Term Rewriting with Logical Constraints

Cynthia Kop and Naoki Nishida

This is a pre-publishing copy of the published paper, which includes an appendix and corrections, and is more spatially verbose in some places. All non-aesthetical changes are listed below.
For the published version, please visit
`http://link.springer.com/chapter/10.1007/978-3-642-40885-4_24`

Rectifications

Compared to the version published in Springer, this paper has some minor modifications.

- In the translation of $\mathbb{Z}$-TRSs to LCTRS, the sort declaration of predicates $=, >, \geq$ was accidentally set to $[\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{int}$; this should be $[\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{bool}$.
- The definition of a trivial critical pair (Definition 2) in the original version required only that $s\ [\varphi] \approx t\ [\varphi]$, which incorrectly has $\langle f(x), f(y), x \neq y \rangle$ trivial. The intended definition (which is used in this version of the paper) states that for any substitution which respects $s\ [\varphi]$ we have $s\gamma = t\gamma$.
- In Theorem 5 the original version had a typo, $l \succeq r$ which should be $l \succ r$.

Term Rewriting with Logical Constraints*

Cynthia Kop[1] and Naoki Nishida[2]

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

[2] Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, 4648603 Nagoya, Japan
nishida@is.nagoya-u.ac.jp

Abstract. In recent works on program analysis, transformations of various programming languages to term rewriting are used. In this setting, *constraints* appear naturally. Several definitions which combine rewriting with logical constraints, or with separate rules for integer functions, have been proposed. This paper seeks to unify and generalise these proposals.

Key words: Term rewriting, Constraints, Integer rewriting

1 Introduction

Given the prevalence of computer programs in modern society, an important role is reserved for program analysis. Such analysis could take the form of for instance *termination* ("will this program end eventually, regardless of user input?"), *productivity* ("will this program stay responsive during its run?") and *equivalence* ("will this optimised code return the same result as the original?").

In recent years, there have been several results which transform a real-world program analysis problem into a query on term rewriting systems (TRSs). Such transformations are used to analyse termination of small, constructed languages (e.g. [3]), but also real code, like Java Bytecode [13], Haskell [10], LLVM [5], or Prolog [15]. Similar transformations are used to analyse code equivalence in [4,9].

In these works, *constraints* arise naturally. Where traditional TRSs generally consider well-founded sets like the natural numbers, more dedicated techniques are necessary when dealing with for instance integers or floating point numbers. Unfortunately, standard techniques for analysing TRSs are not equipped to also handle constraints. While integers and constraints *can* be encoded in TRSs, the results are either hairy or infinite, and generally hard to handle.

For this reason, rewriting with native support for logical constraints over a model was proposed [9]. While the results from normal term rewriting do not immediately apply in this setting, the *ideas* extend easily, so dedicated results are derived without much effort. Thus, constrained TRSs give a useful abstraction layer for program analysis. Several alternative definitions of constrained rewriting, focused on *integer constraints*, have also been given, see e.g. [4,5,8].

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

Unfortunately, the various formalisms are incompatible; results from one style of constrained rewriting do not necessarily transfer to another. This is a shame, as e.g. the lemma generation method in [12] (there used to prove equivalence of C-functions) might otherwise be reused in termination proofs. Also, dependency pairs and graphs are introduced in each of [3,8,14]. Thus, a lot of time is spent on redoing the same work for slightly different settings. Moreover, there are things we cannot do easily with any of them, such as overflow-conscious analysis.

In this paper, we propose a new formalism which unifies existing definitions of constrained rewriting. This formalism seeks to be *general*: unlike most of its predecessors, we do not limit interest to the integers, since in the future we will likely want to analyse programs which involve for instance real numbers or bitvectors. Moreover, we do not restrict attention to one kind of analysis (e.g. only termination or function equivalence). This way, we may for instance use the same dependency pair framework both to analyse termination of Haskell, and as part of a proof that two Java programs produce the same result (as termination is an essential property in inductive equivalence proofs, see e.g. [4,12]).

Paper Setup. This paper is structured as follows. In Section 2 we consider some preliminaries: both mathematical notions and a definition of many-sorted term rewriting. In Sections 3 and 4 we introduce the *LCTRS* formalism, which is the main contribution of this work. In Section 5 we will study how LCTRSs relate to existing definitions. Finally, to demonstrate how existing analysis techniques extend, we will consider basic confluence and termination results in Section 6.

2 Preliminaries

2.1 Sets and Functions

We assume that the mathematical notion of a *set* is well-understood.

The *function space* from a set A to a set B, denoted $A \Longrightarrow B$, consists of all sets f of pairs $\langle a, b\rangle$ with $a \in A$ and $b \in B$, such that for all $a \in A$ there is a unique $b \in B$ with $\langle a, b\rangle \in f$. The $\Longrightarrow$ is considered right-associative, so $A \Longrightarrow B \Longrightarrow C$ is the function space $A \Longrightarrow (B \Longrightarrow C)$. We use functional notation: for $f \in A_1 \Longrightarrow \cdots \Longrightarrow A_n \Longrightarrow B$, $f(a_1, \ldots, a_n)$ denotes the unique b with $\langle a_1, \langle a_2, \ldots \langle a_n, b\rangle \ldots\rangle \in f$. When dealing with constraints, we need a notion of truth. To this end, we will often use the set $\mathbb{B} = \{\top, \bot\}$ of *booleans*.

Example 1. We consider the set $\mathbb{N}$ of natural numbers and the set $\mathbb{R}$ of real numbers. An example element of the function space $\mathbb{R} \Longrightarrow \mathbb{R} \Longrightarrow \mathbb{N}$ is the function $\lambda x \in \mathbb{R}, y \in \mathbb{R}.\ abs(\lceil x + y\rceil - 9)$. Here, the λ notation denotes function construction. The comparison relation $>$ on natural numbers is an element of $\mathbb{N} \Longrightarrow \mathbb{N} \Longrightarrow \mathbb{B}$, and can also be denoted in extended form, $\lambda x \in \mathbb{N}, y \in \mathbb{N}.\ x > y$.

2.2 Many-Sorted Term Rewriting Systems

Next, we consider term rewriting. In constrained rewriting, *types* like integers and booleans appear naturally. Since we have, at present, little reason to introduce function types, let us consider *many-sorted TRSs*.

Sorts and Signature. We assume given a set $\mathcal{S}$ of *sorts* and a set $\mathcal{V}$ of *variables.* A *signature* Σ is a set of *function symbols* f, each equipped with a *sort declaration* of the form $[\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa$ with all ι_i and κ sorts. A *variable environment* is a set Γ of variable : sort pairs.[3]

Terms. Fixing a signature Σ, a *term* is any expression s built from function symbols in Σ, variables, commas and parentheses, such that $\Gamma \vdash s : \iota$ can be derived for some environment Γ and sort ι, using the following inference rules:

$$\frac{}{\Gamma \cup \{x : \iota\} \vdash x : \iota} \qquad \frac{\Gamma \vdash s_1 : \iota_1 \quad \ldots \quad \Gamma \vdash s_n : \iota_n \quad f : [\iota_1 \times \ldots \times \iota_n] \Rightarrow \kappa \in \Sigma}{\Gamma \vdash f(s_1, \ldots, s_n) : \kappa}$$

For any non-variable term s, there is a unique sort ι such that $\Gamma \vdash s : \iota$ for some Γ; we say that ι is the sort of s. The set of terms over Σ and $\mathcal{V}$ is denoted $\mathcal{T}erms(\Sigma, \mathcal{V})$. $Var(s)$ is the set of variables in s. A term s is *ground* if $Var(s) = \emptyset$.

Contexts and Substitution. Fixing an environment Γ, a *substitution* is a mapping $[x_1 := s_1, \ldots, x_k := s_k]$ from variables to terms, with $\{x_1 : \iota_1, \ldots, x_k : \iota_k\} \subseteq \Gamma$ and where $\Gamma \vdash s_1 : \iota_1, \ldots, \Gamma \vdash s_k : \iota_k$. The result $s\gamma$ of applying a substitution γ on a term s, is s with all occurrences of any x_i replaced by s_i. A *context* C is a term with zero or more special variables: $\Box_1, \ldots, \Box_n$, each occurring once. If $\Gamma \cup \{\Box_1 : \iota_1, \ldots, \Box_n : \iota_n\} \vdash C : \kappa$, and also $\Gamma \vdash s_i : \iota_i$ for all i, then we define the term $C[s_1, \ldots, s_n]$ as C with each $\Box_i$ replaced by the corresponding s_i.

Rules and Rewriting. In a many-sorted TRS (without constraints!) rules are pairs $l \to r$ where l and r are terms, l is not a variable and $Var(r) \subseteq Var(l)$; moreover, l and r must have the same sort. A (finite or infinite) set of rules $\mathcal{R}$ induces a rewrite relation $\to_{\mathcal{R}}$ on the set of terms by the following inference rule:

$$C[l\gamma] \to_{\mathcal{R}} C[r\gamma] \text{ for all rules } l \to r, \text{ contexts } C \text{ and substitutions } \gamma.$$

$\to_{\mathcal{R}}^{+}$ denotes the transitive closure of $\to_{\mathcal{R}}$, and $\to_{\mathcal{R}}^{*}$ the reflexive-transitive one.

Example 2. We consider a many-sorted TRS, with signature and rules as follows:

$$\begin{array}{lll} \mathsf{0} : \mathsf{int} & \mathsf{plus} : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{int} & \mathsf{geq2} : [\mathsf{int} \times \mathsf{int} \times \mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{bool} \\ \mathsf{s} : [\mathsf{int}] \Rightarrow \mathsf{int} & \mathsf{sum} : [\mathsf{int}] \Rightarrow \mathsf{int} & \mathsf{sum2} : [\mathsf{bool} \times \mathsf{int}] \Rightarrow \mathsf{int} \\ \mathsf{p} : [\mathsf{int}] \Rightarrow \mathsf{int} & \mathsf{geq} : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{bool} & \end{array}$$

$$\begin{array}{rlrl} \mathsf{sum}(x) & \to \mathsf{sum2}(\mathsf{geq}(\mathsf{0}, x), x) & \mathsf{geq}(x, y) & \to \mathsf{geq2}(x, y, \mathsf{0}, \mathsf{0}) \\ \mathsf{sum2}(\mathsf{true}, x) & \to \mathsf{0} & \mathsf{geq2}(\mathsf{s}(x), y, z, u) & \to \mathsf{geq2}(x, y, \mathsf{s}(z), u) \\ \mathsf{sum2}(\mathsf{false}, \mathsf{s}(x)) & \to \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) & \mathsf{geq2}(\mathsf{p}(x), y, z, u) & \to \mathsf{geq2}(x, y, z, \mathsf{s}(u)) \\ \mathsf{plus}(\mathsf{s}(x), y) & \to \mathsf{s}(\mathsf{plus}(x, y)) & \mathsf{geq2}(\mathsf{0}, \mathsf{s}(x), y, z) & \to \mathsf{geq2}(\mathsf{0}, x, y, \mathsf{s}(z)) \\ \mathsf{plus}(\mathsf{p}(x), y) & \to \mathsf{p}(\mathsf{plus}(x, y)) & \mathsf{geq2}(\mathsf{0}, \mathsf{p}(x), y, z) & \to \mathsf{geq2}(\mathsf{0}, x, \mathsf{s}(y), z) \\ \mathsf{plus}(\mathsf{0}, y) & \to y & \mathsf{geq2}(\mathsf{0}, \mathsf{0}, \mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{geq2}(\mathsf{0}, \mathsf{0}, x, y) \\ \mathsf{s}(\mathsf{p}(x)) & \to x & \mathsf{geq2}(\mathsf{0}, \mathsf{0}, x, \mathsf{0}) & \to \mathsf{true} \\ \mathsf{p}(\mathsf{s}(x)) & \to x & \mathsf{geq2}(\mathsf{0}, \mathsf{0}, \mathsf{0}, \mathsf{s}(x)) & \to \mathsf{false} \end{array}$$

[3] In some sources variables in $\mathcal{V}$ are immediately equipped with a sort. This choice leads to very similar definitions and results.

Here, $\mathsf{sum}(n)$ calculates $\Sigma_{i=1}^{n} i$. Because we consider integers, rather than the (well-founded) natural numbers, this is a somewhat tricky system for common analysis methods. A term $\mathsf{sum}(\mathsf{s}(0))$ is reduced to $\mathsf{s}(0)$ in 11 steps.

Example 3. We might simplify Example 2 by considering an infinite signature, which contains all integers, and an infinite set of rules, as is (roughly) done in [8]:

$$
\begin{array}{rll}
\mathsf{sum}(x) & \to \mathsf{sum2}(\mathsf{geq}(0,x),x) & \\
\mathsf{sum2}(\mathsf{true},x) & \to 0 & \\
\mathsf{sum2}(\mathsf{false},x) & \to \mathsf{plus}(x,\mathsf{sum}(\mathsf{plus}(x,-1))) & \\
\mathsf{plus}(\mathsf{n},\mathsf{m}) & \to \mathsf{k} & \forall n,m,k \in \mathbb{Z} \text{ such that } n+m=k \\
\mathsf{geq}(\mathsf{n},\mathsf{m}) & \to \mathsf{true} & \forall n,m \in \mathbb{Z} \text{ such that } n \geq m \\
\mathsf{geq}(\mathsf{n},\mathsf{m}) & \to \mathsf{false} & \forall n,m \in \mathbb{Z} \text{ such that } n < m
\end{array}
$$

This system is more pleasant, but has infinitely many rules, which makes it awkward to deal with except for dedicated techniques. Also, we still have to encode the constraints in the rules (and add rules to evaluate them), which makes analysis tricky. For example, termination of $\mathsf{sum}(x)$ relies on x getting closer to 0 in every step; to prove this, we must track the implications of $\mathsf{geq}(0,x) \to_{\mathcal{R}}^{*} \mathsf{false}$.

Note: term rewriting is usually defined without sorts. Then, function symbols have an *arity* (number of arguments) rather than a sort declaration. Such a TRS can be seen as a many-sorted TRS by assigning to symbols with arity n a sort declaration $[\mathsf{term} \times \ldots \times \mathsf{term}] \Rightarrow \mathsf{term}$, with n occurrences of term before the $\Rightarrow$.

3 Term Rewriting with Logical Constraints

Examples 2 and 3 illustrate why rewriting with native support for integer operations and constraints is a good idea. Normal rewriting simply does not seem adequate when handling data types which are not usually defined inductively.

We could add integers and integer constraints to rewriting, as in [3]. But with equal effort, we may be more general. Rather than focusing on $\mathbb{Z}$, we follow the ideas of [9] and take the underlying domain, and operations on it, as parameters.

Terms. We assume given a signature $\Sigma = \Sigma_{terms} \cup \Sigma_{theory}$. *Terms* are elements of $\mathcal{T}erms(\Sigma,\mathcal{V})$ as in Section 2.2. Moreover, we assume given a mapping $\mathcal{I}$ which assigns to each sort occurring in Σ_{theory} a set, and an *interpretation mapping* $\mathcal{J}$ 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 $\mathcal{I}_\iota$.

Let $\mathcal{V}al$ be the set of all values. We generally identify a value c with the logical term $c()$. An interpretation mapping can be extended to an interpretation on ground terms in $\mathcal{T}erms(\Sigma_{theory},\mathcal{V})$ in the obvious way:

$$[\![f(s_1,\ldots,s_n)]\!]_{\mathcal{J}} = \mathcal{J}_f([\![s_1]\!]_{\mathcal{J}},\ldots,[\![s_n]\!]_{\mathcal{J}})$$

The elements of Σ_{theory} and Σ_{terms} may overlap only on values ($\Sigma_{theory} \cap \Sigma_{terms} \subseteq \mathcal{V}al$). We call a term in $\mathcal{T}erms(\Sigma_{theory}, \mathcal{V})$ a *logical term*, and a term in $\mathcal{T}erms(\Sigma_{terms}, \mathcal{V})$ a *proper term*. Intuitively, logical terms define a function or constraint in the model, while proper terms are the objects we want to rewrite. Mixed terms typically occur as intermediate steps in a reduction.

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 *logical constraint* is a logical term of some sort bool with $\mathcal{I}_{\mathsf{bool}} = \mathbb{B}$. We generally let $\mathcal{V}al_{\mathsf{bool}} = \{\mathsf{true}, \mathsf{false}\}$. A ground logical constraint s is *valid* if $[\![s]\!]_{\mathcal{J}} = \top$. A non-ground constraint s is valid if $s\gamma$ is valid for all substitutions γ which map the variables in $Var(s)$ to a value.

Example 4. Choosing $\mathcal{I}_{\mathsf{int}} = \mathbb{Z}$ and $\mathcal{I}_{\mathsf{bool}} = \mathbb{B}$, we might let Σ_{theory} be the set below, with interpretations as given in e.g. SMTLIB [1]:

$$\begin{array}{rll} \mathsf{true} : \mathsf{bool} & + : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{int} & \wedge : [\mathsf{bool} \times \mathsf{bool}] \Rightarrow \mathsf{bool} \\ \mathsf{false} : \mathsf{bool} & \geq : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{bool} & \neg : [\mathsf{bool}] \Rightarrow \mathsf{bool} \\ \mathsf{n} : \mathsf{int}\ (n \in \mathbb{Z}) & = : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{bool} & \end{array}$$

Here, we would for instance define $\mathcal{J}(\mathsf{s}) = \lambda n.n + 1$, and $\mathcal{J}(\geq) =$ "the greater than or equal function". Moreover, we let Σ_{terms} consist of:

$$\mathsf{sum} : [\mathsf{int}] \Rightarrow \mathsf{int} \qquad \mathsf{n} : \mathsf{int}\ (\forall n \in \mathbb{Z})$$

The values in Σ are true, false, and n for all $n \in \mathbb{Z}$. Examples of logical terms, considering $\geq$, $+$ and $=$ as infix symbols, are $\mathsf{0} = \mathsf{0} + -\mathsf{1}$ and $x + \mathsf{3} \geq y + -\mathsf{42}$. Both are constraints. $\mathsf{5} + \mathsf{9}$ is also a ground logical term, but is not a constraint. $\mathsf{sum}(x)$ and $\mathsf{sum}(\mathsf{sum}(\mathsf{42})$ are proper terms. The value $\mathsf{0}$ is both a proper and a logical term. $\mathsf{sum}(\mathsf{37} + \mathsf{5})$ is neither, but is still a term (also called *mixed term*).

In Example 4 we restricted interest to functions in SMTLIB for Σ_{theory}, but this is not fundamental; we might also for instance have a symbol $\mathsf{p} : [\mathsf{int}] \Rightarrow \mathsf{int}$ with $\mathcal{J}_{\mathsf{p}} = \lambda x.x - 1$, or $\mathsf{pi} : [\mathsf{int}] \Rightarrow \mathsf{int}$ with $\mathcal{J}_{\mathsf{pi}} = \lambda n.$"the n^{th} decimal of π". It is in general a good idea, however, to limit interest to computable functions.

Rules and Rewriting. A *rule* is a triple $l \to r\ [\varphi]$ where l and r are terms, and φ is a logical constraint. l must have the form $f(l_1, \dots, l_n)$ with $f \in \Sigma_{terms} \setminus \Sigma_{theory}$, and l and r must have the same sort. If $\varphi = \mathsf{true}$ with $\mathcal{J}(\mathsf{true}) = \top$, the rule is usually just denoted $l \to r$. A rule is *regular* if $Var(\varphi) \subseteq Var(l)$ and *standard* if l is a proper term. We define $LVar(l \to r\ [\varphi]))$ as $Var(\varphi) \cup (Var(r) \setminus Var(l))$.

A substitution γ *respects* a rule $l \to r\ [\varphi]$ if:

1. $Dom(\gamma) = Var(l) \cup Var(r) \cup Var(\varphi)$;
2. $\gamma(x)$ is a value for all $x \in LVar(l \to r\ [\varphi])$;
3. $\varphi\gamma$ is valid.

We assume given a set of rules $\mathcal{R}$. The *rewrite relation* $\to_{\mathcal{R}}$ is a relation on terms, defined as the union of $\to_{\texttt{rule}}$ and $\to_{\texttt{calc}}$, where:

$$\begin{array}{rll} C[l\gamma] \to_{\texttt{rule}} C[r\gamma] & \text{if } l \to r\ [\varphi] \in \mathcal{R} \text{ and } \gamma \text{ respects } l \to r\ [\varphi] \\ C[f(s_1, \dots, s_n)] \to_{\texttt{calc}} C[v] & \text{if } f \in \Sigma_{theory} \setminus \Sigma_{terms}, \text{ all } s_i \text{ values,} \\ & \quad \text{and } v \text{ is the value of } f(s_1, \dots, s_n) \end{array}$$

A reduction step with $\rightarrow_{\texttt{calc}}$ is called a *calculation.* A term is in *normal form* if (and only if) it cannot be reduced with $\rightarrow_{\mathcal{R}}$. Sometimes we consider *innermost reduction*: $C[f(\boldsymbol{s})] \rightarrow_{\mathcal{R},in} C[t]$ if $f(\boldsymbol{s}) \rightarrow_{\mathcal{R}} t$, and all s_i are in normal form.

A *logical constrained term rewriting system* (LCTRS) is defined as the pair $(\mathcal{T}erms(\Sigma,\mathcal{V}), \rightarrow_{\mathcal{R}})$. An LCTRS is typically given by providing $\mathcal{I}$ and $\mathcal{J}$ and the sets $\Sigma_{terms}, \Sigma_{theory}$ and $\mathcal{R}$. When clear from context, the signatures and mappings may be omitted. An *innermost LCTRS* is the pair $(\mathcal{T}erms(\Sigma,\mathcal{V}), \rightarrow_{\mathcal{R},in})$.

A (normal or innermost) LCTRS is *standard* or *regular* if all its rules are standard or regular respectively. In a regular LCTRS, $\rightarrow_{\mathcal{R}}$ is computable, provided $\mathcal{J}_f$ is computable for all $f \in \Sigma_{theory}$. Even in a regular LCTRS, the right-hand sides of rules may contain fresh variables. This can for example be used to simulate user input. Think for example of a rule $\mathsf{Start} \rightarrow \mathsf{Handle}(input)$ where *input* is a variable; by definition of $\rightarrow_{\mathcal{R}}$, *input* can only be instantiated by a value.

Example 5. It is time to see how these definitions work in practice. Let us modify Example 2 to use constraints and calculations. We have defined Σ_{theory} and Σ_{terms} in Example 4. The rules are replaced by the following set:

$$\mathsf{sum}(x) \rightarrow \mathsf{0}\ [\mathsf{0} \geq x] \qquad \mathsf{sum}(x) \rightarrow x + \mathsf{sum}(x + -1) \quad [\neg(\mathsf{0} \geq x)]$$

Note that the sum rules may only be applied to a term $\mathsf{sum}(n)$ whose immediate argument n is a value, so this subterm itself cannot contain the symbol sum.

For an example derivation, let us calculate $\Sigma_{n=1}^{2} n$. We have: $\mathsf{sum}(\mathsf{2}) \rightarrow_{\texttt{rule}} \mathsf{2} + \mathsf{sum}(\mathsf{2} + -\mathsf{1}) \rightarrow_{\texttt{calc}} \mathsf{2} + \mathsf{sum}(\mathsf{1}) \rightarrow_{\texttt{rule}} \mathsf{2} + (\mathsf{1} + \mathsf{sum}(\mathsf{1} + -\mathsf{1})) \rightarrow_{\texttt{calc}} \mathsf{2} + (\mathsf{1} + \mathsf{sum}(\mathsf{0})) \rightarrow_{\texttt{rule}} \mathsf{2}+(\mathsf{1}+\mathsf{0}) \rightarrow_{\texttt{calc}} \mathsf{2}+\mathsf{1} \rightarrow_{\texttt{calc}} \mathsf{3}$. In each step, it so happens that we have exactly one choice of what rule to apply, and where. For example in the first step, $\neg(\mathsf{0} \geq \mathsf{2})$ holds and $\mathsf{0} \geq \mathsf{2}$ does not, so only the second rule is applicable. Neither rule can be applied on $\mathsf{sum}(\mathsf{2} + -\mathsf{1})$, as $\mathsf{2} + -\mathsf{1}$ is no value; $\rightarrow_{\texttt{calc}}$ *is* applicable. We cannot use a calculation step on $\mathsf{2} + (\mathsf{1} + \mathsf{sum}(\mathsf{0}))$, as there is no subterm with the right form; the system does not know about associativity.

One might wonder why we insist that all variables in *LVar* are instantiated with values, rather than just logical terms. Having a rule $f(x,y) \rightarrow y\ [x \geq y]$, could we not reduce $f(x + \mathsf{1}, x)$ without this instantiation?

The reason to require that $\gamma(x)$ is ground for $x \in Var(\varphi)$ is simplicity of the rewrite relation: by posing this restriction, validity of $\varphi\gamma$ is mostly easy to test. Without it, validity might not be computable.[4] Moreover, without this restriction the reduction relation is not preserved under substitution: for a symbol $\mathsf{a} \in \Sigma_{terms} \setminus \Sigma_{theory}$, we cannot have $f(\mathsf{a} + \mathsf{1}, \mathsf{a}) \rightarrow \mathsf{a}$, as $\mathsf{a} + \mathsf{1}$ is not a logical term.

It does make sense to study whether a term $f(x+\mathsf{1}, y)$, or even a term $f(x, y)$ with $x > y$ reduces. In Section 4 we will see how to rewrite *constrained terms.*

By requiring that $\gamma(x)$ is even a value, we avoid complicating notions like complexity. If we could $\rightarrow_{\texttt{rule}}$-reduce terms like $\mathsf{sum}(\mathsf{3}+-\mathsf{1}+-\mathsf{1}+-\mathsf{1})$, then these

[4] For example, defining $\mathcal{J}_p(n)$ to be true if a sequence of 9999 nines starts at the n^{th} decimal of π, and false otherwise, and considering a rule $f(x) \rightarrow x\ [\neg p(x)]$, we don't know whether a term $f(x)$ should reduce for all instances of x.

additions would have to be calculated in every step when testing the constraint. There does not seem to be any advantage to allowing these hidden calculations; we can simply $\rightarrow_{\texttt{calc}}$-normalise ground logical terms before applying a rule step.

For the same motivation of complexity, we only allow $\rightarrow_{\texttt{calc}}$ to take single steps, rather than allowing $C[s] \rightarrow_{\texttt{calc}} C[v]$ for any logical term s with value v.

Note that $\rightarrow_{\texttt{calc}}$ is not special; using $\rightarrow_{\texttt{calc}}$ is functionally equivalent to extending $\mathcal{R}$ with all rules $f(x_1, \dots, x_n) \rightarrow y\ [f(x_1, \dots, x_n) = y]$ where $f \in \Sigma_{theory}$.

Regularity and Standardness. Regularity is a useful condition. An irregular LCTRS is not in general deterministic, polynomially solvable, or even computable. Consider for example a rule $\mathsf{f}(x) \rightarrow \mathsf{g}(\mathsf{f}(y), \mathsf{f}(z))\ [x = y{*}z{\wedge}y > 1{\wedge}z > 1]$, which quickly decomposes a natural number into its prime factors.

Still, there is some advantage in allowing irregular systems. For example, in termination analysis a transformation might chain the two regular rules

$$\begin{array}{rll} f(x) \rightarrow g(x-1, \mathit{input}) & [x > 0] \\ g(x,y) \rightarrow f(y) & [x \geq y] \end{array}$$

into a single irregular rule:

$$f(x) \rightarrow f(y)\ [x > 0 \wedge x - 1 \geq y]$$

In addition, an irregular rule can be used to calculate a partial function, e.g. $\mathsf{div}(x, y) \rightarrow z\ [z * y = x]$, which cannot be easily defined otherwise. Note that such a rule does not lead to undecidability.

Similar to regularity, *standardness* is convenient: in a standard system there are no overlaps between $\rightarrow_{\texttt{rule}}$ and $\rightarrow_{\texttt{calc}}$. Standardness is a natural property, since symbols from $\Sigma_{theory} \setminus \Sigma_{terms}$ in terms are conceptually intended primarily as a way to do calculations. We don't use modulo reasoning: a non-standard rule $f(x+1) \rightarrow r$ matches only terms of the form $f(s+1)$, so *not* for instance $f(3)$. As with regularity, non-standard systems may arise during analysis, for example when a rule is reversed. Note that even in standard systems, left-hand sides of rules may contain values (or other symbols in Σ_{terms}); while we often encounter rules of the form $f(x_1, \dots, x_n) \rightarrow r\ [\varphi]$, this is not an innate property.

Overview. Compared to the rather hairy (Example 2) or infinite (Example 3) systems obtained when encoding integer arithmetic and constraints in a normal TRS, LCTRSs offer an elegant alternative. Although LCTRSs often have infinite signatures, calculation steps make it possible to avoid infinite sets of rules.

Example 6. To demonstrate a situation where we should not use the integers as an underlying set, consider the following short imperative program:

```
1. function main() {              5.     x = x * 2;
2.   byte x = input();            6.   }
3.   while (x < 150) {            7.   return x;
4.     if (x == 0) x = 1;         8. }
```

Here a `byte` is an unsigned 8-bit integer. This program doesn't terminate: input 0 gives an infinite reduction (with `x` changing from 0 to 1, 2, 4, 8, 16, 32, 64, 128, 0).

Using the ideas of [3], we might model this program as follows:

$$\begin{array}{lll} \mathsf{main} \to \mathsf{loop}(\mathit{input}) & & \mathsf{loop}_1(x) \to \mathsf{loop}_2(1) \quad [x = 0] \\ \mathsf{loop}(x) \to \mathsf{loop}_1(x) & [x < 150] & \mathsf{loop}_1(x) \to \mathsf{loop}_2(x) \quad [\neg(x = 0)] \\ \mathsf{loop}(x) \to \mathsf{return}(x) & [\neg(x < 150)] & \mathsf{loop}_2(x) \to \mathsf{loop}(x * 2) \end{array}$$

As bytes are not unbounded, our underlying set is *not* $\mathbb{Z}$. Rather, we consider the set $\mathcal{BV}_8$ of bit vectors of length 8, and let $\mathcal{J}$ map to corresponding notions of addition, multiplication, comparison and equality (see SMT-LIB [1]). With this theory, a reduction from start adequately simulates a reduction in the original imperative program. Note that if we had naively translated the program to an LCTRS over the integers, we could have falsely concluded (local) termination.

We can analyse systems on bitvectors in much the same way as we analyse systems over the integers. We will see some ideas for this in Section 6.

4 Constrained Terms

As discussed in Section 3, there are good reasons why the definition of rewriting requires that variables in a constraint are instantiated by values. But sometimes you may want to know whether a term of a certain form rewrites. For example, if we know that $x < -3$, this is enough to decide that $\mathsf{sum}(x)$ reduces to $\mathsf{0}$.

In this section, we will therefore consider *constrained terms*: pairs $s\ [\varphi]$ of a term s and a constraint φ. Constrained terms are harder to rewrite and analyse than normal terms, but sometimes the need may arise. For instance in rewriting induction (see e.g. [4,9]) when proving that $f(x) \leftrightarrow^* g(x, \mathsf{0})\ [x \geq \mathsf{1}]$, being able to reason about the reducts of $f(x)\ [x \geq \mathsf{1}]$ is very relevant.

To rewrite constrained terms, we must take several things into account. For example, given a rule $f(\mathsf{0}) \to \mathsf{1}$, we should be able to reduce a constrained term $f(x)\ [x = \mathsf{0}]$, even though $f(x)$ itself is not matched by the left-hand side. We will also need to deal with irregular rules; given a rule $f(x) \to g(y)\ [y > x]$, we should be able to reduce a constrained term $f(x)\ [x > \mathsf{3}]$ to $g(y)\ [y > \mathsf{4}]$, or at least to an instance, like $g(y)\ [x > \mathsf{3} \wedge y = x + \mathsf{1}]$.

To begin, let us consider how to compare constrained terms. A substitution γ *respects* a constrained term $s\ [\varphi]$ if $\gamma(x)$ is a value for all $x \in \mathit{Var}(\varphi)$ and $[\![\varphi\gamma]\!] = \top$. Two constrained terms $s\ [\varphi]$ and $t\ [\psi]$ are *equivalent*, notation $s\ [\varphi] \approx t\ [\psi]$, if for all substitutions γ which respect $s\ [\varphi]$ there is a substitution δ which respects $t\ [\psi]$ such that $s\gamma = t\delta$, and for all δ which respect $t\ [\psi]$ there is a substitution γ which respects $s\ [\varphi]$ such that $s\gamma = t\delta$.

Example 7. Examples of constrained terms over the signature of sum are:

1. $\mathsf{sum}(x)\ [x \geq \mathsf{3}]$;
2. $x + y\ [x \geq y \wedge \neg(x = y) \wedge x = \mathsf{3}]$;
3. $\mathsf{3} + z\ [\mathsf{1} \geq x \wedge x + \mathsf{1} \geq z]$;

Constrained terms 2 and 3 are equivalent, as the following formulas hold in $\mathbb{Z}$:

$$\forall x, y.(x \geq y \wedge \neg(x = y) \wedge x = 3) \Rightarrow \exists x', z.1 \geq x' \wedge x' + 1 \geq z \wedge x = 3 \wedge y = z$$
$$\forall x', z.(1 \geq x' \wedge x' + 1 \geq z) \Rightarrow \exists x, y.x \geq y \wedge \neg(x = y) \wedge x = 3 \wedge y = z$$

It is clear that equivalence of two constrained terms is not always easy to tell.

To be able to modify constraints, we assume that Σ_{theory} contains a symbol $\wedge : [\mathsf{bool} \times \mathsf{bool}] \Rightarrow \mathsf{bool}$ with $\mathcal{J}_\wedge$ the conjunction operator on the booleans, and for all sorts ι a symbol $=_\iota$: $[\iota \times \iota] \Rightarrow \mathsf{bool}$ with $\mathcal{J}_{=_\iota} := \lambda n, m \in \mathcal{I}_\iota . n = m$.

Rewriting Constrained Terms. We let $s\ [\varphi] \to_{\mathtt{calc}} t\ [\varphi \wedge x = f(s_1, \ldots, s_n)]$ if $s = C[f(s_1, \ldots, s_n)]$ with $f \in \Sigma_{theory} \setminus \Sigma_{terms}$, all s_i in $Var(\varphi) \cup \mathcal{V}al$, x a fresh variable, and $t = C[x]$. Additionally, $s\ [\varphi] \to_{\mathtt{rule}} t\ [\varphi]$ if φ is satisfiable, $s = C[l\gamma]$ and $t = C[r\gamma]$ for some rule $l \to r\ [\psi]$ and substitution γ such that:

- $Dom(\gamma) = Var(l) \cup Var(r) \cup Var(\psi)$
- $\gamma(x)$ is a value or variable in $Var(\varphi)$ for all $x \in LVar(l \to r\ [\psi])$
- $\varphi \Rightarrow (\psi\gamma)$ is valid (that is, for all δ with $\varphi\delta$ valid also $\psi\gamma\delta$ is valid)

The relation $\to_\mathcal{R}$ on constrained terms is defined as $\approx \cdot (\to_{\mathtt{calc}} \cup \to_{\mathtt{rule}}) \cdot \approx$.

Example 8. With the rule $f(\mathsf{0}) \to \mathsf{1}$: $f(x)\ [x = \mathsf{0}] \approx f(\mathsf{0})\ [\mathsf{true}] \to_{\mathtt{rule}} \mathsf{1}\ [\mathsf{true}]$.

Example 9. With the irregular rule $f(x) \to g(y)\ [y > x]$, we have: $f(x)\ [x > \mathsf{3}] \approx f(x)\ [x > \mathsf{3} \wedge y > x] \to_{\mathtt{rule}} g(y)\ [x > \mathsf{3} \wedge y > x] \approx g(y)\ [y > \mathsf{4}]$. Similarly, $f(x)\ [x > \mathsf{0}]$ reduces with $f(x) \to g(y)\ [x = y + \mathsf{1}]$ to $f(y)\ [y \geq \mathsf{0}]$. We do *not* have that $f(x)\ [\mathsf{true}] \to g(x - \mathsf{1})\ [\mathsf{true}]$, as $x - \mathsf{1}$ cannot be instantiated to a value.

Example 10. Following on Example 5, we may reduce $\mathsf{sum}(x)\ [x > \mathsf{2}]$ as follows:

$$\begin{aligned} \mathsf{sum}(x)\ [x > \mathsf{2}] &\to_\mathcal{R} x + \mathsf{sum}(x + -\mathsf{1})\ [x > \mathsf{2}] \\ &\to_\mathcal{R} x + \mathsf{sum}(y)\ [x > \mathsf{2} \wedge y = x + -\mathsf{1}] \\ &\to_\mathcal{R} x + (y + \mathsf{sum}(y + -\mathsf{1}))\ [x > \mathsf{2} \wedge y = x + -\mathsf{1}] \\ &\to_\mathcal{R} x + (y + \mathsf{sum}(z))\ [x > \mathsf{2} \wedge y = x + -\mathsf{1} \wedge z = y + -\mathsf{1}] \end{aligned}$$

The notion of reduction on constrained terms is intimately tied to the notion of reduction on terms, as the following two theorems demonstrate:

Theorem 1. *If $s \to_\mathcal{R} t$ then also $s\ [\mathsf{true}] \to_\mathcal{R} t\ [\mathsf{true}]$.*

Theorem 2. *If $s\ [\varphi] \to_\mathcal{R} t\ [\psi]$ then for all substitutions γ which respect $s\ [\varphi]$ there is a substitution δ which respects $t\ [\psi]$ such that $s\gamma \to_\mathcal{R} t\delta$.*

Thus, we have a notion of constrained terms and reduction thereof. We do not consider these notions as basic; rather, using for instance Theorem 2, they can be used in analysis to find properties of unconstrained terms in the system (think for instance of an inductive proof that $\mathsf{sum}(x) \geq x$ if $x \geq \mathsf{0}$).

Determining whether a constrained term reduces, or what it reduces to, is a difficult problem. In special cases (for instance, regular rules with linear integer arithmetic) it is decidable, but in others we may have to resort to clever guessing.

5 Comparison to Existing Systems

So, we have a new formalism. Is this truly more convenient or general than existing formalisms? Here we will briefly study some formalisms from the literature, and sketch how they relate to the LCTRSs introduced here. However, a comprehensive study of all relevant formalisms is beyond the reach of this paper.

5.1 Constrained TRSs from [9,12,14]

LCTRSs are based primarily on the constrained TRSs in [9,12,14]. Like our LCTRSs, these systems have a separate theory signature, and a given interpretation mapping $\mathcal{J}$. The main difference is that they have no values or calculation steps. Instead, these features are encoded in the terms and rules, with for example the integers being represented as $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \dots, \mathsf{p}(\mathsf{0}), \mathsf{p}(\mathsf{p}(\mathsf{0})), \dots$

Example 11. The sum system is implemented as a CTRS with rules:

$$\begin{array}{llll} \mathsf{sum}(x) \to \mathsf{0} & [\mathsf{0} \geq x] & \mathsf{0} + y \to y & \mathsf{s}(\mathsf{p}(x)) \to x \\ \mathsf{sum}(\mathsf{s}(x)) \to \mathsf{sum}(x) + \mathsf{s}(x) & [x \geq \mathsf{0}] & \mathsf{s}(x) + y \to \mathsf{s}(x + y) & \mathsf{p}(\mathsf{s}(x)) \to x \\ & & \mathsf{p}(x) + y \to \mathsf{p}(x + y) & \end{array}$$

Compared to their predecessors, LCTRS are far simpler to use: by having symbols for all values and using calculation steps, systems are implemented much more concisely (as demonstrated by sum). Moreover, the resulting systems are easier to analyse. For example, note that this version of sum is not a constructor system, and proving confluence or complete reducibility is difficult. Also, due to the countable nature of terms, no finite CTRS can encode Example 12:

Example 12. Using sorts int, real and bool, and addition on the real numbers denoted by +., we might represent the function $n \mapsto \Sigma_{i=i}^{n} \sqrt{n}$ as follows:

$$\begin{array}{ll} \mathsf{sumroot}(x) \to \mathsf{0.0} & [\mathsf{0} \geq x] \\ \mathsf{sumroot}(x) \to \mathsf{sqrt}(x) +. \ \mathsf{sumroot}(x - 1) & [\neg(\mathsf{0} \geq x)] \end{array}$$

There does not seem to be an easy way to simulate CTRSs as LCTRSs or the other way around. However, initial results suggest that results for CTRS [9,12,14] easily extend to LCTRSs, and are moreover vastly simplified by the translation.

5.2 Integer Term Rewriting Systems

In Example 3 we saw a system somewhat like the *integer term rewriting systems* in [8]. These ITRSs are innermost TRSs with an infinite signature $\Sigma \cup \Sigma_{int}$, where Σ_{int} includes $BOp = \{+, -, *, /, \%, >, \geq, <, \leq, =, \neq, \wedge, \Rightarrow\}$ and moreover true, false and all integers. $\mathcal{R}$ is defined as $R \cup \mathcal{PD}$, where $\mathcal{PD} = \{n \circ m \to k \mid n, m, k \in \mathbb{Z} \cup \mathbb{B},\ \circ \in BOp \mid n \circ m = k \text{ holds in } \mathbb{Z} \text{ and } \mathbb{B}\}$ (e.g. $1 + 2 \to 3 \in \mathcal{PD}$).

Example 13. An example ITRS in [8] has $\Sigma = \{\mathsf{log}, \mathsf{lif}\}$ and R consisting of:

$$\begin{array}{ll} \mathsf{log}(x, y) \to \mathsf{lif}(x \geq y \wedge y > 1, x, y) & \mathsf{lif}(\mathsf{true}, x, y) \to 1 + \mathsf{log}(x/y, y) \\ & \mathsf{lif}(\mathsf{false}, x, y) \to \mathsf{0} \end{array}$$

Terms containing symbols $\geq$, $\wedge$, $+$ and $/$ can be rewritten using the $\mathcal{PD}$ rules.

We can model an ITRS as an LCTRS, which is finite if R is finite. ITRSs are not defined with sorts, but sorts can easily be imagined. Indeed, there seems little reason to analyse the behaviour of a term like $\mathsf{log}(\mathsf{true}, 5 + \mathsf{false})$, and for innermost termination (the primary area of interest for ITRSs) presence of sorts makes no difference [7]. The only other issue is that some elements of BOp ($/$ and $\%$) define *partial* functions, so cannot be modelled by calculations.

To define an LCTRS with the same terms and rewrite relation as a given *sorted* ITRS (with sorts int and bool assigned in the obvious way), let $\mathcal{V}al := \{\mathsf{true}, \mathsf{false} : \mathsf{bool}\} \cup \{\mathsf{n} : \mathsf{int} \mid n \in \mathbb{Z}\}$, $\Sigma_{terms} := \Sigma \cup \mathcal{V}al \cup \{/, \% : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{int}\}$ and $\Sigma_{theory} := \mathcal{V}al \cup (BOp \setminus \{/, \%\})$. We use the expected interpretations for Σ_{theory}. Let $\mathcal{R} := R \cup \{x/y \to z\ [x = y{*}z],\ x\%y \to z\ [x = y{*}u{+}z \wedge 0 \leq z \wedge z < y]\}$. Then $\to_{\mathcal{R}}$ is exactly the reduction relation from the original ITRS. $\mathcal{R}$ is finite, because $\to_{\mathtt{calc}}$ and the two irregular rules take over the role of $\to_{\mathcal{PD}}$. Note that the irregularity is not an issue for computability in this case.

Example 14. The system from Example 13 becomes the following LCTRS (ignoring the % symbol which does not occur in any rule):

$$\mathsf{log}(x, y) \to \mathsf{lif}(x \geq y \wedge y > \mathsf{1}, x, y) \qquad \mathsf{lif}(\mathsf{true}, x, y) \to \mathsf{1} + \mathsf{log}(x/y, y)$$
$$x/y \to z\ [x = y * z] \qquad \mathsf{lif}(\mathsf{false}, x, y) \to \mathsf{0}$$

Comment: if, for whatever reason, we do want to analyse the original *unsorted* ITRS, we can also do so with an LCTRS. In this case, we assign a single sort term as suggested in Section 2, and let $\mathcal{I}_{\mathsf{term}} = \mathbb{Z} \cup \mathbb{B}$; we cannot use calculations now, because all functions are partial, but can encode $\to_{\mathcal{PD}}$ with irregular rules.

Conditional ITRSs. Integer TRSs play a role in the termination analysis of Java Bytecode employed in [13]. There, termination of JBC is reduced to termination of a *conditional* ITRS; rules look somewhat like the rules in LCTRSs:

$$\mathsf{log}(x, y) \to 1 + \log(x/y, y) \mid x \geq y \wedge y > 1 \to^* \mathsf{true}$$
$$\mathsf{log}(x, y) \to \mathsf{0} \mid \neg(x \geq y \wedge y > 1) \to^* \mathsf{true}$$

These systems are unravelled to ITRSs for analysis (giving the system from Example 13). However, if the elements of Σ_{int} are not root symbols of left-hand sides of R, and / and % do not occur in the conditions, we can translate such systems into LCTRSs immediately (replacing conditions by constraints), and obtain a system where constraints are not encoded. We can prove that a CITRS generates the same relation as its transformation to an innermost LCTRS.

As for the other direction, LCTRSs are not a special case of ITRS. Most importantly, ITRSs have no native treatment of constraints. These have to be encoded, and to for instance prove termination of even simple systems we need far more powerful techniques than in the LCTRS setting. Moreover, ITRSs are restricted to the integers. While Example 6 *can* be encoded, using rules like $\mathsf{loop}_2(x) \to \mathsf{loop}((x * 2)\%256)$, ITRSs cannot represent for instance Example 12.

5.3 $\mathbb{Z}$-TRSs

Next, we consider a mixture of the ideas in [3,4,5,6].[5] $\mathbb{Z}$-TRSs are based on a many-sorted signature Σ, which must include $\Sigma_{\mathtt{int}} = \{0, 1 : \mathsf{int},\ - : [\mathsf{int}] \Rightarrow \mathsf{int},\ +, * : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{int}\}$. *Constraints* are given by the grammar:

$$\mathsf{C} ::= \mathsf{true} \mid \mathsf{false} \mid s = t \mid s > t \mid s \geq t \mid \neg \mathsf{C} \mid \mathsf{C} \wedge \mathsf{C} \qquad s, t \text{ terms over } \Sigma_{\mathtt{int}}$$

Validity of constraints is defined as expected. Rules are triples $l \to r\ [\varphi]$ where φ is a constraint, l and r are terms with the same sort, and l has the form $f(l_1, \dots, l_n)$. The left-hand sides may not contain any element of $\Sigma_{\mathtt{int}}$. The reduction relation is defined much like in LCTRSs, except that a substitution γ "respects" a $\mathbb{Z}$-TRS rule $l \to r\ [\varphi]$ if all variables of sort int are instantiated by (not necessarily ground) terms over $\Sigma_{\mathtt{int}}$, and $\varphi\gamma$ is valid. Note that symbols $2, 3, \dots$ are *not* included in the signature; terms have forms like $\mathsf{sum}((1+1)+1)$.

Example 15. In [3] we see how a code snippet is translated to a $\mathbb{Z}$-TRS:

```
while (x > 0 && y > 0) {
  if (x > y) { while (x > 0) { x--; y++; } }
  else { while (y > 0) { y--; x++; } }
}
```

$$\begin{array}{ll}
\mathsf{eval}_1(x,y) \to \mathsf{eval}_2(x,y) & [x > 0 \wedge y > 0 \wedge x > y] \\
\mathsf{eval}_1(x,y) \to \mathsf{eval}_3(x,y) & [x > 0 \wedge y > 0 \wedge \neg(x > y)] \\
\mathsf{eval}_2(x,y) \to \mathsf{eval}_2(x + -1, y + 1) & [x > 0] \\
\mathsf{eval}_2(x,y) \to \mathsf{eval}_1(x,y) & [\neg(x > 0)] \\
\mathsf{eval}_3(x,y) \to \mathsf{eval}_3(x + 1, y + -1) & [y > 0] \\
\mathsf{eval}_3(x,y) \to \mathsf{eval}_1(x,y) & [\neg(y > 0)]
\end{array}$$

In fact, $\mathbb{Z}$-TRSs are very close to our LCTRSs, but with a fixed theory signature. Although there is no concept of "values", there is no harm to internally replacing ground terms over $\Sigma_{\mathtt{int}}$ by the corresponding value (in a tool, or when manually rewriting terms), because the symbols in Σ_{int} do not occur in any left-hand side.

For every $\mathbb{Z}$-TRS, we can define an LCTRS which is roughly the same, modulo calculation of integer values. We can do this as follows; let:

$$\begin{array}{rl}
\Sigma_{theory} := & \{\mathsf{n} : \mathsf{int} \mid n \in \mathbb{Z}\} \\
& \cup \{\mathsf{true}, \mathsf{false} : \mathsf{bool},\ - : [\mathsf{int}] \Rightarrow \mathsf{int},\ +, * : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{int},\ =, >, \geq \\
& \quad : [\mathsf{int} \times \mathsf{int}] \Rightarrow \mathsf{bool},\ \neg : [\mathsf{bool}] \Rightarrow \mathsf{bool},\ \wedge : [\mathsf{bool} \times \mathsf{bool}] \Rightarrow \mathsf{bool}\} \\
\Sigma_{terms} := & (\Sigma \setminus \Sigma_{\mathtt{int}}) \cup \{\mathsf{n} : \mathsf{int} \mid n \in \mathbb{Z}\}
\end{array}$$

Every $\mathbb{Z}$-TRS rule is already a standard rule in this LCTRS, and every term in the original $\mathbb{Z}$-TRS is still a term.

Theorem 3. *We can derive, for all ground terms s, t:*

– *if $s \to_{\mathcal{R}} t$ in a $\mathbb{Z}$-TRS, and $s \to_{\mathtt{calc}}^* s' \in \mathcal{T}erms(\Sigma_{terms} \cup \Sigma_{theory}, \mathcal{V})$, then exists t' such that $t \to_{\mathtt{calc}}^* t'$ and $s' \to_{\mathcal{R}}^+ t'$ in the corresponding LCTRS;*

[5] The authors use simplifications of this formalism for different applications. For example, multiplication is omitted, or custom symbols must have output sort unit.

– *if* $s \to_{\texttt{rule}} t$ *in the corresponding LCTRS, and* $s' \in \mathcal{T}erms(\Sigma, \mathcal{V})$ *such that* $s' \to_{\texttt{calc}}^* s$*, then there is a term* t' *such that* $t' \to_{\texttt{calc}}^* t$ *and* $s' \to_{\mathcal{R}} t'$ *in the original* $\mathbb{Z}$*-TRS.*

Thus, results from LCTRSs typically extend to $\mathbb{Z}$-TRSs. As with ITRSs, $\mathbb{Z}$-TRSs cannot model the behaviour of LCTRSs. Being fundamentally restricted to the integers, they cannot easily represent Example 6, nor Example 12. An extension of $\mathbb{Z}$-TRSs, which admits all integers in the signature, can model a variation of Example 6, as discussed in [6]. This analysis uses extra rules to "normalise" integers to their range, e.g.

$$\begin{array}{ll} \mathsf{loop}_2(x) \to \mathsf{loop}_3(x * 2) & \\ \mathsf{loop}_3(x) \to \mathsf{loop}_3(x - 256) & [x \geq 256] \\ \mathsf{loop}_3(x) \to \mathsf{loop}(x) & [x \geq 0 \wedge 256 \geq x] \end{array}$$

5.4 Constrained Equational Systems

For a very different direction, let us consider a system from the further past. In [11], a framework for constrained deduction is developed, which uses constrained terms and rules. Like the current paper, the interpretation of function symbols ($\mathcal{J}_f$) is not fixed, but assumed to be given by the user. There is no notion of values, however. This fits with a typical usage of the formalism, where the underlying model is the set of terms modulo some theory.

Example 16. We consider a constrained system with symbols $* : [\mathsf{term} \times \mathsf{term}] \Rightarrow \mathsf{term}$, $\mathsf{a}, \mathsf{b} : \mathsf{term}$, $=_{\mathsf{AC}}$: $[\mathsf{term} \times \mathsf{term}] \Rightarrow \mathsf{bool}$. The model $\mathcal{I}_{\mathsf{term}}$ is the set of terms over $\{*, \mathsf{a}, \mathsf{b}\}$, where a, b and $*$ are interpreted as themselves and $=_{\mathsf{AC}}$ is interpreted as equality on terms modulo AC (associativity and commutativity) of $*$.

Unlike LCTRSs, this formalism has no separate "term signature": all function symbols have a meaning in the model, and may occur in both terms and constraints. Rules have the form $s \to t$ $[\varphi]$ and are used for example to simplify constrained terms (called *constrained formulas*) modulo an equational theory.

Example 17. In the signature from Example 16, we consider a rule $(x * x) * x \to \mathsf{a}$ $[\neg(x =_{\mathsf{AC}} \mathsf{a})]$, which matches modulo AC. The constrained formula $x * (\mathsf{b} * (\mathsf{b} * y)) =_{\mathsf{AC}} \mathsf{a} * \mathsf{b}$ $[x =_{\mathsf{AC}} \mathsf{b}]$ is AC-equivalent to $((x * \mathsf{b}) * \mathsf{b}) * y =_{\mathsf{AC}} \mathsf{a} * \mathsf{b}$ $[x =_{\mathsf{AC}} \mathsf{b}]$, and since every instance of this formula matches the left-hand side of the rule, it can be reduced to $\mathsf{a} * y =_{\mathsf{AC}} \mathsf{a} * \mathsf{b}$ $[x =_{\mathsf{AC}} \mathsf{b}]$. This notion of reduction is called *total simplification.* There is also a notion of *partial simplification*, where constrained terms are reduced to pairs. This happens when a rule does not necessarily match; for example the constrained formula $x * (\mathsf{b} * (\mathsf{b} * y)) =_{\mathsf{AC}} \mathsf{a} * \mathsf{b}$ $[\neg(x =_{\mathsf{AC}} y]$ reduces to the pair $\mathsf{a} * y =_{\mathsf{AC}} \mathsf{a} * \mathsf{b}$ $[\neg(x =_{\mathsf{AC}} y \wedge (x * x) * x =_{AC} (x * \mathsf{b}) * \mathsf{b} \wedge \neg(x =_{\mathsf{AC}} \mathsf{a})]$ and $((x * \mathsf{b}) * \mathsf{b}) * y =_{\mathsf{AC}} \mathsf{c} * \mathsf{b}$ $[\neg(x =_{\mathsf{AC}} y) \wedge \neg((x * x) * x =_{AC} (x * \mathsf{b}) * \mathsf{b} \wedge \neg(x =_{\mathsf{AC}} \mathsf{a})]$.

There are many similarities between these equational systems and LCTRSs; to a large extent they can be seen as non-standard LCTRSs. From this perspective, complete simplification is exactly constrained rewriting as we saw in Section 4. We have no notion of partial simplification, because it fundamentally relies on the symbols from terms being moved into the constraint, but similar techniques could be defined for the special case that $\Sigma_{terms} = \emptyset$.

However, LCTRSs do not allow reasoning modulo a theory, which alters fundamental properties like computability of reduction. Moreover, the systems from [11] violate an essential rule in LCTRSs: logical terms reduce only to their value. In the presence of rules like $x + (y + z) \to y$, many analysis techniques break.

Thus, while there is an overlap in expressability between these two formalisms, we do not claim to cover or improve on this style of constrained rewriting. The dynamics of the systems are too different, and so are their purposes: where equational systems are designed for equational reasoning in logic, LCTRSs are designed for analysing programs. In the rest of this section, we have seen how LCTRSs relate to several formalisms which share this goal.

6 Analysing LCTRSs

Several times we have alluded to the ease of analysis in LCTRSs, so it is time to give some indication of how this is done. Unfortunately, we cannot do this justice, as there are many questions for analysis and little space. To give some ideas of how common techniques extend to LCTRSs, we will now briefly study some basic confluence and termination results.

6.1 (Weak) Orthogonality

Confluence is the property that whenever $s \to_{\mathcal{R}}^* t$ and $s \to_{\mathcal{R}}^* q$ there is some w such that $t \to_{\mathcal{R}}^* w$ and $q \to_{\mathcal{R}}^* w$. We will extend the common notion of *orthogonality*, a property which implies confluence, to LCTRSs.

It is well-known that for any pair of terms which can be unified, there is a *most general unifier*. Phrased differently, if s and t have distinct variables, and $s\gamma = t\gamma$, then there is a substitution δ such that also $s\delta = t\delta$, and any unifying substitution γ can be written in the form $\epsilon \circ \delta$ for some substitution ϵ (here, $(\epsilon \circ \delta)(x) = \delta(x)\epsilon$ if $x \in Dom(\delta)$ and $\epsilon(x)$ otherwise). A substitution γ *respects variables* of a rule ρ if $\gamma(x)$ is a value or variable for all x in $LVar(\rho)$. If γ respects variables of $l \to r\ [\varphi]$, then $l\gamma \to r\gamma\ [\varphi\gamma]$ is also a rule.

Definition 1 (Critical Pair). *Given rules $\rho_1 \equiv l_1 \to r_1\ [\varphi_1]$ and $\rho_2 \equiv l_2 \to r_2$ $[\varphi_2]$ with distinct variables, the* critical pairs *of ρ_1, ρ_2 are all tuples $\langle s, t, \varphi \rangle$ where:*

- *l_1 can be written as $C[l_1']$, where l_1' is not a variable, but is unifiable with l_2;*
- *$C \neq \Box$, or not $\rho_1 = \rho_2$ modulo renaming of variables, or $Var(r_1) \not\subseteq Var(l_1)$;*
- *the most general unifier γ of l_1' and l_2 respects variables of both ρ_1 and ρ_2;*
- *$\varphi_1\gamma \wedge \varphi_2\gamma$ is satisfiable;*
- *$s = r_1\gamma$ and $t = (C\gamma)[r_2\gamma]$ and $\varphi = \varphi_1\gamma \wedge \varphi_2\gamma$.*

The critical pairs for calculations of a rule ρ are all critical pairs of ρ with any "rule" of the form $f(x_1, \dots, x_n) \to y\ [y = f(\boldsymbol{x})]$ with $f \in \Sigma_{theory} \setminus \mathcal{V}al$.

Note that a rule $f \to g(x)$ has a critical pair with its own renamed copy: $\langle g(x), g(y), \mathsf{true} \wedge \mathsf{true} \rangle$. This is necessary because fresh variables in the right-hand sides of rules are a very likely source of non-confluence.

Example 18. Consider the following rules:

$$\begin{array}{lrl} (\rho_1) & f(x_1, y_1) \to g(x_1 + y_1) & [x_1 \geq y_1] \\ (\rho_2) & f(x_2, y_2) \to g(x_2) & [x_2 \leq y_2] \\ (\rho_3) & f(x_3, y_3) \to g(y_3) & [x_3 < y_3] \\ (\rho_4) & f(x_4, x_4 + y_4) \to g(y_4) & [x_4 > \mathsf{0}] \end{array}$$

There are no critical pairs between ρ_1 and ρ_3: although $f(x_1, y_1)$ and $f(x_3, y_3)$ can be unified (with most general unifier $[x_1 := x,\ x_3 := x,\ y_1 := y,\ y_3 := y]$), the formula $x \geq y \wedge x < y$ is not satisfiable. On the other hand, ρ_1 and ρ_2 do admit a critical pair: $\langle g(x+y), g(y), x \geq y \wedge x \leq y\rangle$. None of the rules ρ_1, ρ_2 or ρ_3 gives a critical pair with ρ_4, since in the resulting mgu γ we have $\gamma(y_1) = x + y$, and thus this substitution does not respect the variables of ρ_1, ρ_2, ρ_3. Finally, ρ_4 has a critical pair for calculations, $\langle g(y), f(x, z), x > \mathsf{0} \wedge z = x + y\rangle$.

Definition 2 (Weak Orthogonality). *A critical pair $\langle s, t, \varphi\rangle$ is* trivial *if for every substitution γ which respects s $[\varphi]$ we have $s\gamma = t\gamma$. An LCTRS $\mathcal{R}$ is* weakly orthogonal *if the left-hand side of each rule is linear (no variable occurs more than once), and for any pair $\rho_1, \rho_2 \in \mathcal{R}$: every critical pair between ρ_1 and a variable-renamed copy of ρ_2, and every critical pair of ρ_1 for calculations, is trivial. It is* orthogonal *if there are no critical pairs.*

The following result follows much like its unconstrained counterpart:

Theorem 4. *A weakly orthogonal LCTRS is confluent.*

Example 19. sum is orthogonal, so by Theorem 4 this LCTRS is confluent.

6.2 The Recursive Path Ordering

To prove termination of a TRS, it suffices to show that its rules are included in the *recursive path ordering* [2], a well-founded ordering $\succ$ which is monotonic and stable under substitutions. We will consider a simple variation of this ordering. To deal with the possibly infinite number of values, we assume that Σ_{theory} contains a symbol $\sqsupset_\iota$ for all sorts ι occurring in $\mathcal{V}al$, which is mapped to a well-founded ordering $>_\iota$ in $\mathcal{I}_\iota$. For example, we might take $\sqsupset_{\mathsf{int}} = \lambda xy.x > y \wedge x \geq 0$. We also assume given a well-founded ordering $\rhd$ on the symbols of $\Sigma_{terms} \setminus \Sigma_{theory}$.

The recursive path ordering is defined by the following derivation rules:

1. $s \succeq t$ $[\varphi]$ if one of the following holds:
   (a) $s, t \in \mathcal{T}erms(\Sigma_{theory}, Var(\varphi))$, and $\varphi \Rightarrow (s = t \vee s \sqsupset t)$ is valid
   (b) $s = f(s_1, \ldots, s_n), t = f(t_1, \ldots, t_n)$ with $f \notin \Sigma_{theory}$ and each $s_i \succeq t_i$ $[\varphi]$
   (c) $s \succ t$ $[\varphi]$, or $s = t$ is a variable
2. $s \succ t$ $[\varphi]$ if one of the following holds:
   (a) $s, t \in \mathcal{T}erms(\Sigma_{theory}, Var(\varphi))$, and $\varphi \Rightarrow s \sqsupset t$ is valid
   (b) $s = f(s_1, \ldots, s_n)$ with $f \in \Sigma_{terms} \setminus \Sigma_{theory}$ and one of:
      i. $s_i \succeq t$ $[\varphi]$ for some $i \in \{1, \ldots, n\}$
      ii. $t = g(t_1, \ldots, t_m)$ with $g \in \Sigma_{theory}$ or $f \rhd g$, and for all i: $s \succ t_i$ $[\varphi]$
      iii. $t = f(t_1, \ldots, t_n)$, all $s_i \succeq t_i$ $[\varphi]$ and for at least one i: $s_i \succ t_i$ $[\varphi]$
      iv. $t \in Var(\varphi)$

Theorem 5. *An LCTRS $\mathcal{R}$ is terminating if we can choose a suitable $\sqsupset_\iota$ for all ι, and some well-founded $\rhd$, such that $l \succ r\ [\varphi]$ for all $l \to r\ [\varphi] \in \mathcal{R}$.*

Proof. We can define a pair $(\equiv, >)$ of an equivalence relation and a compatible ordering with $\to_{\mathtt{calc}} \subseteq \equiv$ and $C[s] > C[t]$ if $s \succ t$ [true] and $s \notin \mathcal{T}erms(\Sigma_{theory}, \emptyset)$. Having these, we observe first that $>$ is well-founded, and second that if $l \succ r\ [\varphi]$, then $l\gamma \succ r\gamma$ [true] for all substitutions γ which respect $l \to r\ [\varphi]$.

Example 20. Taking $n \sqsupset_{\mathsf{int}} m$ if $n > m$ and $n \geq 0$, the sum system is terminating by the recursive path ordering: For the first rule, $\mathsf{sum}(x) \succ \mathsf{0}\ [\mathsf{0} \geq x]$ by 2(b)ii. For the second, writing $\varphi = \neg(\mathsf{0} \geq x)$, we have $\mathsf{sum}(x) \succ x + \mathsf{sum}(x + -\mathsf{1})\ [\varphi]$ by 2(b)ii because $\mathsf{sum}(x) \succ x\ [\varphi]$ by 2(b)iv, and $\mathsf{sum}(x) \succ \mathsf{sum}(x + -\mathsf{1})\ [\varphi]$ by 2(b)iii because $x \succ x + -\mathsf{1}\ [\varphi]$ by 2a, because $\varphi \Rightarrow (x > x + -\mathsf{1} \wedge x \geq \mathsf{0})$ is valid.

Note that Example 3, with encoded constraints, cannot be handled by RPO.

Of course, this is a very basic version of the recursive path ordering. There are various ways to strengthen the technique, but this is left for future work.

6.3 Observations

Both when analysing confluence and termination, a pattern appears: existing techniques extend in fairly natural way, with the constraints handled by proving validity of some formula. In other techniques we have studied but omitted here (such as dependency pairs and inductive equality proofs) a similar pattern arises.

Importantly, this pattern does not depend on the kind of theory we use: analysis takes a similar form whether we reason about integer arrays, reals or bitvectors. The difference is in how to solve the resulting formulas. When automatically analysing properties of LCTRSs, it seems natural to combine a dedicated analysis tool with SMT-solvers for the theory of interest. This way, we can immediately profit from the continuing improvement of the SMT-community, without having to adjust our methods when a new theory is explored.

7 Conclusion

In this paper, we have studied *logical constrained term rewriting systems*. LCTRSs offer an approach to program analysis for a large variety of languages and analysis questions. Due to their similarity to normal term rewriting, we can easily transpose the many powerful techniques of traditional term rewriting. However, by natively handling constraints, we obtain a much simpler analysis than if we were to encode the constraints in the rules.

In conclusion, LCTRS can be summarised with four keywords: They are *natural*: values in the logic are modelled with constants, and calculations do not need to be encoded. They are *general*: LCTRSs are not restricted to for instance the integers, but can handle all kinds of theories. They are *versatile*: LCTRSs can model a wide range of problems, from termination and overflow analysis to program equivalence, and can represent examples from many existing formalisms of constrained or integer rewriting. Finally, they are *flexible*: common analysis techniques for term rewriting extend to LCTRSs without much effort.

In the future, we aim to provide a tool to rewrite and analyse LCTRSs. Such analysis would not necessarily need special treatment for the various theories: in many cases (as we saw in Section 6), an LCTRS problem can be converted into a sequence of SMT-queries which might be fed into an external solver.

In addition, we hope to extend translations of program analysis from e.g. [3,9,13] with arrays and bitvectors, thus making use of the greater generality of LCTRSs, and the power of SMT-solvers for various theories.

References

1. Community. SMT-LIB. `http://www.smtlib.org/`.
2. N. Dershowitz. Orderings for term rewriting systems. *TCS*, 17(3):279–301, 1982.
3. 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.
4. S. Falke and D. Kapur. Rewriting induction + linear arithmetic = decision procedure. In B. Gramlich, D. Miller, and U. Sattler, editors, *Proc. IJCAR 12*, volume 7364 of *LNAI*, pages 241–255. Springer, 2012.
5. S. Falke, D. Kapur, and C. Sinz. Termination analysis of C programs using compiler intermediate languages. In M. Schmidt-Schauß, editor, *Proc. RTA 11*, volume 10 of *LIPIcs*, pages 41–50. Dagstuhl, 2011.
6. S. Falke, D. Kapur, and C. Sinz. Termination analysis of imperative programs using bitvector arithmetic. In R. Joshi, P. Müller, and A. Podelski, editors, *Proc. VSTTE 12*, volume 7152 of *LNCS*, pages 261–277. Springer, 2012.
7. 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.
8. C. Fuhs, J. Giesl, M. Plücker, P. Schneider-Kamp, and S. Falke. Proving termination of integer term rewriting. In R. Treinen, editor, *Proc. RTA 09*, volume 5595 of *LNCS*, pages 32–47. Springer, 2009.
9. 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 Trans. Inf. & Syst.*, 1(2):100–121, 2008. In Japanese.
10. J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *TOPLAS*, 33(2):7:1–7:39, 2011.
11. C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990.
12. N. Nakabayashi, N. Nishida, K. Kusakari, T. Sakabe, and M. Sakai. Lemma generation method in rewriting induction for constrained term rewriting systems. *Computer Software*, 28(1):173–189, 2010. In Japanese.
13. C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated termination analysis of java bytecode by term rewriting. In C. Lynch, editor, *Proc. RTA 10*, volume 6 of *LIPIcs*, pages 259–276. Dagstuhl, 2010.
14. 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*, volume 6816 of *LNCS*, pages 138–155. Springer, 2011.
15. P. Schneider-Kamp, J. Giesl, T. Ströder, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs with cut. *Theory and Practice of Logic Programming*, 10(4–6):365–381, 2010.

A Omitted Proofs

In the text, proofs were omitted pretty much everywhere. Mostly, the proofs are straightforward. They are included here for completeness.

A.1 Proofs for Section 4

Theorem 1. If $s \to_{\mathcal{R}} t$ then also s [true] $\to_{\mathcal{R}} t$ [true].

Proof. Suppose $s \to_{\mathtt{rule}} t$, so $s = C[l\gamma]$ and $t = C[r\gamma]$ for some rule $l \to r$ $[\varphi]$ and substitution γ which respects $l \to r$ $[\varphi]$. By definition of *respects*, $Dom(\gamma) = Var(l) \cup Var(r) \cup Var(\varphi)$ and $\gamma(x)$ is a (variable or) value for all $x \in LVar(l \to r$ $[\varphi]$ and $\varphi\gamma$ is valid, so certainly $\varphi\gamma\delta$ is valid for all δ which substitute variables by values, for which $\mathsf{true}\delta$ is valid. Thus, the conditions to apply constrained term reduction are satisfied, and indeed $C[l\gamma]$ [true] $\to_{\mathtt{rule}} C[r\gamma]$ [true]

Suppose $s \to_{\mathtt{calc}} t$, so $s = C[f(s_1, \dots, s_n)]$ and $t = C[v]$ with v a value such that $[\![v]\!] = [\![f(s_1, \dots, s_n)]\!]$. All s_i are values. Thus, s [true] $\to_{\mathtt{calc}} C[x]$ [$\mathsf{true} \wedge x = f(s_1, \dots, s_n)$] $\approx C[v]$ [true].

To split the proof for the next theorem in more manageable chunks, let us introduce two lemmas.

Lemma 1. *If γ respects s $[\varphi]$, then for all s', φ' with s $[\varphi] \approx s'$ $[\varphi']$ there is a substitution γ' such that $s\gamma = s'\gamma'$.*

Proof. By definition of s' $[\varphi'] \approx s$ $[\varphi]$.

Lemma 2. *If γ respects s $[\varphi]$ and s $[\varphi] \to_{\mathtt{rule}} t$ $[\psi]$, then γ respects t $[\psi]$ and $s\gamma \to_{\mathtt{rule}} t\gamma$.*

Proof. By definition of $\to_{\mathtt{rule}}$ we know that $\psi = \varphi$, so clearly γ respects the resulting constrained term. Furthermore, we can write $s = C[l\delta]$ and $t = C[r\delta]$ for some rule $l \to r$ $[c]$ and substitution δ for which the following properties hold, where $\delta\gamma$ is the substitution on domain $Dom(\delta)$ such that each $\delta\gamma(x) = \delta(x)\gamma$:

- $Dom(\delta) = Var(l) \cup Var(r) \cup Var(\delta)$, so the same holds for $\delta\gamma$
- $\delta(x)$ is a value or variable in $Var(\varphi)$ for all $x \in LVar(l \to r$ $[c])$, so for the substitution $\delta\gamma$ we have: if $\delta(x)$ is a value, then $\delta\gamma(x) = \delta(x)\gamma$ is the same value, and if it is a variable in $Var(\varphi)$, then because γ respects φ also $\delta(x)\gamma$ is a value;
- $\varphi \Rightarrow (c\delta)$ is valid, so since $\varphi\gamma$ is valid by definition of "respects", also $(c\delta)\gamma = c(\delta\gamma)$ is valid

Thus, $\delta\gamma$ respects the same rule, so $s\gamma \to_{\mathtt{rule}} t\gamma$ as required.

Theorem 2. If s $[\varphi] \to_{\mathcal{R}} t$ $[\psi]$ then for all substitutions γ which respect s $[\varphi]$ there is a substitution δ which respects t $[\psi]$ such that $s\gamma \to_{\mathcal{R}} t\gamma$.

Proof. Suppose $s\ [\varphi] \to_{\mathcal{R}} t\ [\psi]$ and γ respects $s\ [\varphi]$. Then $s\ [\varphi] \approx s'\ [\varphi']\ (\to_{\texttt{calc}} \cup \to_{\texttt{rule}})\ t'\ [\psi'] \approx t'\ [\psi']$. By Lemma 1, there exists some δ which respects $s'\ [\varphi']$ such that $s\gamma = s'\delta$.

Suppose this was a $\to_{\texttt{rule}}$ step. By Lemma 2, γ also respects ψ' and $s'\delta \to_{\texttt{rule}} t'\delta$. By Lemma 1 again, there is a substitution χ which respects ψ, such that $t'\delta = t\chi$. Thus, $s\gamma = s'\delta \to_{\texttt{rule}} t'\delta = t\chi$.

Alternatively, suppose this was a $\to_{\texttt{calc}}$ step, so $s' = C[f(s_1, \dots, s_n)]$ with all s_i values or variables in $\mathit{Var}(\varphi')$, and $\psi' = \varphi' \wedge x = f(\boldsymbol{s})$ and $t' = C[x]$. All $s_i\delta$ must be values, so there is a value v with $[\![v]\!] = [\![f(\boldsymbol{s})\gamma]\!]$. Let $\delta'\delta \cup [x := v]$. Then δ' respects $t'\ [\psi']$ and $s'\delta = C\delta[f(\boldsymbol{s})\delta] \to_{\texttt{calc}} C\delta[v] = t'\delta'$. By Lemma 1 again, there is a substitution χ which respects ψ, such that $t'\delta' = t\chi$. Thus, $s\gamma = s'\delta \to_{\texttt{calc}} t'\delta' = t\chi$.

A.2 Proofs for Section 5

Claim. When a sorted ITRS is transformed into an LCTRS, the resulting set of terms and rewrite relation is the same as in the original.

Proof. $\Sigma \cup \Sigma_{int} = \Sigma_{terms} \cup \Sigma_{theory}$, so the set of terms is clearly the same. Every rule in R is in $\mathcal{R}$, and every rule in $\mathcal{PD}$ is either an instance of a $\to_{\texttt{calc}}$ step, or of one of the two irregular rules. Similarly, if $C[f(s_1, \dots, s_n)] \to_{\texttt{calc}} C[v]$, then $C[f(\boldsymbol{s})] \to_{\mathcal{R}} C[v]$, and similar for any instance of the two irregular rules (as the variables x, y, z must all be instantiated with values).

In the text, we have also made a claim about conditional ITRSs. This is perhaps somewhat bold, as it is never fully defined (either here or in e.g. [13]) what we might expect of an innermost conditional integer TRS. But let us make some educated guesses.

CITRSs have a sorted signature $\Sigma \cup \Sigma_{int}$, where $\Sigma_{int} = \mathit{BOp} \cup \mathcal{V}al$ with $\mathit{BOp} = \{+, -, *, /, \%, >, \geq, <, \leq, =, \neq, \wedge, \Rightarrow\}$ and $\mathcal{V}al = \{\mathsf{true},\ \mathsf{false}\} \cup \{\mathsf{n} \mid n \in \mathbb{Z}\}$. The set of rules is $R \cup \mathcal{PD}$, where $\mathcal{PD} = \{n \circ m \to k \mid n, m, k \in \mathbb{Z} \cup \mathbb{B},\ \circ \in \mathit{BOp} \mid n \circ m = k$ holds in $\mathbb{Z}$ and $\mathbb{B}\}$. We assume the symbols are sorted, with sorts int and bool.

The rules in R are conditional rules of the following form: $l \to r \mid s \to^* \mathsf{true}$. If the condition has the form $\mathsf{true} \to^* \mathsf{true}$, it is omitted. We moreover assume that the elements of Σ_{int} are constructors with respect to R, and that s is built using only function symbols in $\Sigma_{int} \setminus \{/, \%\}$. To avoid confusion with the $|$ used in set construction, we will alternatively denote the condition with: $l \to r \Longleftarrow s \to^* \mathsf{true}$.

The conditional innermost rewrite relation is defined as follows, defining R' as the set $\{l \to r \mid l \to r \Longleftarrow s \to^* \mathsf{true} \in R\}$:

- $A_0 := \emptyset$
- $A_{n+1} := \{l\gamma \to r\gamma \mid l \to r \Longleftarrow s \to^* t \in R$ and γ a substitution such that all $\gamma(x)$ and all strict subterms of $l\gamma$ are in normal form with respect to $R' \mid s \to^*_{A_n} t\}$

- $A_\omega := \bigcup_{n \in \mathbb{N}} A_n$
- $\to_{R \cup \mathcal{PD}}$ is the relation given by: $C[l] \to_{R \cup \mathcal{PD}} C[r]$ if $l \to r \in A_\omega$.

Analysing this relation, we obtain:

Lemma 3. *$s \to_{R \cup \mathcal{PD}} t$ if and only if there are a context C, a rule $l \to r \Longleftarrow q \to^* \mathsf{true}$ and a substitution γ such that:*

- *$s = C[l\gamma]$;*
- *$t = C[r\gamma]$;*
- *$\gamma(x)$ is an integer or boolean for all $x \in Var(q)$;*
- *$q\gamma$, interpreted as a constraint over $\mathbb{Z}$ and $\mathbb{B}$ (with the usual interpretations of the functions) is valid*

Proof. If $s \to_{R \cup \mathcal{PD}} t$, then $s = C[l\gamma]$ and $t = C[r\gamma]$ for some $l \to r \Longleftarrow q \to^* \mathsf{true} \in R \cup \mathcal{PD}$, some context C and some substitution γ such that all $\gamma(x)$ and all strict subterms of $l\gamma$ are in normal form with respect to R', which has the property that $q\gamma \to^*_{A_n} \mathsf{true}$.

Now, $q\gamma$ has the following property: every subterm is a functional term $f(\boldsymbol{w})$ with $f \in \Sigma_{int}$, or a normal form $\gamma(x)$.

Observing that terms which are in normal form with respect to R' also do not reduce with any A_n, and that a term $f(q_1, \dots, q_n)$ with $f \in \Sigma_{int}$ cannot root-reduce if some q_i is not a "value" (integer or boolean), we see: $q\gamma$ can only be reduced to true if all subterms which are normal forms are in fact values. Thus: all $\gamma(x)$ with $x \in Var(q)$ are values.

Moreover, since a term $f(\boldsymbol{q}) \in \mathcal{T}erms(\Sigma_{int}, \mathcal{V})$ can only be reduced by the $\mathcal{PD}$-rules, and these rules exactly calculate the value of a term over Σ_{int}, we find that $q\gamma$, interpreted as a constraint over $\mathbb{Z} \cup \mathbb{B}$, must be valid. Thus, we have one direction of the lemma!

For the other direction, we observe that $\to_{\mathcal{PD}}$ is contained in $\to_{A_1}$, because the $\mathcal{PD}$-rules do not carry conditions (or rather, they carry conditions $\mathsf{true} \to^* \mathsf{true}$). We also note that if $q\gamma$ is valid when considered as a ground constraint, then $q\gamma \to^*_{\mathcal{PD}} \mathsf{true}$.

Thus, if $l \to r \Longleftarrow q \to^* \mathsf{true}$ is a rule, and γ is a substitution such that $q\gamma$ is valid, then $l\gamma \to r\gamma \in A_2$. This gives the other direction.

Now let us consider a translation to LCTRSs. We define $\Sigma_{terms} := \Sigma \cup \mathcal{V}al \cup \{/, \%\}$ and $\Sigma_{theory} := \mathcal{V}al \cup (BOp \setminus \{/, \%\}$. Let $\mathcal{R} := \{l \to r\ [\varphi] \mid l \to r \Longleftarrow \varphi \to^* \mathsf{true}\} \cup \{x/y \to z\ [x = y * z],\ x\%y \to z\ [x = y * u + z \wedge 0 \leq z \wedge z < y]\}$.

With this transformation, we obtain the claim in the text:

Claim. When a sorted, conditional innermost ITRS is transformed into an innermost LCTRS, the resulting set of terms and rewrite relation is the same as in the original.

Proof. It is easy to see that the terms in this new LCTRS are exactly the terms in the original ITRS. The claim follows quickly with Lemma 3.

Theorem 3. For all ground terms s, t:

- if $s \to_{\mathcal{R}} t$ in a $\mathbb{Z}$-TRS, and $s \to_{\mathtt{calc}}^* s' \in \mathcal{T}erms(\Sigma_{terms} \cup \Sigma_{theory}, \mathcal{V})$, then exists t' such that $t \to_{\mathtt{calc}}^* t'$ and $s' \to_{\mathcal{R}}^+ t'$ in the corresponding LCTRS;
- if $s \to_{\mathtt{rule}} t$ in the corresponding LCTRS, and $s' \in \mathcal{T}erms(\Sigma, \mathcal{V})$ such that $s' \to_{\mathtt{calc}}^* s$, then there is a term t' such that $t' \to_{\mathtt{calc}}^* t$ and $s' \to_{\mathcal{R}} t'$ in the original $\mathbb{Z}$-TRS.

Proof. Suppose $s \to_{\mathcal{R}} t$ in a $\mathbb{Z}$-TRS, so $s = C[l\gamma]$ and $t = C[r\gamma]$ for some rule $l \to r\ [\varphi]$ and substitution γ on domain $Var(l) \cup Var(r) \cup Var(\varphi)$ such that:

- $\gamma(x) \in \mathcal{T}erms(\Sigma_{int}, \mathcal{V})$ for all $x \in Dom(\gamma)$ which have sort int, and must be ground because s is ground
- $\varphi\gamma$ is valid

Since l does not contain any symbols of Σ_{int}, we must have: if $l\gamma \to_{\mathtt{calc}}^* s'$ then $s' = l\delta$ with each $\gamma(x) \to_{\mathtt{calc}}^* \delta(x)$. But since $\to_{\mathtt{calc}}$ has unique normal forms, we then have: $s' \to_{\mathtt{calc}}^* l\gamma^{\downarrow_{calc}}$, where $\gamma^{\downarrow_{calc}}$ is the substitution on domain $Dom(\gamma)$ with $\gamma^{\downarrow_{calc}}(x) = \gamma(x)\downarrow_{calc}$. We observe that $\gamma^{\downarrow_{calc}}$ respects the LCTRS-rule $l \to r\ [\varphi]$: all $\gamma(x)$ are values for x of sort int (so certainly for all x occurring in φ) and since $[\![D[q_1, \dots, q_m]]\!] = [\![D[v_1, \dots, v_m]]\!]$ whenever q_i has value v_i, we see that $\varphi\gamma^{\downarrow_{calc}}$ is still valid. As $t = r\gamma$ we are done defining $t' = r\gamma^{\downarrow_{calc}}$, as clearly $t \to_{\mathtt{calc}}^* t'$, and $s' \to_{\mathtt{calc}}^+ l\gamma^{\downarrow_{calc}} \to_{\mathtt{rule}} t'$.

For the other direction, suppose $s' \to_{\mathtt{calc}}^* s$ and $s = l\gamma$ for some rule $l \to r\ [\varphi]$ and substitution γ which respects this rule. Then, considering the inverse relation of $\to_{\mathtt{calc}}$, the term s' *must* have the form $l\delta$ with $\gamma = \delta^{\downarrow_{calc}}$. For those x not occurring in l, we take a canonical representation, so $\mathsf{1} + (\mathsf{1} + (\dots))$ or $-\mathsf{1} + (-\mathsf{1} + (\dots))$ or $\mathsf{0}$. That way, δ maps all variables of sort int to a substitution over Σ_{int} as required. As we have $t = r\gamma$, we are done defining $t' = r\delta$ (the validity is easily seen to transpose).

A.3 Proofs for Section 6

The ideas in Section 6 correspond largely to similar results in the unconstrained setting. Due to the presence of constraints a few aspects of the definitions and proofs must change, but there are no great surprises (except perhaps in the definition of $\equiv$ and $>$ for the recursive path ordering). As mentioned in the text, this section is primarily to show the way in which existing results extend to the constrained setting, which is mostly very natural.

Weak Orthogonality In order to prove Theorem 4, we need the following Lemmas:

Lemma 4. *Let $\mathcal{R}$ be weakly orthogonal, and suppose we have rules $\rho_1 \equiv l_1 \to r_1\ [\varphi_1]$ and $\rho_2 \equiv l_2 \to r_2\ [\varphi_2]$. Suppose moreover that we have substitutions γ and δ which respect ρ_1 and ρ_2 respectively, a context C and a non-variable subterm l' of l_1 such that $l_1 = C[l']$ and $l'\gamma = l_2\delta$.*

Then $r_1\gamma = (C\gamma)[r_2\delta]$.

Proof. In the given situation, we have two options: either $C = \Box$ and the rules are equal modulo renaming of variables and $\mathit{Var}(r_1) \subseteq \mathit{Var}(l_1)$, or one of these properties does not hold. In the first case, we rename the variables in ρ_2 so the rules are truly equal (we can do this provided we also rename the variables in the domain of δ). As $l' = l_1 = l_2$, the given statement $l'\gamma = l_2\delta$ implies that $\gamma(x) = \delta(x)$ for all variables occurring in the left-hand side of the rule – which, by assumption, are also all variables occurring in $r_1 = r_2$. Thus, $r_1\gamma = r_1\delta = r_2\delta = (C\gamma)[r_2\delta]$.

In the second case, we rename the variables in ρ_2 and δ so the rules use distinct variables; the variables being distinct, we may also merge γ and δ. Thus we are given that $l'\gamma = l_2\gamma$ and must prove that $r_1\gamma = (C\gamma)[r_2\gamma]$. For γ, we may assume it has a domain $\mathit{Var}(l_1) \cup \mathit{Var}(l_2) \cup \mathit{Var}(r_1) \cup \mathit{Var}(r_2) \cup \mathit{Var}(\varphi_1) \cup \mathit{Var}(\varphi_2)$, that all variables in $\mathit{Var}(\varphi_1) \cup \mathit{Var}(\varphi_2)$ are mapped to values, and that both $\varphi_1\gamma$ and $\varphi_2\gamma$ hold.

We observe that unifiability of l' and l_2 provides a critical pair $\langle r_1\epsilon,\ (C\epsilon)[r_2\epsilon],\ \varphi_1\epsilon \wedge \varphi_2\epsilon\rangle$, where ϵ is a most general unifier. We extend ϵ with $\epsilon(x) = x$ for all $x \in \mathit{Dom}(\gamma)$ which do not occur in the l_i. By the definition of an MGU, we can find a substitution η such that for all terms s with $\mathit{Var}(s) \subseteq \mathit{Dom}(\epsilon)$: $(s\epsilon)\eta = s\gamma$. In particular, this holds for r_1, r_2, φ_1 and φ_2 and the context C.

By weak orthogonality, we know that if η respects both $r_1\epsilon\ [\varphi_1\epsilon \wedge \varphi_2\epsilon]$ and $(C\epsilon)[r_2\epsilon]\ [\varphi_1\epsilon \wedge \varphi_2\epsilon]$, then $r_1\gamma = r_1\epsilon\eta = (C\epsilon)[r_2\epsilon]\eta = C\gamma[r_2\gamma]$ as required. It remains to be seen that η indeed respects both constrained terms. So let x be a variable in $\mathit{Var}(\varphi_1\epsilon \wedge \varphi_2\epsilon)$; we must see that $\eta(x)$ is a value. The truth of this, however, is evident enough: if it were not the case, then $\gamma(x)$ would not be a value for some variable occurring in φ_1 or φ_2. We must also see that $(\varphi_1\epsilon \wedge \varphi_2\epsilon)\eta$ is valid, but since this is exactly $\varphi_1\gamma \wedge \varphi_2\gamma$ and by assumption γ respects both rules, this is evident.

Lemma 5. *Let $\mathcal{R}$ be weakly orthogonal, $l \to r\ [\varphi] \in \mathcal{R}$ and γ a substitution which respects this rule, such that $l\gamma$ can be written as $C[f(s_1, \ldots, s_n)]$ with $f \in \Sigma_{\mathit{theory}} \setminus \Sigma_{\mathit{terms}}$ and all s_i are values.*

Then $r\gamma = (C\gamma)[v]$ where v is the value of $f(s_1, \ldots, s_n)$.

Proof. The proof of this is much the same as the proof of Lemma 4 (we can see the calculation step as the second rule in this proof).

Now we can prove the main theorem:

Theorem 4. A weakly orthogonal LCTRS is confluent.

Proof. We will see that $\mathrel{+\!\!\!+\!\!\!\!\rightarrow}$ is confluent, where $\mathrel{+\!\!\!+\!\!\!\!\rightarrow}$ is the parallel extension of $\to_{\mathcal{R}}$. This implies confluence of $\to_{\mathcal{R}}$, for if $s \to_{\mathcal{R}}^* t$ and $s \to_{\mathcal{R}}^* q$, then (because $\to_{\mathcal{R}}$ is included in $\mathrel{+\!\!\!+\!\!\!\!\rightarrow}$) there is some w such that both $t \mathrel{+\!\!\!+\!\!\!\!\rightarrow}^* w$ and $q \mathrel{+\!\!\!+\!\!\!\!\rightarrow}^* w$, so (as $\mathrel{+\!\!\!+\!\!\!\!\rightarrow}$ is included in $\to_{\mathcal{R}}^*$) also $t \to_{\mathcal{R}}^* w$ and $q \to_{\mathcal{R}}^* w$ as required. Confluence of $\mathrel{+\!\!\!+\!\!\!\!\rightarrow}$ holds if we have the diamond property: if $s \mathrel{+\!\!\!+\!\!\!\!\rightarrow} t, q$ then there is a w such that $t, q \mathrel{+\!\!\!+\!\!\!\!\rightarrow} w$. This, we will prove to be the case in a weakly orthogonal LCTRS.

Suppose $s \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} t$ and $s \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} q$; we construct a suitable w with induction on s. If s is a variable, then $s = t = q$, and we are done choosing $w := s$ as well. Otherwise let $s = f(s_1, \dots, s_n)$. If neither reduction is at the root, then $t = f(t_1, \dots, t_n)$ and $q = f(q_1, \dots, q_n)$ and each $s_i \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} t_i, q_i$. By the induction hypothesis there are $w_1, \dots, w_n$ such that each $s_i, t_i \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} w_i$. Let $w := f(w_1, \dots, w_n)$. Then both $t \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} w$ and $q \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} w$. Otherwise, at least one of the reductions is at the root. We can safely assume that this is the first of the two.

So suppose $s \to_{\mathcal{R}} t$ by a root-reduction and also $s \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} q$. If the first is a reduction with $\to_{\texttt{calc}}$, then s has a very specific form; either $q = s$ or $q = t$. In either case we are quickly done. So, alternatively, let $l \to r\ [c]$ be the relevant rule, with corresponding substitution γ (so $s = l\gamma$ and $t = r\gamma$ and γ respects this rule). Let $p_1, \dots, p_n$ be the positions in s which are reduced in the reduction $s \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} q$. We can split $\{p_1, \dots, p_n\}$ into $\{a_1, \dots, a_k\} \cup \{b_1, \dots, b_{n-k}\}$, where the a_i are positions which are also non-variable positions in l and the b_i are all other positions. We can write $s \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} s' \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} q$ where the first step reduces the positions a_i and the second the positions b_i.

If $k \geq 2$, then l can be written as $C[l_1', l_2']$ and for each i, either there is a rule $l_i \to r_i\ [\varphi_i]$ and a substitution δ_i which respects this rule such that $l_i'\gamma = l_i\delta$ (in which case we let $q_i := r_i\delta$), or $l_i'\gamma$ can be written as $f(s_1, \dots, s_n)$ with $f \in \Sigma_{theory} \setminus \Sigma_{terms}$ and all s_i values (in which case we let q_i be the value of $f(\boldsymbol{s})$). By Lemma 4 or 5 respectively, $t = r\gamma$ both equals $(C\gamma)[q_1, l_2'\gamma]$ and $(C\gamma)[l_1'\gamma, q_2]$. This can only be the case if each $l_i'\gamma = q_i$ and $l\gamma = r\gamma$. Thus, $t = s$, and we are done choosing $w := q$.

If $k = 1$, then l can be written as $C[l']$ and either there are a rule $\tilde{l} \to \tilde{r}\ [\tilde{\varphi}]$ and a respecting substitution δ such that $l'\gamma = \tilde{l}\delta$ (in which case we define $v := \tilde{r}\delta$), or $l'\gamma \to_{\texttt{calc}} v$ for some value v. By Lemma 4 or 5 respectively, then $t = r\gamma = (C\gamma)[v] = s'$. Since we know that $s' \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} q$, it suffices once more to choose $w := q$.

Finally, if $k = 0$, then each of the parallel reduction steps takes place inside the substitution. For each of the b_i we can write: $b_i = x_i \cdot y_i$, with $l_{|x_i}$ a variable and $\gamma(l_{|x_i})_{|y_i} = s_{b_i}$. Let δ be the substitution γ, except with each $\gamma(l_{|x_i})_{|y_i}$ replaced by q_{b_i}. Then $q = l\delta$. Moreover, since each $\delta(x)$ is a reduct of $\gamma(x)$, the substitution δ respects any rule which γ also respects. Choosing $w := r\delta$ we have both $t = r\gamma \mathrel{-\!\!\!\!\parallel\!\!\!\!\to} r\delta = w$ and $q = l\delta \to_{\mathcal{R}} r\delta$.

The Recursive Path Ordering For the recursive path ordering, we follow the proof sketch in the paper. Thus, we first define an equivalence relation an an ordering on unconstrained terms. Then, in Lemma 6 we prove that the relations are indeed an equivalence relation and an ordering, but this is pretty much routine. We then see that $>$ is well-founded, which is similar to the corresponding proof for the normal recursive path ordering, but with some simple additional cases for terms with a root symbol in Σ_{theory}. Then, we prove that $\to_{\texttt{calc}}$ is included in $\equiv$, and $\to_{\texttt{rule}}$ in $>$. The latter has one tricky aspect: we cannot define $>$ to be monotonic. However, in Lemma 10 we will see that monotony is satisfied if the left-hand side is not a ground logical term, which is all we need.

Definition 3 **($\equiv$ and $>$).**

1. *$s \equiv t$ if one of the following holds:*
   (a) *s and t are terms with the same sort in $\mathcal{T}erms(\Sigma_{theory}, \emptyset)$ and $[\![s]\!] = [\![t]\!]$*
   (b) *$s = t \in \mathcal{V}$, or $s = f(s_1, \dots, s_n)$, $t = f(t_1, \dots, t_n)$ and each $s_i \equiv t_i$,*
2. *$s > t$ if one of the following holds:*
   (a) *$s, t \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$, and $[\![s]\!] >_\iota [\![t]\!]$*
   (b) *$s = f(s_1, \dots, s_n)$ with $f \in \Sigma_{terms} \setminus \Sigma_{theory}$ and one of:*
      i. *$s_i > t$ or $s_i \equiv t$ for some $i \in \{1, \dots, n\}$*
      ii. *$t = g(t_1, \dots, t_m)$ with either $g \in \Sigma_{theory}$ or $f \rhd g$, and for all $i \in \{1, \dots, m\}$: $s > t_i$*
      iii. *$t = f(t_1, \dots, t_n)$, there is some i with $s_i > t_i$ and for all other i: $s_i \geq t_i$*
   (c) *$s = f(s_1, \dots, s_n)$ with $f \in \Sigma_{theory}$ but $s \notin \mathcal{T}erms(\Sigma_{theory}, \emptyset)$, and:*
      – *$t = f(t_1, \dots, t_n)$, there is some i with $s_i > t_i$, and for all other i: $s_i \geq t_i$,* or
      – *$t \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$*
3. *$s \geq t$ if either $s \equiv t$ or $s > t$*

Then $(\equiv, >)$ is indeed a pair of an equivalence relation and a compatible well-founded ordering. Well-foundedness we will see separately, but as for the other properties:

Lemma 6. *$\equiv$ is transitive, symmetric and reflexive, $>$ is transitive, and $> \cdot \equiv$ is included in $>$.*

Proof. We first observe that if $s \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$ and $s \equiv t$, then $t \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$ and $[\![s]\!] = [\![t]\!]$ even if case 1b was applied; this follows easily with induction. Then transitivity of $\equiv$ for all terms is easily proved with induction on the definition of $\equiv$. Symmetry and reflexivity are equally evident, with induction on the definition.

For the other two properties, we prove the following: if $s \geq t \geq q$, and one of these is strict, then $s > q$. Note that we already know that if neither is strict, then $s \geq q$ because $s \equiv q$. We prove this by induction first on the derivation of $t \geq q$, second by induction on the derivation of $s \geq t$.

Noting that terms in $\mathcal{T}erms(\Sigma_{theory}, \emptyset)$ only reduce with $\equiv$ or $>$ to terms in the same set, we observe: if $s \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$, then $t, q \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$ and $[\![s]\!] \geq [\![t]\!] \geq [\![q]\!]$ with at least one strict; we are immediately done by transitivity of $>_\iota$ (a relation in the underlying theory).

If $q \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$ but this does not hold for s, then $s > q$ either by repeated application of 2(b)ii or 2c. So we are done unless none of $s, t, q \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$. Consider why $s \geq t$.

If $s > t$ by case 2(b)i, then $s > t$ because $s_i \geq t$, so by the second induction hypothesis and transitivity of $\equiv$ also $s_i > q$, so $s > q$ by case 2(b)i.

If $s > t$ by case 2(b)ii, then $s = f(\boldsymbol{s})$ and $t = g(\boldsymbol{t})$ with either $g \in \Sigma_{theory}$ or $f \rhd g$, and $s > t_i$ for all i. If $g \in \Sigma_{theory}$, then $t \geq q$ by case 1b or 2c. Either way, $h = g(\boldsymbol{q})$ with $t_i \geq q_i$ for all i, so $s > q_i$ by the first induction hypothesis, so $s > q$ by case 2(b)ii. If $g \notin \Sigma_{theory}$ but $t \geq q$ by case 1b or case 2(b)iii, $s > q$ follows in the same way. The only alternative is $t > q$ by 2(b)ii, in which case $q = h(\boldsymbol{q})$ with $s > t > q_i$ for all i, so by the first induction hypothesis $s > q_i$, and either $h \in \Sigma_{theory}$ or $f \rhd g \rhd h$. Since $\rhd$ is an ordering, it is transitive, so $s > q$ by the same case 2(b)ii.

In all other cases, $s = f(\boldsymbol{s})$ and $t = f(\boldsymbol{t})$ and $q = f(\boldsymbol{q})$ with $s_i \geq t_i \geq q_i$, with at least one $s_i > t_i$ or at least one $t_i > q_i$. By the first induction hypothesis, we find that $s_i \geq q_i$ and at least one $s_i > q_i$, and conclude with either case 2c or case 2(b)iii.

We must see two more things: first, that the ordering we define is well-founded on unconstrained terms, and second, that orienting the rules actually implies that $s > t$ whenever $s \to_{\mathtt{rule}} t$, and $s \equiv t$ whenever $s \to_{\mathtt{calc}} t$.

Lemma 7. *$>$ is well-founded.*

Proof. This holds if we can see that all terms s are terminating, that is, that there is no reduction $s > s_1 > s_2 > \ldots$.

We first observe: every variable is terminating. This is because variables are minimal with respect to $>$.

We then note: every element of $\mathcal{T}erms(\Sigma_{theory}, \emptyset)$ is terminating. We prove this by induction on $s \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$ by $[\![s]\!]$, ordered with the relevant $>_\iota$. The proof is trivial, noting that if $s > t$ then only 2a can be applicable, so t also must be ground, must have the same sort, and $s \sqsupset_\iota t$.

Observing that $\equiv$ and $>$ are compatible, termination of some term s implies termination of all $\equiv$-equivalent terms. Thus, fixing some n, we can use induction on $(s_1, \ldots, s_n)$ with all s_i in the set of terminating terms, ordered by the product extension of $>$ but considered modulo $\equiv$.

Then we see: every term of the form $f(\boldsymbol{s})$ with $f \in \Sigma_{theory}$ is terminating if all s_i are, as we prove individually for all f by induction on $\boldsymbol{s_i}$. $f(\boldsymbol{s})$ is terminating if all its reducts are; but the reducts of $f(\boldsymbol{s})$ are all either terms in $\mathcal{T}erms(\Sigma_{theory}, \emptyset)$, which case we are done, or have the form $f(\boldsymbol{t})$ with each $s_i \geq t_i$ and at least one $s_i > t_i$, in which case the induction hypothesis suffices.

What remains are terms of the form $f(s_1, \ldots, s_n)$ with $f \in \Sigma_{terms} \setminus \Sigma_{theory}$. We will prove that all terms of this form are terminating if all s_i are terminating, ordered first by f with $\rhd$ and second by $(s_1, \ldots, s_n)$ ordered by the product of $>$, modulo $\equiv$. A term is terminating if all its reducts are. So suppose $f(\boldsymbol{s}) > t$. We prove that t is terminating by a third reduction, on the derivation of $f(\boldsymbol{s}) > t$. There are only three possibilities:

- some $s_i \geq t$: whether $s_i > t$ or $s_i \equiv t$, we obtain termination of t from termination of s_i;
- $t = g(t_1, \ldots, t_n)$ with $s > t_i$ for all i, and either $f \rhd g$ or $g \in \Sigma_{theory}$; by the third induction hypothesis all t_i are terminating, so t is terminating by the first induction hypothesis if $f \rhd g$, and as we saw before if $g \notin \Sigma_{theory}$;
- $g = f(t_1, \ldots, t_n)$ and for all i, $s_i \geq t_i$, so all t_i are terminating and $\boldsymbol{s} >_{prod} \boldsymbol{t}$; we complete with the second induction hypothesis.

Next, we observe the relation between $\equiv, >$ and $\rightarrow_{\mathcal{R}}$.

Lemma 8. *If $s \rightarrow_{\texttt{calc}} t$ then $s \equiv t$.*

Proof. By induction on the size of s. If the reduction occurs at the root, then $s \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$ and t is its value, so clearly $[\![s]\!] = [\![t]\!]$. If the reduction occurs in a subterm, we use case 1b and the induction hypothesis.

Lemma 9. *If $l \succ r\ [\varphi]$ and γ respects $l \rightarrow r\ [\varphi]$, then $l\gamma > r\gamma$.*

Proof. By a shared induction on the derivation of $l \succ r\ [\varphi]$ or $l \succeq r\ [\varphi]$ we prove two things:

- if $l \succ r\ [\varphi]$ and $\gamma(x)$ is a value for all $x \in Dom(\varphi)$ and $\varphi\gamma$ is valid, then $l\gamma > r\gamma$
- if $l \succeq r\ [\varphi]$ and $\gamma(x)$ is a value for all $x \in Dom(\varphi)$ and $\varphi\gamma$ is valid, then $l\gamma \geq r\gamma$

Suppose first that $l \succeq r\ [\varphi]$ by case 1a, so l and r are terms with the same sort in $\mathcal{T}erms(\Sigma_{theory}, Var(\varphi))$ and $\varphi \Rightarrow (l_i = r_i \vee l_i \sqsupset r_i)$ is valid. Validity of this formula implies that it holds for γ (after all, all variables in the formula are in $Dom(\varphi)$, so are mapped to values by γ), so since $\varphi\gamma$ is true, we must have validity of $l_i\gamma = r_i\gamma \vee l_i\gamma \sqsupset r_i\gamma$. That is, either $[\![l_i\gamma]\!] = [\![r_i\gamma]\!]$ holds, in which case $l_i\gamma \equiv r_i\gamma$, or $[\![l_i\gamma]\!] > [\![r_i\gamma]\!]$, in which case $l_i\gamma > r_i\gamma$.

Alternatively, suppose $l \succeq r\ [\varphi]$ by case 1b, so $l = f(l_1, \dots, l_n)$ and $r = g(r_1, \dots, r_n)$ and each $l_i \succeq r_i\ [\varphi]$. By the induction hypothesis, each $l_i\gamma \geq r_i\gamma$. If all these are $\equiv$-equivalences, then $l\gamma \equiv r\gamma$ by case 1b of the definition of $\equiv$ / $>$, otherwise $l\gamma > r\gamma$ by case 2(b)iii.

If $l \succeq r\ [\varphi]$ because $l \succ r\ [\varphi]$, then immediately $l\gamma > r\gamma$ by the induction hypothesis.

We move on to the case where $l \succ r\ [\varphi]$. If this was derived by case 2a, then validity of $\varphi \Rightarrow l \sqsupset r$ implies that $[\![l\gamma]\!] > [\![r\gamma]\!]$, so we obtain that $l\gamma > r\gamma$ by case 2a of the definition of $>$. In each of the cases 2(b)i, 2(b)ii or 2(b)iii, we use the induction hypothesis and the corresponding case in the definition of $>$. Finally, if $l \succ r\ [\varphi]$ by case 2(b)iv, then $r \in Var(\varphi)$ so $r\gamma$ is a value, so $l\gamma > r\gamma$ by case 2(b)ii.

Now, we are pretty much ready to obtain Theorem 5, but need one last step: monotonicity of $>$. Unfortunately, $>$ is not actually monotonic. For example, $2 > 1$ but $3 - 2 \not> 3 - 1$. Fortunately, we do not need full monotonicity.

Lemma 10. *If $s > t$ and $s \notin \mathcal{T}erms(\Sigma_{theory}, \emptyset)$, then $C[s] > C[t]$ for any context C.*

Proof. We prove this by induction on the size of C, noting that whatever C is, $C[s] \notin \mathcal{T}erms(\Sigma_{theory}, \emptyset)$. We use cases 2(b)ii or 2c if $t \in \mathcal{T}erms(\Sigma_{theory}, \emptyset)$, or cases 2(b)iii and 2c otherwise.

Now, we are ready to combine the results and prove that the recursive path ordering can indeed be used for termination!

Theorem 5. Given an LCTRS $\mathcal{R}$ such that for all $l \to r\ [\varphi] \in \mathcal{R}$ we can prove $l \succ r\ [\varphi]$ for some choice of $\sqsupset_\iota$ and $\rhd$. Such an LCTRS is terminating.

Proof. We will prove that $s \to_{\mathtt{rule}} t$ implies $s > t$ and $s \to_{\mathtt{calc}} t$ implies $s \equiv t$. Since $>$ is well-founded, $\equiv$ and $>$ are compatible, and $\to_{\mathtt{calc}}$ is evidently terminating (so any infinite reduction must have infinitely many $\to_{\mathtt{rule}}$ steps), we are done. So suppose $s \to_{\mathtt{rule}} t$, that is, $s = C[l\gamma]$ and $t = C[r\gamma]$ for some context C, rule $l \to r\ [\varphi]$ and substitution γ which respects φ. By Lemma 9, the property $l \succeq r\ [\varphi]$ implies $l\gamma > r\gamma$. Since the root symbol of l must be in $\Sigma_{terms} \setminus \Sigma_{theory}$, we have by Lemma 10 that $C[l\gamma] > C[r\gamma]$.