Higher-Order LCTRSs and Their Termination

Liye Guo and Cynthia Kop

Radboud University, Netherlands

1 Introduction

Logically constrained term rewriting systems (LCTRSs) [4, 1] are a formalism for analyzing programs. In real-world programming, data types such as integers, as opposed to natural numbers, and arrays are prevalent. Any practical program analyzing technique should be prepared to handle these. One of the defining features of the LCTRS formalism is its native support for such data types, which are not (co)inductively defined and need to be encoded if handled by more traditional TRSs. Another benefit of the formalism is its separation between logical constraints modeling the control flow and other terms representing the program states.

So far, program analysis on the basis of LCTRSs has concerned imperative programs since LCTRSs were introduced as a first-order formalism. We are naturally curious to see if functional programs can also be analyzed by constrained rewriting. What we present here is our ongoing exploration in this direction: First, we define a higher-order variant of the LCTRS formalism, which, despite the absence of lambda abstractions, is capable of representing some real-world functional programs straightforwardly. Then we take a brief look at the termination problem for this new formalism as termination analysis is by itself an important aspect of program analysis as well as a prerequisite for determining some other properties.

2 LCSTRS

We start defining *logically constrained simply-typed term rewriting systems* (LCSTRSs) with types and terms. We postulate a set $\mathcal{S}$, whose elements we call *sorts*, and a subset $\mathcal{S}_\vartheta$ of $\mathcal{S}$, whose elements we call *theory sorts*. The set $\mathcal{T}$ of *types* and its subset $\mathcal{T}_\vartheta$, called the set of *theory types*, are generated as follows: $\mathcal{T} ::= \mathcal{S} \mid (\mathcal{T} \to \mathcal{T})$ and $\mathcal{T}_\vartheta ::= \mathcal{S}_\vartheta \mid (\mathcal{S}_\vartheta \to \mathcal{T}_\vartheta)$. Right-associativity is assigned to $\to$ so we can omit some parentheses in types. We assume given disjoint sets $\mathcal{F}$ and $\mathcal{V}$, whose elements we call *function symbols* and *variables*, respectively. The grammar $\mathbb{T} ::= \mathcal{F} \mid \mathcal{V} \mid (\mathbb{T}\ \mathbb{T})$ generates the set $\mathbb{T}$ of *pre-terms*. Left-associativity is assigned to the juxtaposition operation in the above grammar so $t_0\ t_1\ t_2$ stands for $((t_0\ t_1)\ t_2)$, for example. We assume that every function symbol and variable is assigned a unique type. Typing works as expected: if pre-terms t_0 and t_1 have types $A \to B$ and A, respectively, $t_0\ t_1$ has type B. Pre-terms having a type are called *terms*. We write $t : A$ if a term t has type A. We postulate a subset $\mathcal{F}_\vartheta$ of $\mathcal{F}$, whose elements we call *theory (function) symbols*, and assume that theory symbols have theory types. Terms constructed with only theory symbols and variables are called *logical terms*. The set of variables in a term t, denoted by $\mathrm{Var}(t)$, is defined as follows: $\mathrm{Var}(f) = \emptyset$, $\mathrm{Var}(x) = \{\, x \,\}$ and $\mathrm{Var}(t_0\ t_1) = \mathrm{Var}(t_0) \cup \mathrm{Var}(t_1)$. A term t is called a *ground term* if $\mathrm{Var}(t) = \emptyset$. Note that ground logical terms always have theory types.

Logical terms are distinguished because they will be treated specially when we define the rewrite relation. First, let us define the interpretation of ground logical terms. We postulate an $\mathcal{S}_\vartheta$-indexed family of sets $(\mathfrak{X}_A)_{A \in \mathcal{S}_\vartheta}$, and extend it to a $\mathcal{T}_\vartheta$-indexed family of sets by letting $\mathfrak{X}_{A \to B}$ be the set of maps from $\mathfrak{X}_A$ to $\mathfrak{X}_B$. Now we assume given a $\mathcal{T}_\vartheta$-indexed family of maps $(\llbracket\cdot\rrbracket_A)_{A \in \mathcal{T}_\vartheta}$ where $\llbracket\cdot\rrbracket_A$ assigns to each theory symbol whose type is A an element of $\mathfrak{X}_A$ and is

bijective if $A \in \mathcal{S}_\vartheta$. Theory symbols whose type is a theory sort are called *values*. We extend each indexed map $[\![\cdot]\!]_B$ to a map that assigns to each **ground logical term** whose type is B an element of $\mathfrak{X}_B$ by letting $[\![t_0\ t_1]\!]_B$ be $[\![t_0]\!]_{A\to B}([\![t_1]\!]_A)$. We omit the type and write just $[\![\cdot]\!]$ whenever the type can be deduced from the context. $[\![t]\!]$ is called the *interpretation* of t.

A *substitution* is a type-preserving map from variables to terms. Every substitution σ extends to a type-preserving map $\bar{\sigma}$ from terms to terms. We write $t\sigma$ for $\bar{\sigma}(t)$ and define it as follows: $f\sigma = f$, $x\sigma = \sigma(x)$ and $(t_0\ t_1)\sigma = (t_0\sigma)\ (t_1\sigma)$. Now we postulate a theory sort $\mathbb{B}$ and theory symbols $\bot : \mathbb{B}$ and $\top : \mathbb{B}$. Let $\mathfrak{X}_\mathbb{B}$ be $\{\, \mathtt{o}, \mathtt{i} \,\}$ and assume $[\![\bot]\!] = \mathtt{o}$ and $[\![\top]\!] = \mathtt{i}$. A *rewrite rule* $\ell \to r\ [\varphi]$ is a triple where (i) ℓ and r are terms which have the same type, (ii) ℓ is not a logical term, (iii) φ is a *logical constraint*, i.e., φ is a logical term whose type is $\mathbb{B}$ and the type of each variable in $\mathrm{Var}(\varphi)$ is a theory sort, and (iv) the type of each variable in $\mathrm{Var}(r) \setminus \mathrm{Var}(\ell)$ is a theory sort. A substitution σ is said to *respect* a rewrite rule $\ell \to r\ [\varphi]$ if $\sigma(x)$ is a value for all $x \in \mathrm{Var}(\varphi) \cup (\mathrm{Var}(r) \setminus \mathrm{Var}(\ell))$ and $[\![\varphi\sigma]\!] = \mathtt{i}$. A set $\mathcal{R}$ of rewrite rules induces a *rewrite relation* $\to_\mathcal{R}$ on terms such that $t \to_\mathcal{R} t'$ if and only if one of the following conditions is true:

- $t = \ell\sigma$ and $t' = r\sigma$ for some $\ell \to r\ [\varphi] \in \mathcal{R}$ and some substitution σ that respects $\ell \to r\ [\varphi]$.
- $t = f\ v_1 \cdots v_n$ where f is a theory symbol but not a value while v_i is a value for all i, the type of t is a theory sort, and t' is the unique value such that $[\![f\ v_1 \cdots v_n]\!] = [\![t']\!]$.
- $t = t_0\ t_1$, $t' = t_0'\ t_1$ and $t_0 \to_\mathcal{R} t_0'$.
- $t = t_0\ t_1$, $t' = t_0\ t_1'$ and $t_1 \to_\mathcal{R} t_1'$.

Logical constraints are essentially first-order—higher-order variables are excluded and theory symbols take only first-order arguments. We adopt this restriction because many conditions in functional programs are still first-order and solving higher-order constraints is hard. That is not to say that higher-order constraints are of no interest; we simply leave them out of the scope of LCSTRSs.

Below is an example LCSTRS:

$$\begin{array}{rcll} \mathsf{init} \to \mathsf{fact}\ n\ \mathsf{exit} & [\top] & \mathsf{fact}\ n\ k \to k\ 1 & [n \le 0] \\ \mathsf{comp}\ g\ f\ x \to g\ (f\ x) & [\top] & \mathsf{fact}\ n\ k \to \mathsf{fact}\ (n-1)\ (\mathsf{comp}\ k\ (*\ n)) & [n > 0] \end{array}$$

Here init and exit denote the start and the end of the program, respectively. The core of the program is fact, which computes the factorial function in continuation-passing style, and comp is an auxiliary function for function composition. Integer literals and operators are theory symbols. Note that we use infix notation to improve readability. The occurrence of n in the rewrite rule defining init is an example of a variable that occurs on the right-hand side but not on the left-hand side of a rewrite rule. Such variables can be used to model user input.

Let $\mathcal{R}$ denote the set of rewrite rules in the example and consider the rewrite sequence

$$\mathsf{fact}\ 1\ \mathsf{exit} \to_\mathcal{R} \mathsf{fact}\ (1-1)\ (\mathsf{comp}\ \mathsf{exit}\ (*\ 1)) \to_\mathcal{R} \mathsf{fact}\ 0\ (\mathsf{comp}\ \mathsf{exit}\ (*\ 1)) \to_\mathcal{R} \mathsf{comp}\ \mathsf{exit}\ (*\ 1)\ 1.$$

In the second step, no rewrite rule is invoked. Such rewrite steps are called *calculation steps*. We can write $\to_\emptyset$ for a calculation step. Terms s and t are said to be *joinable* by $\to_\emptyset$, written as $s \downarrow_\emptyset t$, if there exists a term r such that $s \to_\emptyset^* r$ and $t \to_\emptyset^* r$.

3 Termination

In order to prove that a given (unconstrained) TRS $\mathcal{R}$ is terminating, we usually look for a stable, monotonic and well-founded relation $\succ$ which orients every rewrite rule in $\mathcal{R}$, i.e., $\ell \succ r$ for all $\ell \to r \in \mathcal{R}$. This standard technique, however, requires a few tweaks to be applied to LCSTRSs. First, stability should be tightly coupled with rule orientation because every rewrite rule in an LCSTRS is equipped with a logical constraint, which decides what substitutions are expected when the rewrite rule is invoked. Therefore, we say that a type-preserving relation $\succ$ on terms *orients* a rewrite rule $\ell \to r \ [\varphi]$ if $\ell\sigma \succ r\sigma$ for each substitution σ that **respects** the rewrite rule. Second, the monotonicity requirement can be weakened because ℓ is never a logical term in a rewrite rule $\ell \to r \ [\varphi]$. We say that a type-preserving relation $\succ$ on terms is *rule-monotonic* if $t_0 \succ t_0'$ implies $t_0 \ t_1 \succ t_0' \ t_1$ when t_0 is not a logical term, and $t_1 \succ t_1'$ implies $t_0 \ t_1 \succ t_0 \ t_1'$ when t_1 is not a logical term.

We present a tentative definition of HORPO [2] on LCSTRSs. For each theory sort A, we postulate theory symbols $\sqsupset_A : A \to A \to \mathbb{B}$ and $\sqsupseteq_A : A \to A \to \mathbb{B}$ such that $[\![\sqsupset_A]\!]$ is a well-founded ordering on $\mathfrak{X}_A$ and $[\![\sqsupseteq_A]\!]$ is the reflexive closure of $[\![\sqsupset_A]\!]$. We omit the sort and write just $\sqsupset$ and $\sqsupseteq$ whenever the sort can be deduced from the context. Given the *precedence* $\blacktriangleright$, a well-founded ordering on function symbols such that $f \blacktriangleright g$ for all $f \in \mathcal{F} \setminus \mathcal{F}_\vartheta$ and $g \in \mathcal{F}_\vartheta$, and the *status* stat, a map from $\mathcal{F}$ to $\{\mathsf{l}, \mathsf{m}_2, \mathsf{m}_3, \dots\}$, the *higher-order recursive path ordering* (HORPO) $(\succsim_\varphi, \succ_\varphi)$ is a family of type-preserving relation pairs on terms indexed by logical constraints and defined as follows:

- $s \succsim_\varphi t$ if and only if one of the following conditions is true:
  - s and t are logical terms whose type is a theory sort, $\mathrm{Var}(\varphi) \supseteq \mathrm{Var}(s) \cup \mathrm{Var}(t)$ and $\varphi \models \sqsupseteq \ s \ t$.
  - $s \succ_\varphi t$.
  - $s \downarrow_\emptyset t$.
  - s is not a logical term, $s = s_1 \ s_2$, $t = t_1 \ t_2$, $s_1 \succsim_\varphi t_1$ and $s_2 \succsim_\varphi t_2$.
- $s \succ_\varphi t$ if and only if one of the following conditions is true:
  - s and t are logical terms whose type is a theory sort, $\mathrm{Var}(\varphi) \supseteq \mathrm{Var}(s) \cup \mathrm{Var}(t)$ and $\varphi \models \sqsupset \ s \ t$.
  - s and t have the same type and $s \triangleright_\varphi t$.
  - s is not a logical term, $s = x \ s_1 \cdots s_n$ where x is a variable, $t = x \ t_1 \cdots t_n$, $s_i \succsim_\varphi t_i$ for all i and there exists i such that $s_i \succ_\varphi t_i$.
- $s \triangleright_\varphi t$ if and only if s is not a logical term, $s = f \ s_1 \cdots s_m$ where f is a function symbol, and one of the following conditions is true:
  - $s_i \succsim_\varphi t$ for some i.
  - $t = t_0 \ t_1 \cdots t_n$ and $s \triangleright_\varphi t_i$ for all i.
  - $t = g \ t_1 \cdots t_n$, $f \blacktriangleright g$ and $s \triangleright_\varphi t_i$ for all i.
  - $t = f \ t_1 \cdots t_n$, $\mathsf{stat}(f) = \mathsf{l}$, $s_1 \cdots s_m \succ_\varphi^{\mathsf{l}} t_1 \cdots t_n$ and $s \triangleright_\varphi t_i$ for all i.
  - $t = f \ t_1 \cdots t_n$, $\mathsf{stat}(f) = \mathsf{m}_k$, $k \leq n$, $s_1 \cdots s_{\min(m,k)} \succ_\varphi^{\mathsf{m}} t_1 \cdots t_k$ and $s \triangleright_\varphi t_i$ for all i.

In the above, $s_1 \cdots s_m \succ_\varphi^{\mathfrak{l}} t_1 \cdots t_n$ if and only if $\exists i \leq \min(m,n)$ $(s_i \succ_\varphi t_i \wedge \forall j < i\ s_j \succsim_\varphi t_j)$, $\succ_\varphi^{\mathfrak{m}}$ is the generalized multiset extension of $(\succsim_\varphi, \succ_\varphi)$ (see [3]), and $\varphi \models \varphi'$ denotes, on the assumption that φ and φ' are logical constraints such that $\mathrm{Var}(\varphi) \supseteq \mathrm{Var}(\varphi')$, that for each substitution σ which maps variables in $\mathrm{Var}(\varphi)$ to values, $[\![\varphi\sigma]\!] = \mathtt{1}$ implies $[\![\varphi'\sigma]\!] = \mathtt{1}$.

The design is that $\succ_\top$ should orient a rewrite rule $\ell \to r\ [\varphi]$ if $\ell \succ_\varphi r$. Then once a combination of $\sqsupset$, $\blacktriangleright$ and stat that guarantees $\ell \succ_\varphi r$ for all $\ell \to r\ [\varphi] \in \mathcal{R}$ is present, we can conclude that the LCSTRS $\mathcal{R}$ is terminating. The soundness of this method relies on the following properties of $\succ_\varphi$, which we must prove:

- $\succ_\top$ orients $\ell \to r\ [\varphi]$ if $\ell \succ_\varphi r$.
- $\succ_\top$ is rule-monotonic.
- $\succ_\top$ is well-founded.
- $\to_\emptyset \,;\, \succ_\top \subseteq \succ_\top$.

Note that $\to_\emptyset$ is well-founded because the size strictly decreases through every calculation step.

Consider the example LCSTRS given in the previous section. Any combination of $\sqsupset$, $\blacktriangleright$ and stat that satisfies the following properties would witness the well-foundedness of $\to_\mathcal{R}$: $[\![\sqsupset]\!] = \lambda xy.\ x > 0 \wedge x > y$, $\mathsf{init} \blacktriangleright \mathsf{fact} \blacktriangleright \mathsf{comp}$, $\mathsf{init} \blacktriangleright \mathsf{exit}$ and $\mathsf{stat}(\mathsf{fact}) = \mathfrak{l}$.

4 Future Work

LCSTRSs are still a work in progress. While the formalism itself is in a somewhat stable state, the above method for termination analysis is in active development. First and foremost, we need to prove that HORPO on LCSTRSs has the expected properties. When the theory is complete, we would like to make a tool to automate the finding of HORPO on LCSTRSs. It would also be interesting to explore other methods for termination analysis on the new formalism, including StarHorpo [3] (a transitive variant of HORPO), interpretation-based methods and dependency pairs. Another direction is to go beyond LCSTRSs by augmenting the formalism with lambda abstractions or higher-order constraints.

References

[1] C. Fuhs, C. Kop, and N. Nishida. Verifying procedural programs via constrained rewriting induction. *ACM Transactions on Computational Logic*, 18(2):14:1–14:50, 2017.

[2] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proc. LICS*, pages 402–411, 1999.

[3] C. Kop. *Higher Order Termination*. PhD thesis, VU Amsterdam, 2012.

[4] C. Kop and N. Nishida. Term rewriting with logical constraints. In *Proc. FroCoS*, pages 343–358, 2013.