Transformations of higher-order term rewrite systems

Cynthia Kop
Department of Theoretical Computer Science
Vrije Universiteit, Amsterdam
kop@few.vu.nl

We study the common ground and differences of different frameworks for higher-order rewriting from the viewpoint of termination by encompassing them in a generalised framework.

1 Introduction

In the past decades a lot of research has been done on termination of term rewrite systems. However, the specialised area of higher order rewriting is sadly lagging behind. There are many reasons for this. Primarily, the topic is relatively difficult, mostly due to the presence of the beta rule. Applications are also not in as much abudance as with first order rewriting. A third limiting factor is the lack of a set standard. There are several important formalisms, each dealing with the higher order aspect in a different way, plus many variations and restrictions. Because of the differences in what is and is not allowed, results in one formalism do not trivially, or not at all, carry over to another. As such it is difficult to reuse results in a slightly different context, which necessitates a lot of double work.

In this paper we present work in progress investigating the common ground and differences of various formalisms from the viewpoint of termination. We introduce yet another formalism, but show how the usual styles of rewriting can be represented in it. We then look into properties within the general formalism and show which ones can always be obtained by transforming the system and which cannot. Finally, to demonstrate that the system is not too general to work with, we extend the Computability Path Ordering [2] to our formalism.

2 The formalism

In this section we introduce a formalism of higher-order rewriting, called *Higher Order Decidable Rewrite Systems*.

types We assume a set of *base sorts* $\mathscr{B}$ and a set of *type variables* $\mathscr{A}$. Each (base) sort b has a fixed arity, notation: $b : n$; at least one sort has arity 0. A *polymorphic type* is an expression over $\mathscr{B}$ and $\mathscr{A}$ built according to the following grammar:

$$\mathscr{T} = \alpha | b(\mathscr{T}^n) | \mathscr{T} \to \mathscr{T} \quad (\alpha \in \mathscr{A},\ b : n \in \mathscr{B})$$

Submitted to:

A *monomorphic* type does not contain type variables. A type is *composed* if it is headed by the arrow symbol. A type $b()$ with $b : 0 \in \mathscr{B}$ is denoted as just b. The $\to$ associates to the right. We say $\sigma \geq \tau$ if τ can be obtained from σ by substituting types for type variables. For example, $\alpha \geq \alpha \geq \alpha \to \beta \geq \mathbb{N} \to \mathbb{N}$, but not $\alpha \to \alpha \geq \mathbb{N} \to \mathbb{R}$.

(meta-)terms A *metaterm* is a typed expression over a set $\mathscr{F}$ of typed constants (also known as *function symbols*) $f : \sigma$ and infinite set $\mathscr{V}$ of variables. We define the set of metaterms $\mathbb{M}$ together with a set $\mathscr{V}ar$ of free variables for each metaterm recursively with the following rules:

`(var)`	$x_\tau : \tau \in \mathbb{M}$	if	$x \in \mathscr{V}$	$\mathscr{V}ar(x_\tau) = \{x_\tau\}$
`(fun)`	$f_\tau : \tau \in \mathbb{M}$	if	$f : \sigma \in \mathscr{F}$ and $\sigma \geq \tau$	$\mathscr{V}ar(f_\tau) = \emptyset$
`(abs)`	$\lambda x_\sigma.s : \sigma \to \tau \in \mathbb{M}$	if	$x \in \mathscr{V}$, $s : \tau \in \mathbb{M}$ and $\mathscr{V}ar(s) \cup \{x_\sigma\}$ UT$(**)$	$\mathscr{V}ar(\lambda x_\sigma.s) = \mathscr{V}ar(s) \setminus \{x_\sigma\}$
`(app)`	$s \cdot t : \tau \in \mathbb{M}$	if	$s : \sigma \to \tau \in \mathbb{M}$, $t : \sigma \in \mathbb{M}$ and $\mathscr{V}ar(s) \cup \mathscr{V}ar(t)$ UT	$\mathscr{V}ar(s \cdot t) = \mathscr{V}ar(s) \cup \mathscr{V}ar(t)$
`(meta)`	$X_\sigma[s_1,\ldots,s_n] : \tau \in \mathbb{M}$	if	$\sigma = \sigma_1 \to \ldots \sigma_n \to \tau$ and $s_1 : \sigma_1, \ldots, s_n : \sigma_n$ and $\{X_\sigma\} \cup \bigcup_i \mathscr{V}ar(s_i)$ UT	$\mathscr{V}ar(X_\sigma[s_1,\ldots,s_n]) = \{X_\sigma\} \cup \bigcup_i \mathscr{V}ar(s_i)$

$(**)$ A set V of typed variables is called UT, *uniquely typed*, if for any $x_\sigma \in V$ there is no $x_\tau \in V$ with $\sigma \neq \tau$.

A metaterm generated without clause `(meta)` is a *term*. We work modulo renaming of variables bound by an abstraction operator (α-conversion). Explicit typing of terms will usually be omitted. The $\cdot$ operator associates to the left, so a metaterm $s \cdot t \cdot r$ should be read as $(s \cdot t) \cdot r$. We will adopt the custom of writing a (meta-)term $s \cdot t_1 \cdots t_n$ in the form $s(t_1,\ldots,t_n)$.

type substitution A type substitution is a mapping $p : \mathscr{A} \to \mathscr{T}$. For any metaterm s let $s'p$ be s with all type variables α replaced by $p(\alpha)$. As an example, $(\mathtt{if}_{\mathtt{bool}\to\alpha\to\alpha\to\alpha}(X_{\mathtt{bool}}, Y_\alpha, 0_\alpha) : \alpha)'\{\alpha \to \mathbb{N}\} = \mathtt{if}_{\mathtt{bool}\to\mathbb{N}\to\mathbb{N}\to\mathbb{N}}(X_{\mathtt{bool}}, Y_\mathbb{N}, 0_\mathbb{N})$.

We say $s \geq t$ if there is a substitution p such that $s = t'p$. Given a typable expression s (that is, a term with some type indicators omitted), it has a *principal term* t, which is $\geq$ any term r obtained from s by filling in type indicators. When a term is displayed with type indicators missing, we always mean its principal term. For example, if $f : \alpha \to \alpha \in \mathscr{F}$, the principal term of f is $f_{\alpha\to\alpha}$, whereas the principal term of $f(0_N)$ is $f_{N\to N}(0_N)$.

term and metaterm substitution A (term) substitution is the homomorphic extension of a typep-reserving mapping $[x_{1,\sigma_1} := s_1, \ldots, x_{n,\sigma_n} := s_n]$ with all s_i terms. Substitutions for meta-applications $X_\sigma[t_1,\ldots,t_m]$ "eat" their arguments. Formally, let γ be the function mapping x_{i,σ_i} to s_i and $\gamma(x_\tau) = x_\tau$ for other typed variables. For any metaterm s, $s\gamma$ is generated by the following rules:

$x_\sigma\gamma = \gamma(x_\sigma)$ for $x \in \mathscr{V}$
$f_\sigma\gamma = f_\sigma$ for $f \in \mathscr{F}$
$(s \cdot t)\gamma = (s\gamma) \cdot (t\gamma)$

$(\lambda x_\sigma.s)\gamma = \lambda x_\sigma.(s\gamma)$ if $x_\sigma \notin \mathrm{dom}(\gamma)$ (we can rename x if necessary)
$x_\sigma[s_1,\ldots,s_n]\gamma = q[y_1 := s_1\gamma,\ldots,y_m := s_m\gamma]\cdot(s_{m+1}\gamma)\cdots(s_m\gamma)$
if $\gamma(x) = \lambda y_1\ldots y_m.q$, $m \leq n$ and $m = n$ or q is not an abstraction.
A metaterm is *standard* if variables occuring at the head of a meta-application are not bound, and all free variables occur at the head of a meta-application.

Examples: $(x(\lambda y.y)[x := \lambda z.z(a)] = (\lambda z.z(a))(\lambda y.y)$ whereas $x[\lambda y.y][x := \lambda z.z(a)] = (\lambda y.y)(a))$. Even an empty substitution has an effect on a proper metaterm: $x[\lambda y.y][] = x(\lambda y.y)$.

β and η $=_\beta$ is the equivalence relation generated by $(\lambda x.s)\cdot t =_\beta s[x := t]$. Every metaterm s is β-equal to a unique β-*normal* term $s\downarrow_\beta$ which has no subterms $(\lambda x.s)\cdot t$. This is a wellknown result, which is easily extended to HODRSs.

$=_\eta$ is the equivalence relation generated by $s = \lambda x.s\cdot x$ if $x \notin \mathscr{V}ar(s)$, and $X[s_1\ldots,s_n] = \lambda x.X[s_1,\ldots,s_n,x]$. A metaterm is in η-normal form if any higher-order subterm is either an abstraction or meta-application, occurs at the head of an application or occurs as a direct argument of a meta-application (for example, $X[s]$ is η-normal if $X : \sigma\to\tau$ with τ not composed, and all direct subterms of s are η-normal). While a term may have more than one η-normal form ($f : o\to o$ has normal forms $\lambda x.f(x)$ and $\lambda x.(\lambda x.f(x))\cdot x$), we define $s\downarrow_\eta$ as its minimal η-normal form.

$=\beta\eta$ is the union of these relations. Each term has a unique $\beta\eta$-normal form.

rules A term rewrite system consists of an alphabet $\mathscr{F}$, a set of *rules* $\mathscr{R}$ and an equivalence relation δ, where δ is one of β, η, $\beta\eta$ or normal equality ε (the α rule is implicit in all). Rules are tuples (l,r) (commonly denoted $l \Rightarrow r$), where l,r are standard metaterms satisfying the following properties: 1) l and r have the same type, 2) all variables and type variables in r also occur in l, 3) if l has a subterm $X[s_1,\ldots,s_n]$ then the s_i are all distinct bound variables (the *parameter restriction*), 4) if the equivalence relation is either β or $\beta\eta$, no subterms $X[s_1,\ldots,s_n]\cdot t_0\cdots t_m$ $(n,m \geq 0)$ occur in l.

$\mathscr{R}$ induces a rewrite relation $\Rightarrow_\mathscr{R}$ over terms in minimal δ-normal form:

`(top)`	$l'p\gamma =_\delta s \Rightarrow_\mathscr{R} t =_\delta r'p\gamma$	if	$l \Rightarrow r \in \mathscr{R}$, p a type substitution and γ a substitution
`(app-l)`	$s\cdot t \Rightarrow_\mathscr{R} s'\cdot t$	if	$s \Rightarrow_\mathscr{R} s'$
`(app-r)`	$s\cdot t \Rightarrow_\mathscr{R} s\cdot t'$	if	$t \Rightarrow_\mathscr{R} t'$
`(abs)`	$\lambda x.s \Rightarrow_\mathscr{R} \lambda x.s'$	if	$s \Rightarrow_\mathscr{R} s'$

The reduction relation is decidable due to the parameter restriction.

3 Pleasant properties and transformations

To prove results about HODRSs it is often convenient to have a bit more to go on than just the general definition. To this end we define a number of standard properties, which define common subclasses which are relatively easy to work with.

implicit beta the equivalence relation is either β or $\beta\eta$

explicit beta the equivalence relation is either η or ε but $\mathscr{R}$ contains the rule `beta`, that is: $(\lambda x_\alpha.Z[x]) \cdot Y \Rightarrow Z[Y]$

parameter-free in all rules, except possibly `beta`, any meta-applications occuring on either side have the form $X[]$

beta-free the system is parameter-free, does not contain `beta`, and its equivalence relation is either η or ε

monomorphic no rule contains type variables, except possibly `beta`

left-beta-normal the left-hand side of each rule (except possibly `beta`) is β-normal

right-beta-normal the right-hand side of each rule is β-normal

beta-normal both left-beta-normal and right-beta-normal

eta-normal both sides of all rules have η-normal form

n^{th} **order** any variable or function symbol occuring in one of the rules has a type of order at most n. A sort-headed type $b(t_1,\ldots,t_n)$ has order 0, a type $\sigma_1 \to \ldots \sigma_m \to b$ with b not composed has order max(order(σ_1),...,order(σ_m))+1; we only speak of order in a monomorphic system

finite $\mathscr{R}$ is finite

algebraic there are no abstractions in the left-hand side of any rule

abstraction-free there are no abstractions in either side of any rule

left-linear no variable occurs twice free in a term

without head variables no left-hand side contains a subterm $X[s_1,\ldots,s_n] \cdot t$

completely without head variables no left-hand side contains a subterm $X[s_1,\ldots,s_n] \cdot t$ or $X \cdot t$ (so bound variables may also not occur at a head).

function-headed the head of the left-hand side of each rule is a function symbol

with base rules the type of a rule may not be composed

Many of these properties can be made to hold, by transforming the system. When we are analysing termination in specific, we can enforce the following properties without affecting either termination or non-termination of the system:

1. any system can be made monomorphic, although at the price of finiteness
2. any system can be presented in beta-normal form
3. a system with explicit beta can be transformed to have implicit beta
4. any algebraic system can be turned abstraction-free
5. any system has a function-headed equivalent without head variables

Moreover, a system can be turned eta-normal and with base rules without losing non-termination (we can even assume η to be part of the equivalence relation); if the transformed system is terminating then so is the original. However, turning a system eta-normal may sometimes lose termination.

4 Embedding existing systems

There are four mainstream forms of higher-order rewriting: Nipkow's HRSs, Jouannaud and Okada's AFSs, Yamada's STTRSs and Klop's CRSs. The latter three can be embedded into HODRSs, but since HRSs in general do not have a decidable reduction relation, they can not; for example a rule $f(X(a)) \Rightarrow X(b)$ can not be represented. However, the common restriction to *pattern HRSs* essentially gives function-headed HODRSs with an equivalence relation $\beta\eta$. An AFS is a parameter-free system with explicit beta, an STTRS is an abstraction-free, beta-free system, and a CRS can be presented as a second-order HODRS with equivalence relation ε. Several quirks need to be ironed out (such as AFSs using function symbols with arity; a symbol f of arity n only occurs in the form $f(s_1,\ldots,s_n)$, and CRS terms being untyped), but this is easy to do.

5 HORPO

The *recursive path ordering*, a common syntactic termination method, has been extended to AFSs in a long line of research, starting with HORPO [2] and culminating in CPO [1]. Any of these definitions can be extended to HODRSs without any real effort; we consider as an example CPO, the latest version. Given a type ordering on monomorphic types and an ordering on function symbols, both wellfounded, CPO defines a wellfounded ordering on monomorphic terms by a set of rules. There is one quirk: in the AFS format used there, function symbols are required to occur with all their arguments. We counter this by only considering (meta-)terms in eta-normal form; as stated in section 3 a system is terminating if it is terminating modulo η. As we will see, there is no need to redo the whole proof that $\succ_{\text{CPO}}$ is wellfounded; we can use the result as it stands.

Choose a wellfounded ordering on all types satisfying the requirements as defined in [1] and moreover $p(\sigma) > p(\tau)$ if $\sigma > \tau$; we can for example do this by extending such an ordering on monomorphic types with the relation $\sigma \to \tau > \rho$ if $\tau \geq \rho$. Choose a wellfounded ordering on all retyped function symbols $f'p$ such that $f_{p(\sigma)} > g_{p(\tau)}$ if $f_\sigma > g_\tau$; we can for example do this by choosing an initial ordering $>'$ on the function names, and defining $f_\sigma > g_\tau$ if $f >' g$ or $f = g$ and τ is a strict subtype of σ. Expand the CPO rules with a clause $X[s_1,\ldots,s_n] \succeq X[t_1,\ldots,t_n]$ if $s_1 \succeq t_1,\ldots,s_n \succeq t_n$.

Having this, the rules of CPO can be applied to polymorphic metaterms in η-normal form, and we can prove: if $s \succ_{\text{CPO}'} t$, p a type substitution mapping to monomorphic terms and γ a substitution, then $s'p\gamma\downarrow_\eta \succ_{\text{CPO}} t'p\gamma\downarrow_\eta$ and, since `beta` is included in $\succ_{\text{CPO}}$, also $s'p\gamma\downarrow_\eta \succ_{\text{CPO}} t'p\gamma\downarrow_{\beta\eta}$. As a system is terminating if and only if there is no infinite reduction over monomorphic terms, wellfoundedness of $\Rightarrow_{\mathscr{R}}$ follows if $l \succ_{\text{CPO}'} r$ for all $l \Rightarrow r \in \mathscr{R}$.

6 Concluding Remarks

We presented the generalised framework of HODRSs and embedded most of the common formalisms in it. The generality of the system does not have to pose a problem for termination re-

search, since by a number of transformations we can make various convenient assumptions about any given program. Moreover, if for example a result on AFSs is instead proved on parameter-free HODRSs, the result immediately holds on a broad subclass of HRSs as well.

We aim to further research the following questions:

1. Under which conditions can we transform a finite polymorphic system to a finite monomorphic system with equivalent termination?
2. Can rules with implicit beta be transformed into rules with explicit beta (as these are often easier to work with)?
3. Which requirements do we need to define dependency pairs?

References

[1] F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: The end of a quest. In *Lecture Notes in Computer Science (CSL '08)*, pages 1–14, July 2008.

[2] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of the 14th annual IEEE Symposium on Logic in Computer Science (LICS '99)*, pages 402–411, Trento, Italy, July 1999.