Termination of LCTRSs*

Cynthia Kop[1]

1 **Department of Computer Science, University of Innsbruck**
Technikerstraße 21a, 6020 Innsbruck, Austria
`Cynthia.Kop@uibk.ac.at`

Abstract

Logically Constrained Term Rewriting Systems (LCTRSs) provide a general framework for term rewriting with constraints. We discuss a simple dependency pair approach to prove termination of LCTRSs. We see that existing techniques transfer to the constrained setting in a natural way.

1 Introduction

In [4], *logically constrained term rewriting systems* are introduced (building on [3] and [2]). These *LCTRSs* combine many-sorted term rewriting with constraints in an arbitrary theory, and can be used for analysing for instance imperative programs.

Termination is an important part of such analysis, both for its own sake (to guarantee finite program evaluation), and to create an induction principle that can be used as part of other analyses (for instance proofs of confluence [6] or function equality [3]).

In unconstrained term rewriting, many termination techniques exist, often centred around *dependency pairs* [1]. Some of these methods have also been transposed to integer rewriting with constraints [2]. However, that setting is focused purely on proving termination for its own sake, and thus poses very strong restrictions on term and rule formation.

In this paper, we will see how a basic dependency pair approach can be defined for LCTRSs, and extend several termination methods which build around dependency pairs.

2 Preliminaries (from [4])

We assume standard notions of many-sorted term rewriting to be well-uderstood.

Let $\mathcal{V}$ be an infinite set of sorted variables, $\Sigma = \Sigma_{terms} \cup \Sigma_{theory}$ be a many-sorted signature, $\mathcal{I}$ a mapping which assigns to each sort occurring in Σ_{theory} a set, and $\mathcal{J}$ a function which maps each $f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa \in \Sigma_{theory}$ to a function $\mathcal{J}_f$ in $\mathcal{I}_{\iota_1} \Longrightarrow \ldots \Longrightarrow \mathcal{I}_{\iota_n} \Longrightarrow \mathcal{I}_\kappa$. For every sort ι occurring in Σ_{theory} we also fix a set $\mathcal{V}al_\iota \subseteq \Sigma_{theory}$ of *values*: function symbols $a : [] \Rightarrow \iota$, where $\mathcal{J}$ gives a one-to-one mapping from $\mathcal{V}al_\iota$ to $\mathcal{I}_\iota$. A value c is identified with the term $c()$. The elements of Σ_{theory} and Σ_{terms} overlap only on values.

We call a term in $\mathcal{T}erms(\Sigma_{theory}, \mathcal{V})$ a *logical term*. For ground logical terms, we define $[\![f(s_1, \ldots, s_n)]\!] := \mathcal{J}_f([\![s_1]\!], \ldots, [\![s_n]\!])$. A ground logical term s *has value* t if t is a value such that $[\![s]\!] = [\![t]\!]$. Every ground logical term has a unique value. A *constraint* is a logical term of some sort bool with $\mathcal{I}_{\mathsf{bool}} = \mathbb{B}$, the set of booleans. A constraint s is *valid* if $[\![s\gamma]\!]_{\mathcal{J}} = \top$ for all substitutions γ which map the variables in $Var(s)$ to a value.

A *rule* is a triple $l \to r\ [\varphi]$ where l and r are terms with the same sort and φ is a constraint; l is not a logical term (so also not a variable). If $\varphi = \mathsf{true}$ with $\mathcal{J}(\mathsf{true}) = \top$, the rule is just denoted $l \to r$. We define $LVar(l \to r\ [\varphi]))$ as $Var(\varphi) \cup (\,Var(r) \setminus Var(l))$. A substitution γ *respects* $l \to r\ [\varphi]$ if $\gamma(x)$ is a value for all $x \in LVar(l \to r\ [\varphi])$ and $\varphi\gamma$ is valid.

* The research described in this paper is supported by the Austrian Science Fund (FWF) international project I963 and the Japan Society for the Promotion of Science.

Given a set of rules $\mathcal{R}$, the *rewrite relation* $\to_{\mathcal{R}}$ is the union of $\to_{\mathtt{rule}}$ and $\to_{\mathtt{calc}}$, where:

- $C[l\gamma] \to_{\mathtt{rule}} C[r\gamma]$ if $l \to r\ [\varphi] \in \mathcal{R}$ and γ respects $l \to r\ [\varphi]$;
- $C[f(s_1, \ldots, s_n)] \to_{\mathtt{calc}} C[v]$ if $f \in \Sigma_{theory} \setminus \Sigma_{terms}$, all s_i values and v is the value of $f(\vec{s})$

A reduction step with $\to_{\mathtt{calc}}$ is called a *calculation*. In an LCTRS with rules $\mathcal{R}$, the *defined symbols* are all symbols f such that a rule $f(\vec{l}) \to r\ [\varphi]$ exists in $\mathcal{R}$. Symbols $f \in \Sigma_{theory} \setminus \mathcal{V}al$ are called *calculation symbols* and all other symbols are *constructors*.

▶ **Example 1.** We consider an LCTRS with sorts int and bool, with $\mathcal{I}_{\mathsf{bool}} = \mathbb{B}$ and int mapped to the set of 16-bit signed integers; addition is sensitive to overflow. The rules are a naive implementation of the Ackermann function (which will likely fall prey to overflows):

$$\begin{array}{lcll} \mathsf{A}(m,n) & \to & \mathsf{A}(m-1,\mathsf{A}(m,n-1)) & [m \neq 0 \wedge n \neq 0] \\ \mathsf{A}(m,0) & \to & \mathsf{A}(m-1,1) & [m \neq 0] \end{array} \qquad \begin{array}{lcl} \mathsf{A}(0,n) & \to & n+1 \end{array}$$

A is a defined symbols, $+, -, \neq, \wedge$ calculation symbols, and all integers are constructors.

3 Dependency Pairs

As the basis for termination analysis, we will consider *dependency pairs* [1]. We first introduce a fresh sort dpsort, and for all defined symbols $f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa$ also a new symbol $f^\sharp : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \mathsf{dpsort}$. If $s = f(s_1, \ldots, s_n)$ with f defined, then $s^\sharp := f^\sharp(s_1, \ldots, s_n)$.

The dependency pairs of a given rule $l \to r\ [\varphi]$ are all rules of the form $l^\sharp \to p^\sharp\ [\varphi]$ where p is a subterm of r which is headed by a defined symbol. The set of dependency pairs for a given set of rules $\mathcal{R}$, notation $\mathsf{DP}(\mathcal{R})$, consists of all dependency pairs of any rule in $\mathcal{R}$.

▶ **Example 2.** Noting that for instance $\mathsf{A}^\sharp(m,0) \to m -^\sharp 1$ is *not* a dependency pair, since $-$ is a calculation symbol and not a defined symbol, Example 1 has three dependency pairs:

$$\begin{array}{lcll} 1. & \mathsf{A}^\sharp(m,0) & \to & \mathsf{A}^\sharp(m-1,1) & [m \neq 0] \\ 2. & \mathsf{A}^\sharp(m,n) & \to & \mathsf{A}^\sharp(m-1,\mathsf{A}(m,n-1)) & [m \neq 0 \wedge n \neq 0] \\ 3. & \mathsf{A}^\sharp(m,n) & \to & \mathsf{A}^\sharp(m,n-1) & [m \neq 0 \wedge n \neq 0] \end{array}$$

Fixing a set $\mathcal{R}$ of rules, and given a set $\mathcal{P}$ of dependency pairs, a *$\mathcal{P}$-chain* is a sequence $\rho_1, \rho_2, \ldots$ of dependency pairs such that all ρ_i are elements of $\mathcal{P}$, but with distinctly renamed variables, and there is some γ which respects all ρ_i, such that for all i: if $\rho_i = l_i \to p_i\ [\varphi_i]$ and $\rho_{i+1} = l_{i+1} \to p_{i+1}\ [\varphi_{i+1}]$, then $p_i\gamma \to^*_{\mathcal{R}} l_{i+1}\gamma$. Also, the strict subterms of $l_i\gamma$ terminate. We call $\mathcal{P}$ a *DP problem* and say that $\mathcal{P}$ is *chain-free* if there is no infinite $\mathcal{P}$-chain.[1][2]

▶ **Theorem 3.** *An LCTRS $\mathcal{R}$ is terminating if and only if $\mathsf{DP}(\mathcal{R})$ is chain-free.*

4 The Dependency Graph

To prove chain-freeness of a DP problem, we might for instance use the dependency graph:

▶ **Definition 4.** A *dependency graph approximation* of a DP problem $\mathcal{P}$ is a graph G whose nodes are the elements of $\mathcal{P}$ and which has an edge between ρ_1 and ρ_2 if (ρ_1, ρ_2') is a $\mathcal{P}$-chain, where ρ_2' is a copy of ρ_2 with fresh variables. G may have additional edges.

▶ **Theorem 5.** *A DP problem $\mathcal{P}$ with graph approximation G is chain-free if and only if $\mathcal{P}'$ is chain-free for every strongly connected component (SCC) $\mathcal{P}'$ of G.*

---

[1] In the literature, we consider tuples of sets and flags, which is necessary if we also want to consider non-minimal chains, innermost termination or non-termination. For simplicity those are omitted here.

[2] In the literature, the word *finite* is used instead of *chain-free*. Since we have a single set instead of a tuple, we used a different word to avoid confusion (as "finite" might refer to the number of elements).

▶ **Example 6.** Consider an LCTRS with rules $\mathcal{R} = \{f(x) \to f(0 - x)\ [x > 0]\}$. Then $\mathsf{DP}(\mathcal{R}) = \{f^\sharp(x) \to f^\sharp(-x)\ [x > 0]\}$. The dependency graph of $\mathsf{DP}(\mathcal{R})$ has one node, and no edges, since there is no substitution γ which satisfies both $\gamma(x) > 0$ and $\gamma(y) > 0$ and yet has $(-x)\gamma \to^*_{\mathcal{R}} y\gamma$ (as logical terms reduce only with $\to_{\mathtt{calc}}$). Thus, clearly every SCC of this graph is terminating, so $\mathsf{DP}(\mathcal{R})$ is chain-free, so $\mathcal{R}$ is terminating!

Of course, manually choosing a graph approximation is one thing, but finding a good one *automatically* is more difficult. We consider one way to choose such an approximation:

Given a DP problem $\mathcal{P}$, let $G_\mathcal{P}$ be the graph with the elements of $\mathcal{P}$ as nodes, and with an edge from $l_1 \to r_1\ [\varphi_1]$ to $l_2 \to r_2\ [\varphi_2]$ if the formula $\varphi_1 \wedge \varphi_2' \wedge \psi(r_1, l_2', LVar(l_1 \to r_1\ [\varphi_1]) \cup LVar(l_2' \to r_2'\ [\varphi_2']))$ is satisfiable (or its satisfiability cannot be determined). Here, $l_2' \to r_2'\ [\varphi_2']$ is a copy of $l_2 \to r_2\ [\varphi_2]$ with fresh variables, and $\psi(s, t, L)$ is given by the clauses:

- $\psi(s, t, L) = \top$ if either s is a variable not in L, or $s = f(s_1, \ldots, s_n)$ and one of:
  - f is a defined symbol, and $s \notin \mathcal{T}erms(\Sigma_{theory}, L)$,
  - f is a calculation symbol, t a value or variable, and $s \notin \mathcal{T}erms(\Sigma_{theory}, L)$,
  - f is a constructor and t a variable not in L;
- $\psi(s, t, L) = \bigwedge_{i=1}^n \psi(s_i, t_i, L)$ if $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and f not defined;
- $\psi(s, t, L)$ is the formula $s = t$ if $s \in \mathcal{T}erms(\Sigma_{theory}, L)$, $t \in \mathcal{T}erms(\Sigma_{theory}, \mathcal{V})$ and s and t are not headed by the same theory symbol (we already covered that case);
- $\psi(s, t, L) = \bot$ in all other cases.

▶ **Theorem 7.** *$G_\mathcal{P}$ is a graph approximation for $\mathcal{P}$.*

This graph result and the given approximation correspond largely with the result of [5].

▶ **Example 8.** The graph in Example 6 is calculated with this method: $\psi(f^\sharp(-x), f^\sharp(y), \{x, y\}) \wedge x > 0 \wedge y > 0$ evaluates to $-x = y \wedge x > 0 \wedge y > 0$ (as $f^\sharp$ is a constructor with respect to $\mathcal{R}$), which is not satisfiable (as any decent SMT-solver over the integers can tell us).

5 The Value Criterion

To quickly handle DP problems, we consider a technique similar to the subterm criterion in the unconstrained case. This *value criterion* can also be seen as a simpler version of polynomial interpretations, which does not require ordering rules (see Section 6).

▶ **Definition 9.** Fixing a set $\mathcal{P}$ of dependency pairs, a *projection function* for $\mathcal{P}$ is a function ν which assigns to each symbol $f^\sharp : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \mathsf{dpsort}$ a number $\nu(f^\sharp) \in \{1, \ldots, n\}$. A projection function is extended to a function on terms as follows: $\overline{\nu}(f^\sharp(s_1, \ldots, s_n)) = s_{\nu(f^\sharp)}$.

▶ **Theorem 10.** *Let $\mathcal{P}$ be a set of dependency pairs, ι a sort and ν a projection function for $\mathcal{P}$, with the following property: for any dependency pair $l \to r\ [\varphi] \in \mathcal{P}$, if $\overline{\nu}(l)$ has sort ι and is a logical term (this includes variables), then the same holds for $\overline{\nu}(r)$. Let moreover $\succ$ be a well-founded ordering relation on $\mathcal{I}_\iota$ and $\succeq$ a quasi-ordering such that $\succ \cdot \succeq \subseteq \succ$. Suppose additionally that we can write $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$, such that for all $\rho = l \to r\ [\varphi] \in \mathcal{P}$:*

- *if $\overline{\nu}(l)$ is a logical term of sort ι, then so is $\overline{\nu}(r)$, and $Var(\overline{\nu}(r)) \subseteq Var(\overline{\nu}(l))$;*
- *if $\rho \in \mathcal{P}_1$, then $\overline{\nu}(l)$ has sort ι and $\overline{\nu}(l) \in \mathcal{T}erms(\Sigma_{theory}, LVar(\rho))$;*
- *if $\overline{\nu}(l)$ has sort ι and $\overline{\nu}(l) \in \mathcal{T}erms(\Sigma_{theory}, \mathcal{V})$, then $\varphi \Rightarrow \overline{\nu}(l) \succ \overline{\nu}(r)$ is valid if $\rho \in \mathcal{P}_1$, and $\varphi \Rightarrow \overline{\nu}(l) \succeq \overline{\nu}(r)$ is valid if $\rho \in \mathcal{P}_2$.*

Then $\mathcal{P}$ is chain-free if and only if $\mathcal{P}_2$ is chain-free.

Proof. A chain with infinitely many elements of $\mathcal{P}_1$ gives an infinite $\succeq^* \cdot \succ$ reduction. ◀

▶ **Example 11.** Using the value criterion, we can complete termination analysis of the Ackermann example. Choosing for $\succ$ the *unsigned* comparison on bitvectors (so $n \succ m$ if either n is negative and m is not, or $sign(n) = sign(m)$ and $n > m$), and $\nu(\mathsf{A}) = 1$, we have:

- $\mathsf{A}^\sharp(m, 0) \to \mathsf{A}^\sharp(m-1, 1)$ $[m \neq 0]$: $(m \neq 0) \Rightarrow m \succ m-1$
- $\mathsf{A}^\sharp(m, n) \to \mathsf{A}^\sharp(m-1, \mathsf{A}(m, n-1))$ $[m \neq 0 \wedge n \neq 0]$: $(m \neq 0 \wedge n \neq 0) \Rightarrow m \succ m-1$
- $\mathsf{A}^\sharp(m, n) \to \mathsf{A}^\sharp(m, n-1)$ $[m \neq 0 \wedge n \neq 0]$ $(m \neq 0 \wedge n \neq 0) \Rightarrow m \succeq m$

All three are valid, so $\mathcal{P}$ is chain-free if $\mathcal{P}_2 = \{\mathsf{A}^\sharp(m, n) \to \mathsf{A}^\sharp(m, n-1)\ [m \neq 0 \wedge 0 \wedge n \neq 0]\}$ is. This we prove with another application of the value criterion, now taking $\nu(\mathsf{A}^\sharp) = 2$.

Note that the difficulty to apply the value criterion is in finding a suitable value ordering. There are various systematic techniques for doing this (depending on the underlying theory), but their specifics are beyond the scope of this paper.

6 Reduction Pairs

Finally, the most common method to prove chain-freeness is the use of a *reduction pair*.

A reduction pair $(\succsim, \succ)$ is a pair of a *monotonic quasi-ordering* and a *well-founded partial ordering* on terms such that $s \succ t \succsim q$ implies $s \succ q$. Note that it is not required that $\succ$ is included in $\succsim$; $\succsim$ might also for instance be an equivalence relation. A rule $l \to r\ [\varphi]$ is *compatible* with $R \in \{\succsim, \succ\}$ if for all substitutions γ which respect the rule we have: $l\gamma\ R\ r\gamma$.

▶ **Theorem 12.** *A set of dependency pairs $\mathcal{P}$ is chain-free if and only if there is a reduction pair $(\succsim, \succ)$ and we can write $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ such that $\mathcal{P}_2$ is chain-free, and:*

- *all $\rho \in \mathcal{P}_1$ are compatible with $\succ$ and all $\rho \in \mathcal{P}_2$ are compatible with $\succsim$;*
- *either all $\rho \in \mathcal{R}$ are compatible with $\succsim$,*
  or all $\rho \in \mathcal{P}$ have the form $l \to f(s_1, \ldots, s_i)\ [\varphi]$ with all $s_i \in \mathcal{T}erms(\Sigma_{theory}, LVar(\rho))$;
- *$f(\vec{v}) \succsim w$ if f is a calculation symbol, $v_1, \ldots, v_n$ are values and w is the value of $f(\vec{v})$.*

Note that all rules must be compatible with $\succsim$, unless the subterms of the right-hand sides in $\mathcal{P}$ can only be instantiated to ground logical terms; in this (reasonably common!) case, we can ignore the rules in the termination argument. This is a weak step in the direction of *usable rules*, a full treatment of which is beyond the scope of this short paper.

For the reduction pair, we might for instance use the recursive path ordering described in [4]. Alternatively, we could consider *polynomial interpretations*:

▶ **Theorem 13.** *Given a mapping μ which assigns to each function symbol $f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa \in \Sigma_{terms} \cup \Sigma_{theory}$ an n-ary polynomial over $\mathbb{Z}$, and a valuation α which maps each variable to an integer, every term s corresponds to an integer $\overline{\mu}_\alpha(s)$. Let $s \succ t$ if for all α: $\overline{\mu}_\alpha(s) > \max(0, \overline{\mu}_\alpha(t))$, and $s \succsim t$ if for all α: $\overline{\mu}_\alpha(s) = \overline{\mu}_\alpha(t)$. Then $(\succsim, \succ)$ is a reduction pair.*

Here, $\succsim$ is an equivalence relation. Alternatively we might base $\succsim$ on the $\geq$ relation in $\mathbb{Z}$, but then we must pose an additional weak monotonicity requirement on μ.

▶ **Example 14.** We consider an LCTRS over the integers, without overflow. This example uses bounded iteration, which is common in systems derived from imperative programs:

$$\mathsf{sum}(x, y) \to 0\ [x > y] \qquad \mathsf{sum}(x, y) \to x + \mathsf{sum}(x+1, y)\ [x \leq y]$$

This system admits one dependency pair: $\mathsf{sum}^\sharp(x, y) \to \mathsf{sum}^\sharp(x+1, y)\ [x \leq y]$. Neither the dependency graph nor the value criterion can handle this pair. We *can* orient it using polynomial interpretations, with $\mu(\mathsf{sum}) = \lambda nm.m - n + 1$; integer functions and integers are interpreted as themselves. Then $x \leq y \Rightarrow y - x + 1 > \max(0, y - (x+1) + 1)$ is valid, so the pair is compatible with $\succ$ as required.

Thus, $\mathsf{DP}(\mathcal{R})$ is chain-free if and only if $\emptyset$ is chain-free, which is obviously the case!

7 Related Work

The most important related work is [2], where a constrained term rewriting formalism over the integers is introduced, and methods are developed to prove termination similar to the ones discussed here. The major difference with the current work is that the authors of [2] impose very strong type restrictions: they consider only theory symbols (of sort `int`) and defined symbols (of sort `unit`). Rules have the form $f(x_1, \ldots, x_n) \to g(s_1, \ldots, s_n)$, where the x_i are variables and all s_i are logical terms. This significantly simplifies the analysis (for example, the dependency pairs are exactly the rules), but has more limited applications; it suffices for proving termination of simple (imperative) integer programs, but does not help directly for analysing confluence or function equivalence.

8 Conclusion

In this paper, we have seen how termination methods for normal TRSs, and in particular the dependency pair approach, extend naturally to the setting of LCTRSs. Decision procedures are handled by solving validity of logical formulas. While this is undecidable in general, many practical cases can be handled using today's powerful SMT-solvers.

Considering termination results, we have only seen the tip of the iceberg. In the future, we hope to extend the constrained dependency pair framework to handle also innermost termination and non-termination. Moreover, the dependency pair approach can be strengthened with various techniques for simplifying dependency pair processors, both adaptations of existing techniques for unconstrained term rewriting (such as usable rules) and specific methods for constrained term rewriting (such as the *chaining* method used in [2] or methods to add constraints in some cases).

In addition, we hope to provide an automated termination tool for LCTRSs in the near future. Such a tool could for instance be coupled with a transformation tool from e.g. C or Java to be immediately applicable for proving termination of imperative programs, or can be used as a back-end for analysis tools of confluence or function equivalence.

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236(1-2):133–178, 2000.

2 S. Falke and D. Kapur. A term rewriting approach to the automated termination analysis of imperative programs. In R. Schmidt, editor, *Proc. CADE 09*, volume 5663 of *LNCS*, pages 277–293. Springer, 2009.

3 Y. Furuichi, N. Nishida, M. Sakai, K. Kusakari, and T. Sakabe. Approach to procedural-program verification based on implicit induction of constrained term rewriting systems. *IPSJ Transactions on Programming*, 1(2):100–121, 2008. In Japanese.

4 C. Kop and N. Nishida. Term rewriting with logical constraints. In *Proc. FroCoS 13*, 2013. To Appear, `http://cl-informatik.uibk.ac.at/users/kop/frocos13.pdf`.

5 T. Sakata, N. Nishida, and T. Sakabe. On proving termination of constrained term rewrite systems by eliminating edges from dependency graphs. In H. Kuchen, editor, *Proc. WFLP 11*, LNCS, pages 138–155. Springer, 2011.

6 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in TCS*. Cambridge University Press, 2003.