play a relevant role in formative rules.

On the other hand, by omitting the complicating aspects of higher-order term rewriting (such as λ-abstraction and "collapsing" rules $l \to x \cdot y$) we also gain possibilities not present in the original setting; both things which *have not* been done, as the higher-order dependency pair framework [11] is still rather limited, and things which *cannot* be done, at least with current theory. Therefore, in this paper, we will redefine the method for (many-sorted) first-order term rewriting.

New compared to [13], we will integrate formative rules into the dependency pair framework [7], which is the basis of most contemporary termination provers for first-order term rewriting. Within this framework, formative rules are used either as a stand-alone processor or with reduction pairs, and can be coupled with usable rules and argument filterings. We also formulate a semantic characterisation of formative rules, to enable future generalisations of the definition. Aside from this, we present a (new) way to weaken the detrimental effect of collapsing rules.

This paper is organised as follows. After the preliminaries in Section 2, a first definition of formative rules is given and then generalised in Section 3. Section 4 shows various ways to use formative rules in the dependency pair framework. Section 5 gives an alternative way to deal with collapsing rules. In Section 6 we consider innermost termination, Section 7 describes implementation and experiments, and in Section 8 we point out possible future work and conclude. All proofs and an improved formative rules approximation are provided in [5].

* Support by EPSRC & the Austrian Science Fund (FWF) international project I963.

2 Preliminaries

We consider *many-sorted term rewriting*: term rewriting with *sorts*, basic types. While sorts are not usually considered in studies of first-order term rewrite systems (TRSs) and for instance the Termination Problems Data Base[3] does not include them (for first-order TRSs),[4] they are a natural addition; in typical applications there is little need to allow untypable terms like `3`+`apple`. Even when no sorts are present, a standard TRS can be seen as a many-sorted TRS with only one sort.[5]

Many-sorted TRSs We assume given a non-empty set $\mathcal{S}$ of *sorts*; these are typically things like `Nat` or `Bool`, or (for representing unsorted systems) $\mathcal{S}$ might be the set with a single sort $\{\mathtt{o}\}$. A *sort declaration* is a sequence $[\kappa_1 \times \ldots \times \kappa_n] \Rightarrow \iota$ where ι and all κ_i are sorts. A sort declaration $[] \Rightarrow \iota$ is just denoted ι.

A *many-sorted signature* is a set Σ of function symbols f, each equipped with a sort declaration σ, notation $f : \sigma \in \Sigma$. Fixing a many-sorted signature Σ and an infinite set $\mathcal{V}$ of sorted variables, the set of *terms* consists of those expressions s over Σ and $\mathcal{V}$ for which we can derive $s : \iota$ for some sort ι, using the clauses:

$$x : \iota \text{ if } x : \iota \in \mathcal{V}$$
$$f(s_1, \ldots, s_n) : \iota \text{ if } f : [\kappa_1 \times \ldots \times \kappa_n] \Rightarrow \iota \in \Sigma \text{ and } s_1 : \kappa_1,\ \ldots,\ s_n : \kappa_n$$

We often denote $f(s_1, \ldots, s_n)$ as just $f(\boldsymbol{s})$. Clearly, every term has a unique sort. Let $Var(s)$ be the set of all variables occurring in a term s. A term s is *linear* if every variable in $Var(s)$ occurs only once in s. A term t is a *subterm* of another term s, notation $s \trianglerighteq t$, if either $s = t$ or $s = f(s_1, \ldots, s_n)$ and some $s_i \trianglerighteq t$. A *substitution* γ is a mapping from variables to terms of the same sort; the application $s\gamma$ of a substitution γ on a term s is s with each $x \in \mathsf{domain}(\gamma)$ replaced by $\gamma(x)$.

A *rule* is a pair $\ell \to r$ of terms with the same sort such that ℓ is not a variable.[6] A rule is *left-linear* if ℓ is linear, and *collapsing* if r is a variable. Given a set of rules $\mathcal{R}$, the *reduction relation* $\to_{\mathcal{R}}$ is given by: $\ell\gamma \to_{\mathcal{R}} r\gamma$ if $\ell \to r \in \mathcal{R}$ and γ a substitution; $f(\ldots, s_i, \ldots) \to_{\mathcal{R}} f(\ldots, s_i', \ldots)$ if $s_i \to_{\mathcal{R}} s_i'$. A term s is in *normal form* if there is no t such that $s \to_{\mathcal{R}} t$.

The relation $\to_{\mathcal{R}}^*$ is the transitive-reflexive closure of $\to_{\mathcal{R}}$. If there is a rule $f(\boldsymbol{l}) \to r \in \mathcal{R}$ we say that f is a *defined symbol*; otherwise f is a *constructor*.

A *many-sorted term rewrite system (MTRS)* is a pair $(\Sigma, \mathcal{R})$ with signature Σ and a set $\mathcal{R}$ of rules $\ell \to r$ with $Var(r) \subseteq Var(\ell)$. A term s is *terminating* if there is no infinite reduction $s \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \ldots$ An MTRS is terminating if all terms are.

Example 1. An example of a many-sorted TRS $(\Sigma, \mathcal{R})$ with more than one sort is the following system, which uses lists, natural numbers and a `RESULT` sort:

[3] More information on the *TPDB*: `http://termination-portal.org/wiki/TPDB`

[4] This may also be due to the fact that currently most termination tools for first-order rewriting only make very limited use of the additional information carried by types.

[5] However, the method of this paper is stronger given more sorts. We may be able to (temporarily) infer richer sorts, however. We will say more about this in Section 6.

[6] Often also $Var(r) \subseteq Var(\ell)$ is required. However, we use *filtered* rules $\overline{\pi}(\ell) \to \overline{\pi}(r)$ later, where the restriction is inconvenient. As a rule is non-terminating if $Var(r) \not\subseteq Var(\ell)$, as usual we forbid such rules in the input $\mathcal{R}$ and in dependency pair problems.

$$\begin{array}{rlrlrl}
\mathtt{0} : & \mathtt{NAT} & \mathtt{Cons} : & [\mathtt{NAT} \times \mathtt{LIST}] \Rightarrow \mathtt{LIST} & \mathtt{Run} : & [\mathtt{LIST}] \Rightarrow \mathtt{RESULT} \\
\mathtt{S} : & [\mathtt{NAT}] \Rightarrow \mathtt{NAT} & \mathtt{Ack} : & [\mathtt{NAT} \times \mathtt{NAT}] \Rightarrow \mathtt{NAT} & \mathtt{Return} : & [\mathtt{NAT}] \Rightarrow \mathtt{RESULT} \\
\mathtt{Nil} : & \mathtt{LIST} & \mathtt{Big} : & [\mathtt{NAT} \times \mathtt{LIST}] \Rightarrow \mathtt{NAT} & \mathtt{Rnd} : & [\mathtt{NAT}] \Rightarrow \mathtt{NAT} \\
\mathtt{Err} : & \mathtt{RESULT} & \mathtt{Upd} : & [\mathtt{LIST}] \Rightarrow \mathtt{LIST} & &
\end{array}$$

$$\begin{array}{rlrl}
1. & \mathtt{Rnd}(x) \to x & 6. & \mathtt{Big}(x, \mathtt{Nil}) \to x \\
2. & \mathtt{Rnd}(\mathtt{S}(x)) \to \mathtt{Rnd}(x) & 7. & \mathtt{Big}(x, \mathtt{Cons}(y, z)) \to \mathtt{Big}(\mathtt{Ack}(x, y), \mathtt{Upd}(z)) \\
3. & \mathtt{Upd}(\mathtt{Nil}) \to \mathtt{Nil} & 8. & \mathtt{Upd}(\mathtt{Cons}(x, y)) \to \mathtt{Cons}(\mathtt{Rnd}(x), \mathtt{Upd}(y)) \\
4. & \mathtt{Run}(\mathtt{Nil}) \to \mathtt{Err} & 9. & \mathtt{Run}(\mathtt{Cons}(x, y)) \to \mathtt{Return}(\mathtt{Big}(x, y)) \\
5. & \mathtt{Ack}(\mathtt{0}, y) \to \mathtt{S}(y) & 10. & \mathtt{Ack}(\mathtt{S}(x), y) \to \mathtt{Ack}(x, \mathtt{S}(y)) \\
 & & 11. & \mathtt{Ack}(\mathtt{S}(x), \mathtt{S}(y)) \to \mathtt{Ack}(x, \mathtt{Ack}(\mathtt{S}(x), y))
\end{array}$$

$\mathtt{Run}(lst)$ calculates a potentially very large number, depending on the elements of lst and some randomness. We have chosen this example because it will help to demonstrate the various aspects of formative rules, without being too long.

The Dependency Pair Framework As a basis to study termination, we will use the *dependency pair (DP) framework* [7], adapted to include sorts.

Given an MTRS $(\Sigma, \mathcal{R})$, let $\Sigma^\sharp = \Sigma \cup \{f^\sharp : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \mathtt{dpsort} \mid f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa \in \Sigma \wedge f$ a defined symbol of $\mathcal{R}\}$, where $\mathtt{dpsort}$ is a fresh sort. The set $\mathsf{DP}(\mathcal{R})$ of *dependency pairs (DPs)* of $\mathcal{R}$ consists of all rules of the form $f^\sharp(l_1, \ldots, l_n) \to g^\sharp(r_1, \ldots, r_m)$ where $f(\boldsymbol{l}) \to r \in \mathcal{R}$ and $r \trianglerighteq g(\boldsymbol{r})$ with g a defined symbol.

Example 2. The dependency pairs of the system in Example 1 are:

$$\begin{array}{rlrl}
\mathtt{Rnd}^\sharp(\mathtt{S}(x)) \to \mathtt{Rnd}^\sharp(x) & & \mathtt{Big}^\sharp(x, \mathtt{Cons}(y, z)) \to \mathtt{Big}^\sharp(\mathtt{Ack}(x, y), \mathtt{Upd}(z)) \\
\mathtt{Upd}^\sharp(\mathtt{Cons}(x, y)) \to \mathtt{Rnd}^\sharp(x) & & \mathtt{Big}^\sharp(x, \mathtt{Cons}(y, z)) \to \mathtt{Ack}^\sharp(x, y) \\
\mathtt{Upd}^\sharp(\mathtt{Cons}(x, y)) \to \mathtt{Upd}^\sharp(y) & & \mathtt{Big}^\sharp(x, \mathtt{Cons}(y, z)) \to \mathtt{Upd}^\sharp(z) \\
\mathtt{Run}^\sharp(\mathtt{Cons}(x, y)) \to \mathtt{Big}^\sharp(x, y) & & \mathtt{Ack}^\sharp(\mathtt{S}(x), \mathtt{S}(y)) \to \mathtt{Ack}^\sharp(x, \mathtt{Ack}(\mathtt{S}(x), y)) \\
\mathtt{Ack}^\sharp(\mathtt{S}(x), y) \to \mathtt{Ack}^\sharp(x, \mathtt{S}(y)) & & \mathtt{Ack}^\sharp(\mathtt{S}(x), \mathtt{S}(y)) \to \mathtt{Ack}^\sharp(\mathtt{S}(x), y)
\end{array}$$

For sets $\mathcal{P}$ and $\mathcal{R}$ of rules, an infinite $(\mathcal{P}, \mathcal{R})$-chain is a sequence $[(\ell_i \to r_i, \gamma_i) \mid i \in \mathbb{N}]$ where each $\ell_i \to r_i \in \mathcal{P}$ and γ_i is a substitution such that $r_i\gamma_i \to_{\mathcal{R}}^* \ell_{i+1}\gamma_{i+1}$. This chain is *minimal* if each $r_i\gamma_i$ is terminating with respect to $\to_{\mathcal{R}}$.

Theorem 3. *(following [1,7,9,10]) An MTRS $(\Sigma, \mathcal{R})$ is terminating if and only if there is no infinite minimal $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$-chain.*

A *DP problem* is a triple $(\mathcal{P}, \mathcal{R}, f)$ with $\mathcal{P}$ and $\mathcal{R}$ sets of rules and $f \in \{\mathsf{m}, \mathsf{a}\}$ (denoting $\{\mathsf{minimal}, \mathsf{arbitrary}\}$).[7] A DP problem $(\mathcal{P}, \mathcal{R}, f)$ is *finite* if there is no infinite $(\mathcal{P}, \mathcal{R})$-chain, which is minimal if $f = \mathsf{m}$. A *DP processor* is a function which maps a DP problem to a set of DP problems. A processor *proc* is *sound* if, for all DP problems A: if all $B \in proc(A)$ are finite, then A is finite.

The goal of the DP framework is, starting with a set $D = \{(\mathsf{DP}(\mathcal{R}), \mathcal{R}, \mathsf{m})\}$, to reduce D to $\emptyset$ using sound processors. Then we may conclude termination of the initial MTRS $(\Sigma, \mathcal{R})$.[8] Various common processors use a *reduction pair*, a pair

[7] Here we do not modify the signature $\Sigma^\sharp$ of a DP problem, so we leave $\Sigma^\sharp$ implicit.

[8] The full DP framework [7] can also be used for proofs of *non-termination*. Indeed, by [7, Lemma 2], all processors introduced in this paper (except Theorem 17 for innermost rewriting) are "complete" and may be applied in a non-termination proof.

$(\succsim, \succ)$ of a monotonic, stable (closed under substitutions) quasi-ordering $\succsim$ on terms and a well-founded, stable ordering $\succ$ compatible with $\succsim$ (i.e., $\succ \cdot \succsim \subseteq \succ$).

Theorem 4. *(following [1,7,9,10]) Let $(\succsim, \succ)$ be a reduction pair. The processor which maps a DP problem $(\mathcal{P}, \mathcal{R}, f)$ to the following result is sound:*

- $\{(\mathcal{P} \setminus \mathcal{P}^{\succ}, \mathcal{R}, f)\}$ *if:*
  - $\ell \succ r$ *for* $\ell \to r \in \mathcal{P}^{\succ}$ *and* $\ell \succsim r$ *for* $\ell \to r \in \mathcal{P} \setminus \mathcal{P}^{\succ}$ *(with* $\mathcal{P}^{\succ} \subseteq \mathcal{P}$*);*
  - $\ell \succsim r$ *for* $\ell \to r \in \mathcal{R}$*.*
- $\{(\mathcal{P}, \mathcal{R}, f)\}$ *otherwise*

Here, we must orient all elements of $\mathcal{R}$ with $\succsim$. As there are many processors which remove elements from $\mathcal{P}$ and few which remove from $\mathcal{R}$, this may give many constraints. *Usable rules*, often combined with *argument filterings*, address this:

Definition 5. *(following [9,10]) Let Σ be a signature and $\mathcal{R}$ a set of rules. An* argument filtering *is a function that maps each $f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa$ to a set $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$.*[9] *The* usable rules *of a term t with respect to an argument filtering π are defined as the smallest set $UR(t, \mathcal{R}, \pi) \subseteq \mathcal{R}$ such that:*

- *if $\mathcal{R}$ is not finitely branching (i.e. there are terms with infinitely many direct reducts), then $UR(t, \mathcal{R}, \pi) = \mathcal{R}$;*
- *if $t = f(t_1, \ldots, t_n)$, then $UR(t_i, \mathcal{R}, \pi) \subseteq UR(t, \mathcal{R}, \pi)$ for all $i \in \pi(f)$;*
- *if $t = f(t_1, \ldots, t_n)$, then $\{\ell \to r \in \mathcal{R} \mid \ell = f(\ldots)\} \subseteq UR(t, \mathcal{R}, \pi)$;*
- *if $\ell \to r \in UR(t, \mathcal{R}, \pi)$, then $UR(r, \mathcal{R}, \pi) \subseteq UR(t, \mathcal{R}, \pi)$.*

For a set of rules $\mathcal{P}$, we define $UR(\mathcal{P}, \mathcal{R}, \pi) = \bigcup_{s \to t \in \mathcal{P}} UR(t, \mathcal{R}, \pi)$.

Argument filterings π are used to disregard arguments of certain function symbols. Given π, let $f_\pi : [\iota_{i_1} \times \ldots \times \iota_{i_k}] \Rightarrow \kappa$ be a fresh function symbol for all f with $\pi(f) = \{i_1, \ldots, i_k\}$ and $i_1 < \ldots < i_k$, and define $\overline{\pi}(x) = x$ for x a variable, and $\overline{\pi}(f(s_1, \ldots, s_n)) = f_\pi(\overline{\pi}(s_{i_1}), \ldots, \overline{\pi}(s_{i_k}))$ if $\pi(f) = \{i_1, \ldots, i_k\}$ and $i_1 < \ldots < i_k$. For a set of rules $\mathcal{R}$, let $\overline{\pi}(\mathcal{R}) = \{\overline{\pi}(l) \to \overline{\pi}(r) \mid l \to r \in \mathcal{R}\}$. The idea of usable rules is to only consider rules relevant to the pairs in $\mathcal{P}$ after applying $\overline{\pi}$.

Combining usable rules, argument filterings and reduction pairs, we obtain:

Theorem 6. *([9,10]) Let $(\succsim, \succ)$ be a reduction pair and π an argument filtering. The processor which maps a DP problem $(\mathcal{P}, \mathcal{R}, f)$ to the following result is sound:*

- $\{(\mathcal{P} \setminus \mathcal{P}^{\succ}, \mathcal{R}, \mathsf{m})\}$ *if $f = \mathsf{m}$ and:*
  - $\overline{\pi}(\ell) \succ \overline{\pi}(r)$ *for* $\ell \to r \in \mathcal{P}^{\succ}$ *and* $\overline{\pi}(\ell) \succsim \overline{\pi}(r)$ *for* $\ell \to r \in \mathcal{P} \setminus \mathcal{P}^{\succ}$*;*
  - $\overline{\pi}(\ell) \succsim \overline{\pi}(r)$ *for* $\ell \to r \in UR(\mathcal{P}, \mathcal{R}, \pi) \cup \mathcal{C}_\epsilon$*,*
    where $\mathcal{C}_\epsilon = \{\mathsf{c}_\iota(x, y) \to x, \mathsf{c}_\iota(x, y) \to y \mid$ *all sorts* $\iota\}$*.*
- $\{(\mathcal{P}, \mathcal{R}, f)\}$ *otherwise*

We define $UR(\mathcal{P}, \mathcal{R})$ as $UR(\mathcal{P}, \mathcal{R}, \pi_{\mathcal{T}})$, where $\pi_{\mathcal{T}}$ is the *trivial filtering*: $\pi_{\mathcal{T}}(f) = \{1, \ldots, n\}$ for $f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa \in \Sigma$. Then Theorem 6 is exactly the standard reduction pair processor, but with constraints on $UR(\mathcal{P}, \mathcal{R}) \cup \mathcal{C}_\epsilon$ instead of $\mathcal{R}$. We could also use a processor which maps $(\mathcal{P}, \mathcal{R}, \mathsf{m})$ to $\{(\mathcal{P}, UR(\mathcal{P}, \mathcal{R}) \cup \mathcal{C}_\epsilon, \mathsf{a})\}$, but as this loses the minimality flag, it is usually not a good idea (various processors need this flag, including usable rules!) and can only be done once.

[9] Usual definitions of argument filterings also allow $\pi(f) = i$, giving $\overline{\pi}(f(\boldsymbol{s})) = \overline{\pi}(s_i)$, but for usable rules, $\pi(f) = i$ is treated the same as $\pi(f) = \{i\}$, cf. [9, Section 4].

3 Formative Rules

Where *usable rules* [1,9,10] are defined primarily by the right-hand sides of $\mathcal{P}$ and $\mathcal{R}$, the *formative rules* discussed here are defined by the left-hand sides. This has consequences; most importantly, we cannot handle non-left-linear rules very well.

We fix a signature Σ. A term $s : \iota$ *has shape* f with $f : [\boldsymbol{\kappa}] \Rightarrow \iota \in \Sigma$ if either $s = f(r_1, \dots, r_n)$, or s is a variable of sort ι. That is, there exists some γ with $s\gamma = f(\dots)$: one can *specialise* s to have f as its root symbol.

Definition 7. *Let $\mathcal{R}$ be a set of rules. The* basic formative rules *of a term t are defined as the smallest set $FR_{\mathsf{base}}(t, \mathcal{R}) \subseteq \mathcal{R}$ such that:*

- *if t is not linear, then $FR_{\mathsf{base}}(t, \mathcal{R}) = \mathcal{R}$;*
- *if $t = f(t_1, \dots, t_n)$, then $FR_{\mathsf{base}}(t_i, \mathcal{R}) \subseteq FR_{\mathsf{base}}(t, \mathcal{R})$;*
- *if $t = f(t_1, \dots, t_n)$, then $\{\ell \to r \in \mathcal{R} \mid r \text{ has shape } f\} \subseteq FR_{\mathsf{base}}(t, \mathcal{R})$;*
- *if $\ell \to r \in FR_{\mathsf{base}}(t, \mathcal{R})$, then $FR_{\mathsf{base}}(\ell, \mathcal{R}) \subseteq FR_{\mathsf{base}}(t, \mathcal{R})$.*

For rules $\mathcal{P}$, let $FR_{\mathsf{base}}(\mathcal{P}, \mathcal{R}) = \bigcup_{s \to t \in \mathcal{P}} FR_{\mathsf{base}}(s, \mathcal{R})$. Note that $FR_{\mathsf{base}}(x,\mathcal{R}) = \emptyset$.

Note the strong symmetry with Definition 5. We have omitted the argument filtering π here, because the definitions are simpler without it. In Section 4 we will see how we can add argument filterings back in without changing the definition.

Example 8. In the system from Example 1, consider $\mathcal{P} = \{\mathtt{Big}^\sharp(x, \mathtt{Cons}(y, z)) \to \mathtt{Big}^\sharp(\mathtt{Ack}(x, y), \mathtt{Upd}(z))\}$. The symbols in the left-hand side are just $\mathtt{Big}^\sharp$ (which has sort $\mathtt{dpsort}$, which is not used in $\mathcal{R}$) and $\mathtt{Cons}$. Thus, $FR_{\mathsf{base}}(\mathcal{P}, \mathcal{R}) = \{8\}$.

Intuitively, the formative rules of a dependency pair $\ell \to r$ are those rules which might contribute to creating the pattern ℓ. In Example 8, to reduce a term $\mathtt{Big}^\sharp(\mathtt{Ack}(\mathtt{S}(\mathtt{0}), \mathtt{0}), \mathtt{Upd}(\mathtt{Cons}(\mathtt{0}, \mathtt{Nil})))$ to an instance of $\mathtt{Big}^\sharp(x, \mathtt{Cons}(y, z))$, a single step with the $\mathtt{Upd}$ rule 8 gives $\mathtt{Big}^\sharp(\mathtt{Ack}(\mathtt{S}(\mathtt{0}), \mathtt{0}), \mathtt{Cons}(\mathtt{Rnd}(\mathtt{0}), \mathtt{Upd}(\mathtt{Nil})))$; we need not reduce the $\mathtt{Ack}()$ or $\mathtt{Rnd}()$ subterms for this. To create a *non*-linear pattern, any rule could contribute, as a step deep inside a term may be needed.

Example 9. Consider $\Sigma = \{\mathtt{a}, \mathtt{b} : \mathtt{A}, \mathtt{f}^\sharp : [\mathtt{B} \times \mathtt{B}] \Rightarrow \mathtt{dpsort}, \mathtt{h} : [\mathtt{A}] \Rightarrow \mathtt{B}\}$, $\mathcal{R} = \{\mathtt{a} \to \mathtt{b}\}$ and $\mathcal{P} = \{\mathtt{f}^\sharp(x, x) \to \mathtt{f}^\sharp(\mathtt{h}(\mathtt{a}), \mathtt{h}(\mathtt{b}))\}$. Without the linearity restriction, $FR_{\mathsf{base}}(\mathcal{P}, \mathcal{R})$ would be $\emptyset$, as $\mathtt{dpsort}$ does not occur in the rules and $FR_{\mathsf{base}}(x, \mathcal{R}) = \emptyset$. But there is no infinite $(\mathcal{P}, \emptyset)$-chain, while we do have an infinite $(\mathcal{P}, \mathcal{R})$-chain, with $\gamma_i = [x := \mathtt{h}(\mathtt{b})]$ for all i. The $\mathtt{a} \to \mathtt{b}$ rule is needed to make $\mathtt{h}(\mathtt{a})$ and $\mathtt{h}(\mathtt{b})$ equal. Note that this happens even though the sort of x does not occur in $\mathcal{R}$!

Thus, as we will see, in an infinite $(\mathcal{P}, \mathcal{R})$-chain we can limit interest to rules in $FR_{\mathsf{base}}(\mathcal{P}, \mathcal{R})$. We call these *basic* formative rules because while they demonstrate the concept, in practice we would typically use more advanced extensions of the idea. For instance, following the *TCap* idea of [8, Definition 11], a rule $l \to f(\mathtt{0})$ does not need to be a formative rule of $f(\mathtt{S}(x)) \to r$ if $\mathtt{0}$ is a constructor.

To use formative rules with DPs, we will show that any $(\mathcal{P}, \mathcal{R})$-chain can be altered so that the $r_i\gamma_i \to_{\mathcal{R}}^* \ell_{i+1}\gamma_{i+1}$ reduction has a very specific form (which uses only formative rules of ℓ_{i+1}). To this end, we consider *formative reductions.* A formative reduction is a reduction where, essentially, a rewriting step is only done if it is needed to obtain a result of the right form.

Definition 10 (Formative Reduction). *For a term ℓ, substitution γ and term s, we say $s \to_{\mathcal{R}}^* \ell\gamma$ by a* formative ℓ-reduction *if one of the following holds:*

1. *ℓ is non-linear;*
2. *ℓ is a variable and $s = \ell\gamma$;*
3. *$\ell = f(l_1, \dots, l_n)$ and $s = f(s_1, \dots, s_n)$ and each $s_i \to_{\mathcal{R}}^* l_i\gamma$ by a formative l_i-reduction;*
4. *$\ell = f(l_1, \dots, l_n)$ and there are a rule $\ell' \to r' \in \mathcal{R}$ and a substitution δ such that $s \to_{\mathcal{R}}^* \ell'\delta$ by a formative ℓ'-reduction and $r'\delta = f(t_1, \dots, t_n)$ and each $t_i \to_{\mathcal{R}}^* l_i\gamma$ by a formative l_i-reduction.*

Point 2 is the key: a reduction $s \to_{\mathcal{R}}^* x\gamma$ must be postponed. Formative reductions are the base of a semantic definition of formative rules:

Definition 11. *A function FR that maps a term ℓ and a set of rules $\mathcal{R}$ to a set $FR(\ell, \mathcal{R}) \subseteq \mathcal{R}$ is a* formative rules approximation *if for all s and γ: if $s \to_{\mathcal{R}}^* \ell\gamma$ by a formative ℓ-reduction, then this reduction uses only rules in $FR(\ell, \mathcal{R})$.*

Given a formative rules approximation FR, let $FR(\mathcal{P}, \mathcal{R}) = \bigcup_{s \to t \in \mathcal{P}} FR(s, \mathcal{R})$.

As might be expected, FR_{base} is indeed a formative rules approximation:

Lemma 12. *A formative ℓ-reduction $s \to_{\mathcal{R}}^* \ell\gamma$ uses only rules in $FR_{\mathsf{base}}(\ell, \mathcal{R})$.*

Proof. By induction on the definition of a formative ℓ-reduction. If ℓ is non-linear, then $FR_{\mathsf{base}}(\ell, \mathcal{R}) = \mathcal{R}$, so this is clear. If $s = \ell\gamma$ then no rules play a part.

If $s = f(s_1, \dots, s_n)$ and $\ell = f(l_1, \dots, l_n)$ and each $s_i \to_{\mathcal{R}}^* l_i\gamma$ by a formative l_i-reduction, then by the induction hypothesis each formative l_i-reduction $s_i \to_{\mathcal{R}}^* l_i\gamma$ uses only rules in $FR_{\mathsf{base}}(l_i, \mathcal{R})$. Observing that by definition $FR_{\mathsf{base}}(l_i, \mathcal{R}) \subseteq FR_{\mathsf{base}}(\ell, \mathcal{R})$, we see that all steps of the reduction use rules in $FR_{\mathsf{base}}(\ell, \mathcal{R})$.

If $s \to_{\mathcal{R}}^* \ell'\delta \to_{\mathcal{R}} r'\delta = f(t_1, \dots, t_n) \to_{\mathcal{R}}^* f(l_1, \dots, l_n)\gamma = \ell\gamma$, then by the same reasoning the reduction $r'\delta \to_{\mathcal{R}}^* \ell\gamma$ uses only formative rules of ℓ, and by the induction hypothesis $s \to_{\mathcal{R}}^* \ell'\delta$ uses only formative rules of ℓ'. Noting that r' obviously has the same sort as ℓ, and either r' is a variable or a term $f(r'_1, \dots, r'_n)$, we see that r' has shape f, so $\ell' \to r' \in FR_{\mathsf{base}}(\ell, \mathcal{R})$. Therefore $FR_{\mathsf{base}}(\ell', \mathcal{R}) \subseteq FR_{\mathsf{base}}(\ell, \mathcal{R})$, so all rules in the reduction are formative rules of ℓ. □

In the following, we will assume a fixed formative rules approximation *FR*. The relevance of formative rules is clear from their definition: if we can prove that a $(\mathcal{P}, \mathcal{R})$-chain can be altered to use formative reductions in the $\to_{\mathcal{R}}$ steps, then we can drop all non-formative rules from a DP problem.

The key result in this paper is the following technical lemma, which allows us to alter a reduction $s \to_{\mathcal{R}}^* \ell\gamma$ to a formative reduction (by changing γ):

Lemma 13. *If $s \to_{\mathcal{R}}^* \ell\gamma$ for some terms s, ℓ and a substitution γ on domain $Var(\ell)$, then there is a substitution δ on the same domain such that $s \to_{FR(\ell,\mathcal{R})}^* \ell\delta$ by a formative ℓ-reduction.*

Proof. For non-linear ℓ this is clear, choosing $\delta := \gamma$. So let ℓ be a linear term. By definition of *FR*, it suffices to see that $s \to_{\mathcal{R}}^* \ell\delta$ by a formative ℓ-reduction. This follows from the following claim: *If $s \mathrel{-\!\!\!\|\!\!\!\!\to}_{\mathcal{R}}^{k} \ell\gamma$ for some k, term s, linear term ℓ and substitution γ on domain $Var(\ell)$, then there is a substitution δ on $Var(\ell)$ such that $s \to_{\mathcal{R}}^* \ell\delta$ by a formative ℓ-reduction, and each $\delta(x) \mathrel{-\!\!\!\|\!\!\!\!\to}_{\mathcal{R}}^{k} \gamma(x)$.*

Here, the parallel reduction relation $\mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}$ is defined by: $x \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}} x$; $\ell\gamma \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}} r\gamma$ for $\ell \to r \in \mathcal{R}$; if $s_i \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}} t_i$ for $1 \leq i \leq n$, then $f(s_1, \dots, s_n) \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}} f(t_1, \dots, t_n)$. The notation $\mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k}$ indicates k *or fewer* successive $\mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}$ steps. Note that $\mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}$ is reflexive, and if each $s_i \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{N_i} t_i$, then $f(\boldsymbol{s}) \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{\max(N_1,\dots,N_n)} f(\boldsymbol{t})$.

We prove the claim by induction first on k, second on the size of ℓ.

If ℓ is a variable we are immediately done, choosing $\delta := [\ell := s]$.

Otherwise, let $\ell = f(l_1, \dots, l_n)$ and $\gamma = \gamma_1 \cup \ldots \cup \gamma_n$ such that all γ_i have disjoint domains and each $l_i\gamma_i = l_i\gamma$; this is possible due to linearity.

First suppose the reduction $s \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k} \ell\gamma$ uses no topmost steps. Thus, we can write $s = f(s_1, \dots, s_n)$ and each $s_i \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k} l_i\gamma$. By the second induction hypothesis we can find $\delta_1, \dots, \delta_n$ such that each $s_i \to_{\mathcal{R}}^{*} l_i\delta_i$ by a formative l_i-reduction and each $\delta_i(x) \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k} \gamma_i(x)$. Choose $\delta := \delta_1 \cup \ldots \cup \delta_n$; this is well-defined by the assumption on the disjoint domains. Then $s \to_{\mathcal{R}}^{*} \ell\delta$ by a formative ℓ-reduction.

Alternatively, a topmost step was done, which cannot be parallel with other steps: $s \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{m} \ell'\gamma' \to_{\mathcal{R}} r'\gamma' \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k-m-1} \ell\gamma$ for some $\ell' \to r' \in \mathcal{R}$ and substitution γ'; we can safely assume that $r'\gamma' \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k-m-1} \ell\gamma$ does not use topmost steps (otherwise we could just choose a later step). Since $m < k$, the first induction hypothesis provides δ' such that $s \to_{\mathcal{R}}^{*} \ell'\delta'$ by a formative ℓ'-reduction and each $\delta'(x) \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{m} \gamma'(x)$. But then also $r'\delta' \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{m} r'\gamma'$. Since $r'\gamma' \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k-m-1} \ell\gamma$, we have that $r'\delta' \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k-1} \ell\gamma$. Thus, by the first induction hypothesis, there is δ such that $r'\delta' \to_{\mathcal{R}}^{*} \ell\delta$ by a formative ℓ-reduction, and each $\delta(x) \mathrel{+\!\!\!\!\!\rightarrow}_{\mathcal{R}}^{k-1} \gamma(x)$.

We are done if the full reduction $s \to_{\mathcal{R}}^{*} \ell'\delta' \to_{\mathcal{R}} r'\delta' \to_{\mathcal{R}}^{*} \ell\delta$ is ℓ-formative; this is easy with induction on the number of topmost steps in the second part. □

Lemma 13 lays the foundation for all theorems in this paper. To start:

Theorem 14. *$(\Sigma, \mathcal{R})$ is non-terminating if and only if there is an infinite minimal formative $(\mathsf{DP}(\mathcal{R}), FR(\mathsf{DP}(\mathcal{R}), \mathcal{R}))$-chain. Here, a chain $[(\ell_i \to r_i, \gamma_i) \mid i \in \mathbb{N}]$ is* formative *if always $r_i\gamma_i \to_{FR(\ell_{i+1},\mathcal{R})}^{*} \ell_{i+1}\gamma_{i+1}$ by a formative ℓ_{i+1}-reduction.*

Proof Sketch: Construct an infinite $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$-chain following the usual proof, but when choosing γ_{i+1}, use Lemma 13 to guarantee that $r_i\gamma_i \to_{FR(\ell_{i+1},\mathcal{R})}^{*} \ell_{i+1}\gamma_{i+1}$ by a formative ℓ_{i+1}-reduction. □

Note that this theorem extends the standard dependency pairs result (Theorem 3) by limiting interest to chains with formative reductions.

Example 15. The system from Example 1 is terminating iff there is no infinite minimal formative $(\mathcal{P}, Q)$-chain, where $\mathcal{P} = \mathsf{DP}(\mathcal{R})$ from Example 2 and $Q = \{1, 2, 3, 5, 6, 7, 8, 10, 11\}$. Rules 4 and 9 have right-hand sides headed by symbols `Err` and `Return` which do not occur in the left-hand sides of DP or its formative rules.

4 Formative Rules in the Dependency Pair Framework

Theorem 14 provides a basis for using DPs with formative rules to prove termination: instead of proving that there is no infinite minimal $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$-chain, it suffices if there is no infinite minimal formative $(\mathsf{DP}(\mathcal{R}), FR(\mathsf{DP}(\mathcal{R}), \mathcal{R}))$-chain. So in the DP framework, we can start with the set $\{(\mathsf{DP}(\mathcal{R}), FR(\mathsf{DP}(\mathcal{R}), \mathcal{R}), \mathsf{m})\}$ instead of $\{(\mathsf{DP}(\mathcal{R}), \mathcal{R}, \mathsf{m})\}$, as we did in Example 15. We thus obtain a similar improvement to Dershowitz' refinement [3] in that it yields a smaller *initial* DP problem: by [3], we can reduce the initial set $\mathsf{DP}(\mathcal{R})$; by Theorem 14 we can reduce the initial set $\mathcal{R}$. However, there (currently) is no way to keep track of the information that we only need to consider formative chains. Despite this, we can define several processors. All of them are based on this consequence of Lemma 13:

Lemma 16. *If there is a $(\mathcal{P},\mathcal{R})$-chain $[(\ell_i \to r_i, \gamma_i) \mid i \in \mathbb{N}]$, then there are δ_i for $i \in \mathbb{N}$ such that $[(\ell_i \to r_i, \delta_i) \mid i \in \mathbb{N}]$ is a formative $(\mathcal{P}, \mathit{FR}(\mathcal{P},\mathcal{R}))$-chain.*

Proof. Given $[(\ell_i \to r_i, \gamma_i) \mid i \in \mathbb{N}]$ we construct the formative chain as follows. Let $\delta_1 := \gamma_1$. For given i, suppose δ_i is a substitution such that $\delta_i \to_{\mathcal{R}}^* \gamma_i$, so still $r_i\delta_i \to_{\mathcal{R}}^* \ell_{i+1}\gamma_{i+1}$. Use Lemma 13 to find δ_{i+1} such that $r_i\delta_i \to_{\mathit{FR}(\ell_{i+1},\mathcal{R})}^* \ell_{i+1}\delta_{i+1}$ by a formative ℓ_{i+1}-reduction, and moreover $\delta_{i+1} \to_{\mathcal{R}}^* \gamma_{i+1}$. □

This lemma for instance allows us to remove all non-formative rules from a DP problem. To this end, we use the following processor:

Theorem 17. *The DP processor which maps a DP problem $(\mathcal{P},\mathcal{R},f)$ to the set $\{(\mathcal{P}, \mathit{FR}(\mathcal{P},\mathcal{R}), \mathsf{a})\}$ is sound.*

Proof Sketch: This follows immediately from Lemma 16. □

Example 18. Let $Q = \mathit{FR}_{\mathsf{base}}(\mathsf{DP}(\mathcal{R}),\mathcal{R})$ from Example 15, and let $\mathcal{P} = \{\mathtt{Big}^\sharp(x, \mathtt{Cons}(y,z)) \to \mathtt{Big}^\sharp(\mathtt{Ack}(x,y), \mathtt{Upd}(z))\}$ as in Example 8. If, during a termination proof with dependency pairs, we encounter a DP problem $(\mathcal{P}, Q, \mathsf{m})$, we can soundly replace it by $(\mathcal{P}, T, \mathsf{a})$, where $T = \mathit{FR}_{\mathsf{base}}(\mathcal{P}, Q) = \{8\}$.

Thus, we can (permanently) remove all non-formative rules from a dependency pair problem. This processor has a clear downside, however: given a problem $(\mathcal{P},\mathcal{R},\mathsf{m})$, we lose minimality. This m flag is very convenient to have, as several processors require it (such as reduction pairs with usable rules from Theorem 6).

Could we preserve minimality? Unfortunately, the answer is no. By modifying a chain to use formative reductions, we may lose the property that each $r_i\gamma_i$ is terminating. This happens for instance for $(\mathcal{P},\mathcal{R},\mathsf{m})$, where $\mathcal{P} = \{\mathtt{g}^\sharp(x) \to \mathtt{h}^\sharp(\mathtt{f}(x)),\ \mathtt{h}^\sharp(\mathtt{c}) \to \mathtt{g}^\sharp(\mathtt{a})\}$ and $\mathcal{R} = \{\mathtt{a} \to \mathtt{b},\ \mathtt{f}(x) \to \mathtt{c},\ \mathtt{f}(\mathtt{a}) \to \mathtt{f}(\mathtt{a})\}$. Here, $\mathit{FR}_{\mathsf{base}}(\mathcal{P},\mathcal{R}) = \{\mathtt{f}(x) \to \mathtt{c},\ \mathtt{f}(\mathtt{a}) \to \mathtt{f}(\mathtt{a})\}$. While there is an infinite minimal $(\mathcal{P},\mathcal{R})$-chain, the only infinite $(\mathcal{P}, \mathit{FR}_{\mathsf{base}}(\mathcal{P},\mathcal{R}))$-chain is non-minimal.

Fortunately, there *is* an easy way to use formative rules without losing any information: by using them in a reduction pair, as we typically do for usable rules. In fact, although usable and formative rules seem to be opposites, there is no reason why we should use either one or the other; we can combine them. Considering also argument filterings, we find the following extension of Theorem 6.

Theorem 19. *Let $(\succsim, \succ)$ be a reduction pair and π an argument filtering. The processor which maps $(\mathcal{P},\mathcal{R},f)$ to the following result is sound:*

- *$\{(\mathcal{P} \setminus \mathcal{P}^\succ, \mathcal{R}, f)\}$ if:*
  - *$\overline{\pi}(\ell) \succ \overline{\pi}(r)$ for $\ell \to r \in \mathcal{P}^\succ$ and $\overline{\pi}(\ell) \succsim \overline{\pi}(r)$ for $\ell \to r \in \mathcal{P} \setminus \mathcal{P}^\succ$;*
  - *$u \succsim v$ for $u \to v \in \mathit{FR}(\overline{\pi}(\mathcal{P}), \overline{\pi}(U))$,*
    where $U = \mathcal{R}$ if $f = \mathsf{a}$ and $U = \mathit{UR}(\mathcal{P},\mathcal{R},\pi) \cup \mathcal{C}_\epsilon$ if $f = \mathsf{m}$;
- *$\{(\mathcal{P},\mathcal{R},f)\}$ otherwise.*

Proof Sketch: Given an infinite $(\mathcal{P}, \mathcal{R})$-chain, we use argument filterings and maybe usable rules to obtain a $(\overline{\pi}(\mathcal{P}), \overline{\pi}(U))$-chain which uses the same dependency pairs infinitely often (as in [9]); using Lemma 16 we turn this chain formative. □

Note that we use the argument filtering here in a slightly different way than for usable rules: rather than including π in the definition of *FR* and requiring that $\overline{\pi}(\ell) \succsim \overline{\pi}(r)$ for $\ell \to r \in \mathit{FR}(\mathcal{P}, \mathcal{R}, \pi)$, we simply use $\mathit{FR}(\overline{\pi}(\mathcal{P}), \overline{\pi}(\mathcal{R}))$. For space reasons, we give additional semantic and syntactic definitions of formative rules with respect to an argument filtering in the technical report [5, Appendix C].

Example 20. To handle $(\mathcal{P}, Q, \mathsf{m})$ from Example 18, we can alternatively use a reduction pair. Using the trivial argument filtering, with a polynomial interpretation with $\mathtt{Big}^\sharp(x, y) = x + y$, $\mathtt{Ack}(x, y) = 0$, $\mathtt{Upd}(x) = x$ and $\mathtt{Cons}(x, y) = y + 1$, all constraints are oriented, and we may remove the only element of $\mathcal{P}$.

Note that we *could* have handled this example without using formative rules; `Ack` and `Rnd` can be oriented with an extension of $\succsim$, or we might use an argument filtering with $\pi(\mathtt{Big}^\sharp) = \{2\}$. Both objections could be cancelled by adding extra rules, but we kept the example short, as it suffices to illustrate the method.

Discussion It is worth noting the parallels between formative and usable rules. To start, their definitions are very similar; although we did not present the semantic definition of usable rules from [16] (which is only used for *innermost* termination), the syntactic definitions are almost symmetric. Also the usage corresponds: in both cases, we lose minimality when using the direct rule removing processor, but can safely use the restriction in a reduction pair (with argument filterings).

There are also differences, however. The transformations used to turn a chain usable or formative are very different, with the usable rules transformation (which we did not discuss) encoding subterms whose root is not usable, while the formative rules transformation is simply a matter of postponing reduction steps.

Due to this difference, usable rules are useful only for a finitely branching system (which is standard, as all finite MTRSs are finitely branching); formative rules are useful mostly for left-linear systems (also usual, especially in MTRSs originating from functional programming, but typically seen as a larger restriction). Usable rules introduce the extra $\mathcal{C}_\epsilon$ rules, while formative rules are all included in the original rules. But for formative rules, even definitions extending $\mathit{FR}_{\mathsf{base}}$, necessarily all collapsing rules are included, which has no parallel in usable rules; the parallel of collapsing rules would be rules $x \to r$, which are not permitted.

To use formative rules without losing minimality information, an alternative to Theorem 17 allows us to permanently delete rules. The trick is to add a new component to DP problems, as for higher-order rewriting in [11, Ch. 7]. A DP problem becomes a tuple $(\mathcal{P}, \mathcal{R}, f_1, f_2)$, with $f_1 \in \{\mathsf{m}, \mathsf{a}\}$ and $f_2 \in \{\mathsf{form}, \mathsf{arbitrary}\}$, and is finite if there is no infinite $(\mathcal{P}, \mathcal{R})$-chain which is minimal if $f_1 = \mathsf{m}$, and formative if $f_2 = \mathsf{form}$. By Theorem 14, $\mathcal{R}$ is terminating iff $(\mathsf{DP}(\mathcal{R}), \mathcal{R}, \mathsf{m}, \mathsf{form})$ is finite.

Theorem 21. *In the extended DP framework, the processor which maps $(\mathcal{P}, \mathcal{R}, f_1, f_2)$ to $\{(\mathcal{P}, \mathit{FR}(\mathcal{P}, \mathcal{R}), f_1, f_2)\}$ if $f_2 = \mathsf{form}$ and $\{(\mathcal{P}, \mathcal{R}, f_1, f_2)\}$ otherwise, is sound.*

Proof: This follows immediately from Lemma 12. □

The downside of changing the DP framework in this way is that we have to revisit all existing DP processors to see how they interact with the formative flag. In many cases, we can simply pass the flag on unmodified (i.e. if $\mathit{proc}((\mathcal{P},\mathcal{R},f_1)) = A$, then $\mathit{proc}'((\mathcal{P},\mathcal{R},f_1,f_2)) = \{(\mathcal{P}',\mathcal{R}',f_1',f_2) \mid (\mathcal{P}',\mathcal{R}',f_1') \in A\}$). This is for example the case for processors with reduction pairs (like the one in Theorem 19), the dependency graph and the subterm criterion. Other processors would have to be checked individually, or reset the flag to $\mathsf{arbitrary}$ by default.

Given how long the dependency pair framework has existed (and how many processors have been defined, see e.g. [16]), and that the formative flag clashes with the component for innermost rewriting (see Section 6), it is unlikely that many tool programmers will make the effort for a single rule-removing processor.

5 Handling the Collapsing Rules Problem

A great weakness of the formative rules method is the matter of collapsing rules. Whenever the left-hand side of a dependency pair or formative rule has a symbol $f : [\boldsymbol{\iota}] \Rightarrow \kappa$, all collapsing rules of sort κ are formative. And then all *their* formative rules are also formative. Thus, this often leads to the inclusion of all rules of a given sort. In particular for systems with only one sort (such as all first-order benchmarks in the Termination Problems Data Base), this is problematic.

For this reason, we will consider a new notion, building on the idea of formative rules and reductions. This notion is based on the observation that it might suffice to include *composite rules* rather than the formative rules of all collapsing rules. To illustrate the idea, assume given a uni-sorted system with rules $\mathtt{a} \to \mathtt{f}(\mathtt{b})$ and $\mathtt{f}(x) \to x$. $\mathit{FR}_{\mathsf{base}}(\mathtt{c})$ includes $\mathtt{f}(x) \to x$, so also $\mathtt{a} \to \mathtt{f}(\mathtt{b})$. But a term $\mathtt{f}(\mathtt{b})$ does not reduce to $\mathtt{c}$. So intuitively, we should not really need to include the first rule.

Instead of including the formative rules of all collapsing rules, we might imagine a system where we *combine* rules with collapsing rules that could follow them. In the example above, this gives $\mathcal{R} = \{\mathtt{a} \to \mathtt{f}(\mathtt{b}),\ \mathtt{a} \to \mathtt{b},\ \mathtt{f}(x) \to x\}$. Now we might consider an alternative definition of formative rules, where we still need to include the collapsing rule $\mathtt{f}(x) \to x$, but no longer need to have $\mathtt{a} \to \mathtt{f}(\mathtt{b})$.

To make this idea formal, we first consider how rules can be combined. In the following, we consider systems with *only one sort*; this is needed for the definition to be well-defined, but can always be achieved by replacing all sorts by $\mathtt{o}$.

Definition 22 (Combining Rules). *Given an MTRS $(\Sigma,\mathcal{R})$, let $A := \{f(\boldsymbol{x}) \to x_i \mid f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa \in \Sigma \wedge 1 \leq i \leq n\}$ and $B := \{\ell \to p \mid \ell \to r \in \mathcal{R} \wedge r \trianglerighteq p\}$. Let $X \subseteq A \cup B$ be the smallest set such that $\mathcal{R} \subseteq X$ and for all $\ell \to r \in X$:*

a. if r is a variable, $\ell \trianglerighteq f(l_1,\ldots,l_n)$ and $l_i \trianglerighteq r$, then $f(x_1,\ldots,x_n) \to x_i \in X$;
b. if $r = f(r_1,\ldots,r_n)$ and $f(x_1,\ldots,x_n) \to x_i \in X$, then $\ell \to r_i \in X$.

Let $\mathit{Cl} := A \cap X$ and $\mathit{NC} = \{\ell \to r \in X \mid r \text{ not a variable}\}$. Let $A_{\mathcal{R}} := \mathit{Cl} \cup \mathit{NC}$.

It is easy to see that $\to_{\mathcal{R}}^*$ is included in $\to_{A_{\mathcal{R}}}^*$: all non-collapsing rules of $\mathcal{R}$ are in NC, and all collapsing rules are obtained as a concatenation of steps in Cl.

Example 23. Consider an unsorted version of Example 1. Then for $(\mathcal{P},Q)$ as in Example 18, we have $U := \mathit{UR}(\mathcal{P},Q) = \{1,2,3,5,8,10,11\}$. Unfortunately, only (3) is not formative, as the two $\mathtt{Rnd}$ rules cause inclusion of all rules in $\mathit{FR}_{\mathsf{base}}(\mathtt{S}(x), U)$. Let us instead calculate X, which we do as an iterative procedure starting from $\mathcal{R}$. In the following, $C \Rightarrow D_1, \dots, D_n$ should be read as: "by requirement a, rule C enforces inclusion of each D_i", and $C, D \Rightarrow E$ similarly refers to requirement b.

$2, 1 \Rightarrow 12$	$5, 13 \Rightarrow 14$	$10, 15 \Rightarrow 16$	$16, 13 \Rightarrow 18$	$17, 15 \Rightarrow 19$
$12 \Rightarrow 1, 13$	$14 \Rightarrow 15$	$11, 15 \Rightarrow 17$	$18 \Rightarrow 15, 13$	$19 \Rightarrow 15, 13$

12. $\mathtt{Rnd}(\mathtt{S}(x)) \to x$	15. $\mathtt{Ack}(x, y) \to y$	18. $\mathtt{Ack}(\mathtt{S}(x), y) \to y$
13. $\mathtt{S}(x) \to x$	16. $\mathtt{Ack}(\mathtt{S}(x), y) \to \mathtt{S}(y)$	19. $\mathtt{Ack}(\mathtt{S}(x), \mathtt{S}(y)) \to y$
14. $\mathtt{Ack}(\mathtt{0}, y) \to y$	17. $\mathtt{Ack}(\mathtt{S}(x), \mathtt{S}(y)) \to \mathtt{Ack}(\mathtt{S}(x), y)$	

Now $\mathit{Cl} = \{1, 13, 15\}$ and $\mathit{NC} = \{2, 3, 5, 8, 10, 11, 16, 17\}$, and $A_U = \mathit{Cl} \cup \mathit{NC}$.

Although combining a system $\mathcal{R}$ into $A_{\mathcal{R}}$ may create significantly more rules, the result is not necessarily harder to handle. For many standard reduction pairs, like RPO or linear polynomials over $\mathbb{N}$, we have: if $s \succsim x$ where $x \in \mathit{Var}(s)$ occurs exactly once, then $f(\dots, t, \dots) \succsim t$ for any t with $s \trianglerighteq t \trianglerighteq x$. For such a reduction pair, $A_{\mathcal{R}}$ can be oriented whenever $\mathcal{R}$ can be (if $\mathcal{R}$ is left-linear).

$A_{\mathcal{R}}$ has the advantage that we never need to follow a non-collapsing rule $l \to f(\boldsymbol{r})$ by a collapsing step. This is essential to use the following definition:

Definition 24. *Let A be a set of rules. The* split-formative rules *of a term t are defined as the smallest set $\mathit{SR}(t, A) \subseteq A$ such that:*
- *if t is not linear, then $\mathit{SR}(t, A) = A$;*
- $\boxed{\textit{all collapsing rules in } A \textit{ are included in } \mathit{SR}(t, A);}$
- *if $t = f(t_1, \dots, t_n)$, then $\mathit{SR}(t_i, A) \subseteq \mathit{SR}(t, A)$;*
- *if $t = f(t_1, \dots, t_n)$, then $\{\ell \to r \in A \mid r \text{ has the form } f(\dots)\} \subseteq \mathit{SR}(t, A)$;*
- *if $\ell \to r \in \mathit{SR}(t, A)$ $\boxed{\textit{and } r \textit{ is not a variable}}$, then $\mathit{SR}(\ell, A) \subseteq \mathit{SR}(t, A)$.*

For a set of rules $\mathcal{P}$, we define $\mathit{SR}(\mathcal{P}, A) = \bigcup_{s \to t \in \mathcal{P}} \mathit{SR}(s, A)$.

Definition 24 is an alternative definition of formative rules, where collapsing rules have a smaller effect (differences to Definition 7 are $\boxed{\text{highlighted}}$). *SR* is *not* a formative rules approximation, as shown by the $\mathtt{a}$-formative reduction $\mathtt{f}(\mathtt{a}) \to_{\mathcal{R}} \mathtt{g}(\mathtt{a}) \to_{\mathcal{R}} \mathtt{a}$ with $\mathcal{R} = \{\mathtt{f}(x) \to \mathtt{g}(x),\ \mathtt{g}(x) \to x\}$ but $\mathit{SR}(\mathtt{a}, \mathcal{R}) = \{\mathtt{g}(x) \to x\}$. However, given the relation between $\mathcal{R}$ and $A_{\mathcal{R}}$, we find a similar result to Lemma 12:

Lemma 25. *Let $(\Sigma, \mathcal{R})$ be an MTRS. If $s \to_{\mathcal{R}}^* \ell\gamma$ by a formative ℓ-reduction, then $s \to_{\mathit{SR}(\ell, A_{\mathcal{R}})}^* \ell\gamma$ by a formative ℓ-reduction.*

Unlike Lemma 12, the altered reduction might be different. We also do not have that $\mathit{SR}(\mathcal{P}, A_{\mathcal{R}}) \subseteq \mathcal{R}$. Nevertheless, by this lemma we can use split-formative rules in reduction pair processors with formative rules, such as Theorem 19.

Proof Sketch: The original reduction $s \to_{\mathcal{R}}^* \ell\gamma$ gives rise to a formative reduction over $A_{\mathcal{R}}$, simply replacing collapsing steps by a sequence of rules in Cl. So, we assume given a formative ℓ-reduction over $A_{\mathcal{R}}$, and prove with induction first on the number of non-collapsing steps in the reduction, second on the length of the reduction, third on the size of s, that $s \to_{\mathit{SR}(\ell, A_{\mathcal{R}})}^* \ell\gamma$ by a formative ℓ-reduction.

This is mostly easy with the induction hypotheses; note that if a root-rule in NC is followed by a rule in Cl, there can be no internal $\to_{\mathcal{R}}^*$ reduction in between (as this would not be a formative reduction); combining a rule in NC with a rule in Cl gives either a rule in NC (and a continuation with the second induction hypothesis) or a sequence of rules in Cl (and the first induction hypothesis). $\square$

Note that this method unfortunately does not transpose directly to the higher-order setting, where collapsing rules may have more complex forms. We also had to give up sort differentiation, as otherwise we might not be able to flatten a rule $f(g(x)) \to x$ into $f(x) \to x,\ g(x) \to x$. This is not such a great problem, as reduction pairs typically do not care about sorts, and we circumvented the main reason why sorts are important for formative rules. We have the following result:

Theorem 26. *Let $(\succsim, \succ)$ be a reduction pair and π an argument filtering. The processor which maps a DP problem $(\mathcal{P}, \mathcal{R}, f)$ to the following result is sound:*

- *$\{(\mathcal{P} \setminus \mathcal{P}^{\succ}, \mathcal{R}, f)\}$ if:*
  - *$\overline{\pi}(\ell) \succ \overline{\pi}(r)$ for $\ell \to r \in \mathcal{P}^{\succ}$ and $\overline{\pi}(\ell) \succsim \overline{\pi}(r)$ for $\ell \to r \in \mathcal{P} \setminus \mathcal{P}^{\succ}$;*
  - *$u \succsim v$ for $u \to v \in SR(\overline{\pi}(\mathcal{P}), A_{\overline{\pi}(U)})$, and $Var(t) \subseteq Var(s)$ for $s \to t \in \overline{\pi}(U)$, where $U = \mathcal{R}$ if $f = \mathsf{a}$ and $U = UR(\mathcal{P}, \mathcal{R}, \pi) \cup \mathcal{C}_{\epsilon}$ if $f = \mathsf{m}$;*
- *$\{(\mathcal{P}, \mathcal{R}, f)\}$ otherwise.*

Proof Sketch: Like Theorem 19, but using Lemma 25 to alter the created formative $(\overline{\pi}(\mathcal{P}), \overline{\pi}(U))$-chain to a split-formative $(\overline{\pi}(\mathcal{P}), SR(\overline{\pi}(\mathcal{P}), A_{\overline{\pi}(U)}))$-chain. $\square$

Example 27. Following Example 23, $SR(\mathtt{Big}^{\sharp}(x, \mathtt{Cons}(y, z)) \to \mathtt{Big}^{\sharp}(\mathtt{Ack}(x, y), \mathtt{Upd}(z)), A_U) = Cl \cup \{8\}$, and Theorem 26 gives an easily orientable problem.

6 Formative Rules for Innermost Termination

So far, we have considered only *full termination.* A very common related query is *innermost termination*; that is, termination of $\to_{\mathcal{R}}^{\mathsf{in}}$, defined by:

- $f(\boldsymbol{l})\gamma \to_{\mathcal{R}}^{\mathsf{in}} r\gamma$ if $f(\boldsymbol{l}) \to r \in \mathcal{R}$, γ a substitution and all $l_i\gamma$ in normal form;
- $f(s_1, \dots, s_i, \dots, s_n) \to_{\mathcal{R}}^{\mathsf{in}} f(s_1, \dots, s_i', \dots, s_n)$ if $s_i \to_{\mathcal{R}}^{\mathsf{in}} s_i'$.

The innermost reduction relation is often used in for instance program analysis.

An innermost strategy can be included in the dependency pair framework by adding the $\mathsf{innermost}$ flag [9] to DP problems (or, more generally, a component $\mathcal{Q}$ [7] which indicates that when reducing any term with $\to_{\mathcal{P}}$ or $\to_{\mathcal{R}}$, its strict subterms must be normal with respect to $\mathcal{Q}$). Usable rules are more viable for innermost than normal termination: we do not need minimality, the $\mathcal{C}_{\epsilon}$ rules do not need to be handled by the reduction pair, and we can define a sound processor that maps $(\mathcal{P}, \mathcal{R}, f, \mathsf{innermost})$ to $\{(\mathcal{P}, UR(\mathcal{P}, \mathcal{R}), f, \mathsf{innermost})\}$.

This is not the case for formative rules. Innermost reductions eagerly evaluate arguments, yet formative reductions postpone evaluations as long as possible. In a way, these are exact opposites. Thus, it should not be surprising that formative rules are *weaker* for innermost termination than for full termination. Theorem 14 has no counterpart for $\to_{\mathcal{R}}^{\mathsf{in}}$; for innermost termination we must start the DP framework with $(\mathcal{P}, \mathcal{R}, \mathsf{m}, \mathsf{innermost})$, not with $(\mathcal{P}, FR(\mathcal{P}, \mathcal{R}), \mathsf{m}, \mathsf{innermost})$. Theorem 17 is only sound if the innermost flag is removed: $(\mathcal{P}, \mathcal{R}, f, \mathsf{innermost})$ is mapped to $\{(\mathcal{P}, FR(\mathcal{P}, \mathcal{R}), \mathsf{a}, \mathsf{arbitrary})\}$. Still, we *can* safely use formative rules with reduction pairs. For example, we obtain this variation of Theorem 19:

Theorem 28. *Let $(\succsim, \succ)$ be a reduction pair and π an argument filtering. The processor which maps a DP problem $(\mathcal{P}, \mathcal{R}, f_1, f_2)$ to the following result is sound:*

- $\{(\mathcal{P} \setminus \mathcal{P}^{\succ}, \mathcal{R}, f_1, f_2)\}$ *if:*
  - $\overline{\pi}(\ell) \succ \overline{\pi}(r)$ *for* $\ell \to r \in \mathcal{P}^{\succ}$ *and* $\overline{\pi}(\ell) \succsim \overline{\pi}(r)$ *for* $\ell \to r \in \mathcal{P} \setminus \mathcal{P}^{\succ}$*;*
  - $u \succsim v$ *for* $u \to v \in FR(\overline{\pi}(\mathcal{P}), \overline{\pi}(U))$*, where* U *is:* $UR(\mathcal{P}, \mathcal{R}, \pi)$ *if* $f_2 = \mathsf{innermost}$*; otherwise* $UR(\mathcal{P}, \mathcal{R}, \pi) \cup \mathcal{C}_\epsilon$ *if* $f_1 = \mathsf{m}$*; otherwise* $\mathcal{R}$*.*
- $\{(\mathcal{P}, \mathcal{R}, f_1, f_2)\}$ *otherwise.*

Proof Sketch: The proof of Theorem 19 still applies; we just ignore that the given chain might be innermost (aside from getting more convenient usable rules). □

Theorem 26 extends to innermost termination in a similar way.

Conveniently, innermost termination is *persistent* [4], so modifying Σ does not alter innermost termination behaviour, as long as all rules stay well-sorted. In practice, we could infer a typing with as many different sorts as possible, and get stronger formative-rules-with-reduction-pair processors. With the *innermost switch processor* [16, Thm. 3.14], which in cases can set the innermost flag on a DP problem, we could also often use this trick even for proving full termination.

In Section 4, we used the extra flag f_2 as the formative flag. It is not contradictory to use f_2 in both ways, allowing $f_2 \in \{\mathsf{arbitrary}, \mathsf{form}, \mathsf{innermost}\}$, since it is very unlikely for a $(\mathcal{P}, \mathcal{R})$-chain to be both formative and innermost at once! When using both extensions of the DP framework together, termination provers (human or computer) will, however, sometimes have to make a choice which flag to add.

7 Implementation and Experiments

We have performed a preliminary implementation of formative rules in the termination tool AProVE [6]. Our automation borrows from the usable rules of [8] (see [5, Appendices B+D]) and uses a constraint encoding [2] for a combined search for argument filterings and corresponding formative rules. While we did not find any termination proofs for examples from the TPDB where none were known before, our experiments show that formative rules do improve the power of reduction pairs for widely used term orders (e.g., polynomial orders [14]). For more information, see also: `http://aprove.informatik.rwth-aachen.de/eval/Formative`

For instance, we experimented with a configuration where we applied dependency pairs, and then alternatingly dependency graph decomposition and reduction pairs with linear polynomials and coefficients ≤ 3. On the TRS Standard category of the TPDB (v8.0.7) with 1493 examples, this configuration (without formative rules, but with usable rules w.r.t. an argument filter) shows termination of 579 examples within a timeout of 60 seconds (on an Intel Xeon 5140 at 2.33 GHz). With additional formative rules, our implementation of Theorem 19 proved termination of 6 additional TRSs. (We did, however, lose 4 examples to timeouts, which we believe are due in part to the currently unoptimised implementation.)

The split-formative rules from Theorem 26 are not a subset of $\mathcal{R}$, in contrast to the usable rules. Thus, it is *a priori* not clear how to combine their encodings w.r.t. an argument filtering, and we conducted experiments using only the standard usable rules. Without formative rules, 532 examples are proved terminating. In contrast, adding either the formative rules of Theorem 19 or the split-formative rules of Theorem 26 we solved 6 additional examples each (where Theorem 19 and Theorem 26 each had 1 example the other could not solve), losing 1 to timeouts.

Finally, we experimented with the improved dependency pair transformation based on Theorem 14, which drops non-formative rules from $\mathcal{R}$. We applied DPs as the first technique on the 1403 TRSs from TRS Standard with at least one DP. This reduced the number of rules in the initial DP problem for 618 of these TRSs, without any search problems and without sacrificing minimality.

Thus, our current impression is that while formative rules are not the next "killer technique", they nonetheless provide additional power to widely-used orders in an elegant way and reduce the number of term constraints to be solved in a termination proof. The examples from the TPDB are all untyped, and we believe that formative rules may have a greater impact in a typed first-order setting.

8 Conclusions

In this paper, we have simplified the notion of formative rules from [13] to the first-order setting, and integrated it in the dependency pair framework. We did so by means of *formative reductions*, which allows us to obtain a semantic definition of formative rules (more extensive syntactic definitions are discussed in [5]).

We have defined three processors to use formative rules in the standard dependency pair framework for full termination: one is a processor to permanently remove rules, the other two combine formative rules with a reduction pair.

We also discussed how to strengthen the method by adding a new flag to the framework – although doing so might require too many changes to existing processors and strategies to be considered worthwhile – and how we can still use the technique in the innermost case, and even profit from the innermost setting.

Related Work In the first-order DP framework two processors stand out as relevant to formative rules. The first is, of course, usable rules; see Section 4 for a detailed discussion. The second is the *dependency graph*, which determines whether any two dependency pairs can follow each other in a $(\mathcal{P}, \mathcal{R})$-chain, and uses this information to eliminate elements of $\mathcal{P}$, or to split $\mathcal{P}$ in multiple parts.

In state-of-the-art implementations of the dependency graph (see e.g. [16]), both left- and right-hand side of dependency pairs are considered to see whether a pair can be preceded or followed by another pair. Therefore it seems quite surprising that the same mirroring was not previously tried for usable rules.

Formative rules *have* been previously defined, for higher-order term rewriting, in [13], which introduces a limited DP framework, with formative rules (but not formative reductions) included in the definition of a chain: we simply impose the restriction that always $r_i\gamma_i \rightarrow^*_{FR(\mathcal{P},\mathcal{R})} \ell_{i+1}\gamma_{i+1}$. This gives a reduction pair processor which considers only formative rules, although it cannot be combined with usable rules and argument filterings. The authors do not yet consider rule removing processors, but if they did, Theorem 21 would also go through.

In the second author's PhD thesis [11], a more complete higher-order DP framework is considered. Here, we do see formative reductions, and a variation of Lemma 13 which, however, requires that s is terminating: the proof style used here does not go through there due to β-reduction. Consequently, Lemma 16 does not go through in the higher-order setting, and there is no counterpart to Theorems 17 or 19. We *do*, however, have Theorem 21. Furthermore, the results of Section 5 are entirely new to this paper, and do not apply in the higher-order setting, where rules might also have a form $l \to x \cdot s_1 \cdots s_n$ (with x a variable).

Future Work In the future, it would be interesting to look back at higher-order rewriting, and see whether we can obtain some form of Lemma 16 after all. Alternatively, we might be able to use the specific form of formative chains to obtain formative (and usable) rules w.r.t. an argument filtering.

In the first-order setting, we might turn our attention to non-left-linear rules. Here, we could think for instance of renaming apart some of these variables; a rule $f(x,x) \to g(x,x)$ could become any of $f(x,y) \to g(x,y),\ f(x,y) \to g(y,x),\ \ldots$

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
2. M. Codish, J. Giesl, P. Schneider-Kamp, and R. Thiemann. SAT solving for termination proofs with recursive path orders and dependency pairs. *Journal of Automated Reasoning*, 49(1):53–93, 2012.
3. N. Dershowitz. Termination by abstraction. In *Proc. ICLP '04*, 2004.
4. C. Fuhs, J. Giesl, M. Parting, P. Schneider-Kamp, and S. Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 47(2):133–160, 2011.
5. C. Fuhs and C. Kop. First-order formative rules. Technical Report `arXiv:1404.7695 [cs.LO]`, 2014. `http://arxiv.org/abs/1404.7695`.
6. J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Proving termination of programs automatically with AProVE. In *Proc. IJCAR '14*, 2014. To appear.
7. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR '04*. 2005.
8. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. FroCoS '05*, 2005.
9. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
10. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
11. C. Kop. *Higher Order Termination.* PhD thesis, Vrije Univ. Amsterdam, 2012.
12. C. Kop and F. van Raamsdonk. Higher order dependency pairs for algebraic functional systems. In *Proc. RTA '11*, 2011.
13. C. Kop and F. van Raamsdonk. Dynamic dependency pairs for algebraic functional systems. *Logical Methods in Computer Science*, 8(2), 2012.
14. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, 1979.
15. V. Tannen and G.H. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83(1):3–28, 1991.
16. R. Thiemann. *The DP framework for proving termination of term rewriting.* PhD thesis, RWTH Aachen, 2007.