Loop Detection by Logically Constrained Term Rewriting

Naoki Nishida[1] and Sarah Winkler[2]

[1] Department of Computing and Software Systems
Graduate School of Informatics, Nagoya University, Japan
nishida@i.nagoya-u.ac.jp
[2] Department of Computer Science
University of Innsbruck, Innsbruck, Austria
sarah.winkler@uibk.ac.at

Abstract. Logically constrained rewrite systems constitute a very general rewriting formalism that can capture simplification processes in various domains as well as computation in imperative programs. In both of these contexts, nontermination is a critical source of errors. We present new criteria to find loops in logically constrained rewrite systems which are implemented in the tool Ctrl. We illustrate the usefulness of these criteria in three example applications: to find loops in LLVM peephole optimizations, to detect looping program executions of C programs, and to establish nontermination of integer transition systems.

1 Introduction

Rewriting in presence of side constraints captures simplification processes in various areas, such as expression rewriting in compilers, theorem provers, or SMT solvers [11, 14, 16]. But also computations in an imperative program can be seen as rewrite sequences according to a constrained rewrite system describing the control flow graph [7]. In both cases the imposed side constraints can typically be expressed as formulas over a decidable logic. *Logically constrained rewrite systems* (LCTRSs) [12] formalize a very general rewriting mechanism that can express both of these settings, as well as earlier formalisms of constrained rewriting (cf. [12]). Side constraints of LCTRSs can be formulated over an arbitrary logic, though their application for practical analysis tasks requires decidability of the logic under consideration, typically by means of an SMT solver. But thanks to the impressive progress of SMT solving in the last two decades, we can use theories including, for instance, integer as well as bitvector arithmetic and arrays. This renders LCTRSs a powerful analysis tool in a wide range of areas, including program verification [7].

Termination is a key property of simplification and computation processes, and loops the most common violation thereof. We consider an example from the field of compiler optimizations.

Example 1. The Instcombine pass in the LLVM compilation suite performs *peephole optimizations* to simplify expressions in the intermediate representation. The

current optimization set contains over 1000 simplification rules to e.g. replace multiplications by shifts or perform bitwidth changes. About 500 of them have recently been translated into the domain-specific language Alive. The following simplification is an example rule in this format.

```
Name: MulDivRem 9
Pre: C < 0 && isPowerOf2(abs(C))
%Op0 = sub %Y, %X
%r = mul %Op0, C
  =>
%sub = sub %X, %Y
%r = mul %sub, abs(C)
```

It consists of a precondition labelled `Pre`, a left-hand side (the expression before the arrow `=>`), and the right-hand side (the expression after the arrow). Both expressions are given as a set of variable assignments. The last variable on each side is the root variable, in this case `%r`, which identifies the pattern to be replaced. Denoted as an LCTRS rule, this simplification reads as follows:

$$\mathsf{mul}(\mathsf{sub}(y,x),c) \to \mathsf{mul}(\mathsf{sub}(x,y),\mathsf{abs}(c)) \ [c <_s \#\mathsf{x0} \wedge \mathsf{isPowerOf2}(\mathsf{abs}(c))] \quad (1)$$

where the side constraint is formulated over bitvector arithmetic.

The Instcombine optimization suite is community-maintained, and unintended interplay of rules may occur. For instance, rule (1) admits the following loop:

$$\mathsf{mul}(\mathsf{sub}(x,x),\#\mathsf{x8000}) \to \mathsf{mul}(\mathsf{sub}(x,x),\mathsf{abs}(\#\mathsf{x8000})) \to \mathsf{mul}(\mathsf{sub}(x,x),\#\mathsf{x8000})$$

In this paper we present new criteria to recognize loops in LCTRSs. We implemented them in the Constrained Rewrite tooL Ctrl [13], which can now for instance detect the loop shown in Example 1. In order to illustrate the usefulness of these criteria, we discuss applications in three example domains: (1) finding loops in the Instcombine optimization suite, (2) detecting loops in C programs, and (3) establishing nontermination of integer transition systems.

The remainder of this paper is structured as follows. In Section 2 we recall preliminaries about logically constrained rewrite systems. We present our nontermination criteria in Section 3. Afterwards, we outline our implementation within the tool Ctrl in Section 4, and report on detecting loops in some example application areas in Section 5. In Section 6 we conclude.

2 Preliminaries

We assume familiarity with the basic notions of term rewrite systems [1], but briefly recapitulate the notion of logically constrained rewriting [7, 12] that our approach is based on.

We consider a sorted signature $\mathcal{F} = \mathcal{F}_{\text{terms}} \cup \mathcal{F}_{\text{theory}}$ such that $\mathcal{T}(\mathcal{F},\mathcal{V})$ denotes the set of terms over this signature. Here symbols in $\mathcal{F}_{\text{terms}}$ are called term symbols, while $\mathcal{F}_{\text{theory}}$ contains logical symbols. A substitution σ is a

mapping from variables to terms. As usual, we write $t\sigma$ for the application of σ to a term t. Terms over logical symbols are assumed to have a fixed semantics. To this end, we assume a mapping $\mathcal{I}$ that assigns to every sort ι occurring in $\mathcal{F}_{\text{theory}}$ a carrier set $\mathcal{I}(\iota)$, and an interpretation $\mathcal{J}$ that assigns to every symbol $f \in \mathcal{F}_{\text{theory}}$ a function $f_{\mathcal{J}}$ of appropriate sort. Moreover, for every sort ι occurring in $\mathcal{F}_{\text{theory}}$ we assume a set $\mathcal{V}\mathsf{al}_\iota \subseteq \mathcal{F}_{\text{theory}}$ of value symbols, such that all $c \in \mathcal{V}\mathsf{al}_\iota$ are constants of sort ι and $\mathcal{J}$ constitutes a bijective mapping between $\mathcal{V}\mathsf{al}_\iota$ and $\mathcal{I}(\iota)$. Hence there exists a constant symbol for every value in the carrier set. The interpretation $\mathcal{J}$ naturally extends to an interpretation of ground terms, by setting $[f(t_1, \ldots, t_n)]_{\mathcal{J}} = f_{\mathcal{I}}([t_1]_{\mathcal{J}}, \ldots, [t_n]_{\mathcal{J}})$. In this way every ground term has a unique value. We demand that theory symbols and term symbols overlap only on values, i.e., $\mathcal{F}_{\text{terms}} \cap \mathcal{F}_{\text{theory}} \subseteq \mathcal{V}\mathsf{al}$ holds. A term in $\mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ is called a *logical* term. In particular we assume a sort bool such that $\mathcal{I}(\mathsf{bool}) = \mathbb{B} = \{\top, \bot\}$, $\mathcal{V}\mathsf{al}_{\mathsf{bool}} = \{\mathsf{true}, \mathsf{false}\}$, $\mathsf{true}_{\mathcal{I}} = \top$, and $\mathsf{false}_{\mathcal{I}} = \bot$. Logical terms of sort bool are called *constraints*. A constraint φ is *valid* if $[\varphi\gamma]_{\mathcal{J}} = \top$ for all substitutions γ with $\mathcal{D}\mathsf{om}(\gamma) \subseteq \mathcal{T}(\mathcal{V}\mathsf{al})$.

Logically Constrained Rewriting. A *constrained rewrite rule* is a triple $\ell \to r\ [\varphi]$ where $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, φ is a logical constraint, and $\mathsf{root}(\ell) \in \mathcal{F}_{\text{terms}} \setminus \mathcal{F}_{\text{theory}}$. If $\varphi = \mathsf{true}$ then the constraint is often omitted, and the rule is denoted as $\ell \to r$. A set of constrained rewrite rules is called a *logically constrained term rewrite system* (LCTRS for short).

We now define rewriting using constrained rewrite rules. To this end, a substitution σ is said to *respect* a constraint φ if $\varphi\sigma$ is valid and $\sigma(x) \in \mathcal{V}\mathsf{al}$ for all $x \in \mathcal{V}\mathsf{ar}(\varphi)$. A *calculation step* $s \to_{\mathsf{calc}} t$ satisfies $s = C[f(s_1, \ldots, s_n)]$ for some $f \in \mathcal{F}_{\text{theory}} \setminus \mathcal{V}\mathsf{al}$, $t = C[u]$, $s_i \in \mathcal{V}\mathsf{al}$ for all $1 \leqslant i \leqslant n$, and $u \in \mathcal{V}\mathsf{al}$ is the value symbol of $[f(s_1, \ldots, s_n)]_{\mathcal{J}}$. In this case $f(x_1, \ldots, x_n) \to y\ [y = f(x_1, \ldots, x_n)]$ is a *calculation rule*, where y is a variable different from $x_1, \ldots, x_n$. A *rule step* $s \to_{\ell \approx r\ [\varphi]} t$ satisfies $s = C[\ell\sigma]$, $t = C[r\sigma]$, and σ respects φ. For an LCTRS $\mathcal{R}$, we also write $\to_{\mathsf{rule}, \mathcal{R}}$ to refer to the relation $\{\to_\alpha\}_{\alpha \in \mathcal{R}}$, and denote $\to_{\mathsf{calc}} \cup \to_{\mathsf{rule}, \mathcal{R}}$ by $\to_{\mathcal{R}}$. The subscript $\mathcal{R}$ is dropped if clear from the context.

Example 2. Consider the sort int (besides bool) and let $\mathcal{F}_{\text{theory}}$ consist of symbols $\cdot$, $+$, $-$, $\leqslant$, and $\geqslant$ as well as values n for all $n \in \mathbb{Z}$, with the usual interpretations on $\mathbb{Z}$. Let $\mathcal{F}_{\text{terms}} = \mathcal{V}\mathsf{al} \cup \{\mathsf{fact}\}$. The LCTRS $\mathcal{R}$ consisting of the rules

$$\mathsf{fact}(x) \to 1 \quad [x \leqslant 0] \qquad\qquad \mathsf{fact}(x) \to \mathsf{fact}(x-1) \cdot x \quad [x - 1 \geqslant 0]$$

admits the following rewrite steps:

$$\begin{aligned}
\mathsf{fact}(2) &\to_{\mathsf{rule}} \mathsf{fact}(2-1) \cdot 2 && && (\text{as } 2 - 1 \geqslant 0 \text{ is valid})\\
&\to_{\mathsf{calc}} \mathsf{fact}(1) \cdot 2 && \to_{\mathsf{rule}} (\mathsf{fact}(1-1) \cdot 1) \cdot 2 && (\text{as } 1 - 1 \geqslant 0 \text{ is valid})\\
&\to_{\mathsf{calc}} (\mathsf{fact}(0) \cdot 1) \cdot 2 && \to_{\mathsf{rule}} (1 \cdot 1) \cdot 2 && (\text{as } 0 \leqslant 0 \text{ is valid})\\
&\to^+_{\mathsf{calc}} 2
\end{aligned}$$

An LCTRS $\mathcal{R}$ is *terminating* if $\to_{\mathcal{R}}$ is well-founded. A *loop* is a rewrite sequence of the form $t \to^+_{\mathcal{R}} C[t\sigma]$. Due to the sequence $t \to^+_{\mathcal{R}} C[t\sigma] \to^+_{\mathcal{R}} C^2[t\sigma^2] \to^+_{\mathcal{R}}$

... existence of a loop implies nontermination. For example, a rewrite rule $\mathsf{f}(x, y) \to \mathsf{h}(\mathsf{f}(-x, \mathsf{g}(y)))$ $[x \geqslant 0]$ gives rise to the loop

$$\mathsf{f}(0, y) \to_{\mathsf{rule}} \mathsf{h}(\mathsf{f}(-0, \mathsf{g}(y))) \to_{\mathsf{calc}} \mathsf{h}(\mathsf{f}(0, \mathsf{g}(y))) \to_{\mathsf{rule}} \mathsf{h}(\mathsf{h}(\mathsf{f}(-0, \mathsf{g}(\mathsf{g}(y))))) \to_{\mathsf{calc}} \dots$$

Rewriting Constrained Terms. In addition to the notion of rewriting defined so far, it is for the sake of analysis convenient to define a notion of rewriting on *constrained* terms.

A *constrained term* is a pair $s\ [\varphi]$ of a term s and a constraint φ. Two constrained terms $s\ [\varphi]$ and $t\ [\psi]$ are *equivalent*, denoted by $s\ [\varphi] \sim t\ [\psi]$, if for every substitution γ respecting φ there is some substitution δ that respects ψ such that $s\gamma = t\delta$, and vice versa. For example, $\mathsf{fact}(x) \cdot x\ [x = 1 \wedge x < y] \sim \mathsf{fact}(1) \cdot y\ [y > 0 \wedge y < 2]$ holds, but these terms are not equivalent to $\mathsf{fact}(x) \cdot y\ [x = y]$ or $\mathsf{fact}(1)\ [\mathsf{true}]$.

Definition 1. *Let $\mathcal{R}$ be a set of constrained rewrite rules.*

- *A* calculation step $s\ [\varphi] \to_{\mathsf{calc}} t\ [\varphi \wedge x = f(s_1, \dots, s_n)]$ *satisfies* $s = C[f(s_1, \dots, s_n)]$ *for some* $f \in \mathcal{F}_{\mathrm{theory}} \setminus \mathcal{F}_{\mathrm{terms}}$ *and* $t = C[x]$ *such that* $s_1, \dots, s_n \in \mathcal{V}\mathsf{ar}(\varphi) \cup \mathcal{V}\mathsf{al}$ *and x is a fresh variable.*
- *A constrained rewrite rule* $\alpha\colon \ell \to r\ [\psi]$ *admits a* rule step $s\ [\varphi] \to_\alpha t\ [\varphi]$ *if* φ *is satisfiable,* $s = C[\ell\sigma]$, $t = C[r\sigma]$, $\sigma(x) \in \mathcal{V}\mathsf{al} \cup \mathcal{V}\mathsf{ar}(\varphi)$ *for all* $x \in \mathcal{V}\mathsf{ar}(\psi)$, *and* $\varphi \Rightarrow \psi\sigma$ *is valid.*

Given an LCTRS $\mathcal{R}$, we again write $\to_{\mathsf{rule},\mathcal{R}}$ for $\{\to_\alpha\}_{\alpha\in\mathcal{R}}$. The main rewrite relation $\to_\mathcal{R}$ on constrained terms is defined as $\sim \cdot (\to_{\mathsf{calc}} \cup \to_{\mathsf{rule},\mathcal{R}}) \cdot \sim$.

Example 3. Consider the LCTRS from Example 2, the constraint $\varphi = x \geqslant 1 \wedge y \geqslant 0$, and let z be a fresh variable. Then the following rewrite steps are possible:

$$\mathsf{fact}(x + y)\ [\varphi] \to_{\mathsf{rule}} \mathsf{fact}(x + y - 1) \cdot (x + y)\ [\varphi]$$
$$\mathsf{fact}(x + y)\ [\varphi] \to_{\mathsf{calc}} \mathsf{fact}(z)\ [\varphi \wedge z = x + y]$$

Narrowing Constrained Terms. We next define narrowing on constrained terms (cf. [4, 19]).

Definition 2. *A constrained rewrite rule* $\alpha\colon \ell \to r\ [\psi]$ *admits a narrowing step* $s\ [\varphi] \rightsquigarrow^\mu_{\alpha,p} t\ [\varphi']$ *if* $s = s[s']_p$, *the terms s' and ℓ are unifiable with mgu μ, the resulting term is* $t = (s[r]_p)\mu$, $\varphi' = (\varphi \wedge \psi)\mu$, *and φ' is satisfiable.*

If irrelevant or clear from the context, we omit the position p and substitution μ in the notation $\rightsquigarrow^\mu_{\alpha,p}$. Given an LCTRS $\mathcal{R}$, we write $\rightsquigarrow_\mathcal{R}$ for $\{\to_\alpha\}_{\alpha\in\mathcal{R}}$. We also write $s\ [\varphi]\ {}^\mu_\alpha\!\!\leftsquigarrow t\ [\varphi']$ if $\alpha\colon \ell \to r\ [\psi]$ admits a step $t\ [\varphi'] \rightsquigarrow^\mu_{r\to\ell\ [\psi]} s\ [\varphi]$.

Lemma 1. *If* $s\ [\varphi] \rightsquigarrow^\mu_{\alpha,p} t\ [\varphi']$ *then* $s\mu\ [\varphi\mu] \to_{\alpha,p} t\ [\varphi']$.

3 Loop Criteria

In this section we discuss criteria to detect looping derivations in LCTRSs. In unconstrained rewriting, given a TRS $\mathcal{R}$, tools commonly detect loops by searching for rewrite sequences of the form $t \to_{\mathcal{R}}^{+} C[t\sigma]$. In constrained rewriting, a sequence $t\ [\varphi] \to_{\mathcal{R}}^{+} C[t\sigma]\ [\psi]$ does not necessarily imply a loop: this depends on whether the constraints remain satisfied after repeated execution of the respective rewrite steps. In this section we consider a rewrite sequence $t\ [\varphi] \to_{\mathcal{R}}^{+} C[t\sigma]\ [\psi]$ such that $\psi \implies \varphi$ is valid, which we abbreviate by $t \to_{\psi,\mathcal{R}}^{+} C[t\sigma]$, and look for sufficient criteria such that this rewrite sequence gives rise to a loop.

The following criterion was already given in [17].

Lemma 2. *Let $\mathcal{R}$ be an LCTRS, and ψ a logical constraint. If $t \to_{\psi,\mathcal{R}}^{+} C[t\sigma]$ for some term t, context C, and substitution σ such that $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ for all $x \in \mathcal{V}\mathsf{ar}(\psi)$, ψ is satisfiable, and $\psi \implies \psi\sigma$ is valid. Then $\mathcal{R}$ is non-terminating.*

As a nontermination criterion, Lemma 2 has the disadvantage that it cannot detect loops which occur only for *specific* input values, such as the loop from Example 1. We next propose two criteria which remedy this shortcoming.

Lemma 3. *Let $\mathcal{R}$ be an LCTRS, and ψ a logical constraint. Suppose that $t \to_{\psi,\mathcal{R}}^{+} C[t\sigma]$ for some term t, context C, and substitution σ such that $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ for all $x \in \mathcal{V}\mathsf{ar}(\psi)$, and $\psi \wedge \bigwedge_{y \in \mathcal{D}\mathsf{om}(\sigma)} y = y\sigma$ is satisfiable by some assignment α.*

Then the loop $t\alpha \to_{\mathcal{R}}^{+} C[t\alpha\sigma]$ witnesses nontermination of $\mathcal{R}$.

Proof. Suppose $\psi \wedge \bigwedge_{y \in \mathcal{D}\mathsf{om}(\sigma)} y = y\sigma$ is satisfied by an assignment α. Thus $\psi\alpha$ is valid, and $[t\alpha]_{\mathcal{J}} = [t\sigma\alpha]_{\mathcal{J}}$. So there is a loop $t\alpha \to_{\mathcal{R}}^{+} C[t\sigma\alpha] \to_{\mathsf{calc}}^{*} C[t\alpha] \to_{\mathcal{R}}^{+} \cdots$. □

Example 4. Returning to Example 1, the two rewrite steps

$$\mathsf{mul}(\mathsf{sub}(y, x), c)\ [\varphi] \to_{\mathsf{rule}} \mathsf{mul}(\mathsf{sub}(x, y), \mathsf{abs}(c))\ [\varphi] \to_{\mathsf{calc}} \mathsf{mul}(\mathsf{sub}(x, y), c')\ [\psi]$$

constitute a loop candidate, where $\varphi = c <_s \#\mathsf{x0} \wedge \mathsf{isPowerOf2}(\mathsf{abs}(c))$ and $\psi = \varphi \wedge c' = \mathsf{abs}(c)$. We thus have $t\ [\psi] \to_{\mathcal{R}}^{+} C[t\sigma]\ [\psi]$ for $t = \mathsf{mul}(\mathsf{sub}(y, x), c)$, $C = \Box$, and $\sigma = \{y \mapsto x,\ c \mapsto c'\}$, such that $\sigma(x)$ is a logical term for all x in ψ. The formula $\psi \wedge c = c'$ is satisfiable by the assignment $\alpha(c) = \alpha(c') = \#\mathsf{x8000}$. This assignment corresponds to the loop given in Example 1:

$$\mathsf{mul}(\mathsf{sub}(x, x), \#\mathsf{x8000}) \to \mathsf{mul}(\mathsf{sub}(x, x), \mathsf{abs}(\#\mathsf{x8000})) \to \mathsf{mul}(\mathsf{sub}(x, x), \#\mathsf{x8000})$$

This criterion is rather restrictive in that it demands that the starting term occurs again as a subterm after some steps (modulo calculations). The following result adds some flexibility in this respect.

Lemma 4. *Let $\mathcal{R}$ be an LCTRS, and ψ a logical constraint. Suppose that $t \to_{\psi,\mathcal{R}}^{+} C[t\sigma]$ for some term t, context C, and substitution σ such that $\sigma(x) \in$*

$\mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ *for all* $x \in \mathcal{V}\mathsf{ar}(\psi)$. *Suppose* $\mathcal{D}\mathsf{om}(\sigma) = \{y_1, \ldots, y_n\}$, *and let* $\rho = \{y_1 \mapsto z_1, \ldots, y_n \mapsto z_n\}$ *be a renaming to fresh variables* $z_1, \ldots, z_n$.

If $\forall y_1 \ldots y_n.(\psi \Longrightarrow \psi\sigma) \wedge \psi\rho$ *is satisfiable by* α *then the loop* $t\rho\alpha \to^+_{\mathcal{R}} C[t\sigma\rho\alpha]$ *witnesses nontermination of* $\mathcal{R}$.

Proof. Suppose that $\forall y_1 \ldots y_n.(\psi \Longrightarrow \psi\sigma) \wedge \psi\rho$ is satisfied by some assignment α. So in particular $\psi\rho\alpha$ holds. Since moreover $\forall y_1 \ldots y_n.(\psi \Longrightarrow \psi\sigma)$ is valid and $\mathcal{D}\mathsf{om}(\sigma) = \{y_1, \ldots, y_n\}$ we have validity of $\forall y_1 \ldots y_n.(\psi\sigma^k \Longrightarrow \psi\sigma^{k+1})$ for all $k \geqslant 0$. Therefore $\psi\sigma^k\rho\alpha$ is valid for all $k \geqslant 0$ such that

$$t\rho\alpha \to^+_{\mathcal{R}} C[t\sigma\rho\alpha] \to^+_{\mathcal{R}} C^2[t\sigma^2\rho\alpha] \to^+_{\mathcal{R}} \cdots$$

is indeed a loop.

Example 5. Consider the following LCTRS with side constraints over the integers:

$$\mathsf{f}(x, y) \to \mathsf{f}(x + 1 - y, y) - 1 \ [y \leqslant 0 \wedge x \geqslant 0]$$

The rule constitutes itself a loop candidate: We have $t\ [\psi] \to^+_{\mathcal{R}} C[t\sigma]\ [\psi]$ for $t = \mathsf{f}(x, y)$, $C = \Box - 1$, and $\sigma = \{x \mapsto x + 1 - y\}$ with $\mathcal{D}\mathsf{om}(\sigma) = \{x\}$. The formula

$$\forall x\ (y \leqslant 0 \wedge x \geqslant 0 \implies (y \leqslant 0 \wedge (x + 1 - y) \geqslant 0)) \wedge y \leqslant 0 \wedge z \geqslant 0$$

is satisfied e.g. by the assignment $\alpha(y) = \alpha(z) = 0$. Thus the following loop is recognized:

$$\mathsf{f}(0, 0) \to_{\mathcal{R}} \mathsf{f}(0 + 1 - 0, 0) \to^+_{\mathsf{calc}} \mathsf{f}(1, 0) \to_{\mathcal{R}} \mathsf{f}(1 + 1 - 0, 0) \to^+_{\mathsf{calc}} \mathsf{f}(2, 0) \to \cdots$$

Note that this loop is not captured by the criteria demanded in Lemmas 2 and 3.

4 Implementation

We extended the Ctrl [13] tool by nontermination techniques that exploit the criteria presented in Section 3. Optionally a starting term can be given, i.e., two modes are supported:

(a) Given an LCTRS $\mathcal{R}$, find a loop $t\ [\varphi] \to^+_{\mathcal{R}} C[t\sigma]\ [\varphi]$ such that φ is satisfiable.
(b) Given an LCTRS $\mathcal{R}$ and a starting term u, find a loop reachable from u, i.e., a sequence $u\ [\mathsf{true}] \to^*_{\mathcal{R}} t\ [\varphi] \to^+_{\mathcal{R}} C[t\sigma]\ [\varphi]$ such that φ is satisfiable.

An input file in the `ctrs` format specifies the logical theory to be used, the signature, the rewrite rules, and a query to fix the problem statement for Ctrl. To support nontermination analysis, we specify `loops` as a query in input files:

```
QUERY loops t
```

where the optional argument t is a term from which a loop should be reachable. Ctrl offers theory specifications for integers and arrays, and we added bitvectors of fixed sizes for the present work. Alternatively a user-defined theory specification can be used.

We next describe how loops are detected in our implementation. We call a *sequence tuple* a tuple of the form $(s \to t\ [\psi], S)$ where $S = [(\alpha_1, p_1), \ldots, (\alpha_k, p_k)]$, $s \to t\ [\psi]$ is a constrained rewrite rule, α_i is a rule of the form $\ell_i \to r_i\ [\varphi_i]$ in $\mathcal{R} \cup \mathcal{R}_{\mathsf{calc}}$ and p_i are positions for all $0 \leqslant i \leqslant k$ such that there is the rewrite sequence

$$s\ [\psi] \to_{\alpha_1,p_i} \cdots \to_{\alpha_k,p_k} t\ [\psi].$$

In either of the modes (a) and (b), we proceed in five steps as follows.

(0) Using the dependency pair (DP) framework present in Ctrl [12], the problem is split into strongly connected components of the dependency graph. This results in a set of DP problems of the form $(\mathcal{P}, \mathcal{R})$, where $\mathcal{P}$ is a set of dependency pairs and $\mathcal{R}$ the given LCTRS. (Basically this amounts to splitting the problem into rules $\mathcal{P}$ that are applied at the root of a term and rules $\mathcal{R}$ that can be applied below. Then potential cycles in the call graph are identified, and only upon these the analysis continues; see [12] for details.) The following steps are then performed for each of these DP problems:

(1) A set of initial sequence tuples T_0 is determined. In case of (a), we take the set of all single-step sequences $(\ell \to r\ [\varphi], [(\ell \to r\ [\varphi])])$ such that $\ell \to r\ [\varphi] \in \mathcal{P}$.

(2) Given tuples T_i, we set T_{i+1} to the set

$$\{(s\tau \to u\ [\varphi'],\ S_f) \mid (s \to t\ [\psi], S) \in T_i,\ \beta \in \mathcal{P} \cup \mathcal{R} \text{ and } t\ [\varphi] \rightsquigarrow^{\tau}_{\beta,q} u\ [\varphi']\}$$

for forward unfolding where $S_f = S +\!\!+ [(\beta, q)]$, and to

$$\{(u\tau \to t\ [\varphi'],\ S_b) \mid (s \to t\ [\psi], S) \in T_i,\ \beta \in \mathcal{P} \cup \mathcal{R} \text{ and } u\ [\varphi]\ {}^{\tau}_{\beta,q}\!\!\leftsquigarrow s\ [\varphi']\}$$

for backward unfolding, where $S_b = [(\beta, q)] +\!\!+ S$. Here $+\!\!+$ denotes list concatenation.

(3) Given $T = \bigcup_{i \leqslant n} T_i$ for some n, we call $s\mu\ [\psi\mu] \to^+_{\mathcal{R}} C[s\mu]\ [\psi\mu]$ a *loop candidate* if $(s \to C[s']\ [\psi], S) \in T$ such that s and s' are unifiable with mgu μ and $\psi\mu$ is satisfiable.

(4) We finally use Lemmas 2, 3, and 4 to check whether there are input values for which the loop candidates correspond to actual loops.

Since it is known that forward and backward unfolding are incomparable in general [18], both methods are supported. A link to the tool and its source code as well as example input files corresponding to the examples used in this paper can be obtained on-line.[3]

5 Applications

We now illustrate the loop support of Ctrl in three different application domains.

[3] `http://cl-informatik.uibk.ac.at/users/swinkler/lctrs_loops`

Table 1. Experimental results on the Instcombine LCTRSs.

	`add-sub`	`mul-div`	`shift`	`and-or`	`select`	`loops`	`all`
# rules	66	118	75	180	85	43	518
`fw` 3-loops	4	8	4	22	2	40	51
time (s)	16	80	9	3601	24	25	>32k
`bw` 3-loops	4	8	4	10	2	27	TO
time (s)	29	727	9	8400	21	24	TO

LLVM Instcombine Simplifications

We transformed the around 500 simplifications in the Alive language mentioned in Example 1 into LCTRSs using bitvector theory as background logic. These simplifications are split into domains. We tested Ctrl on the simplification sets for addition and subtraction, multiplication and division, shifts, bitwise logical operations, and select operations, as well as on their union. Table 1 summarizes our results. The columns refer to the different domains, and `loops` refers to the set of rules involved in all loops found in the work [15] discussed below. The rows indicate how many loops of length at most 3 were found by Ctrl using forward (`fw`) and backward (`bw`) unfolding, respectively, and how much time was required. In general forward unfolding seems to be more useful than backward unfolding. We remark that not all loops found can actually occur in the LLVM Instcombine pass since the rule set is applied with a particular strategy, such that certain optimizations can "shadow" other ones. Thus it needs to be checked by hand whether the detected potential loops can actually occur.

A dedicated tool `alive-loops` to detect loops in the Instcombine optimizations was presented in [15]. We briefly compare our criteria to their approach: First of all, we found the same loops with Ctrl that were exhibited by `alive-loops`, modulo combination and nesting of loops. But the loop check applied in `alive-loops` is different: It amounts to the search for a loop candidate $t \to^+_{\psi,\mathcal{R}} C[t\sigma]$ such that $\psi \implies \psi\sigma$ is satisfiable. While this is obviously a necessary condition it is in general not sufficient:

Example 6. As an (artificial) example, consider the constrained rewrite rule $\mathsf{and}(\#\mathsf{x0}, x) \to \mathsf{and}(\#\mathsf{x0}, x \gg_u \#\mathsf{x1})\ [x > \#\mathsf{x0}]$. It gives rise to a loop candidate $t \to^+_{\psi,\mathcal{R}} t\sigma$ where $\psi = x > \#\mathsf{x0}$, $t = \mathsf{and}(\#\mathsf{x0}, x)$, and $\sigma = \{x \mapsto x \gg_u \#\mathsf{x1}\}$. The constraint $\psi \implies \psi\sigma$ is satisfiable. But logically shifting x to the right will eventually result in a bit vector $\#\mathsf{x0000}$, hence no such loop exists. Indeed `alive-loops` finds a spurious loop in this example, but Ctrl does not.

Moreover `alive-loops` is limited in that it restricts to loop candidates which are not size-increasing. The shadowing problem mentioned above occurs as well.

Loops in Integer Transition Systems

Integer term rewriting has been introduced as a rewriting formalism which natively supports integer operations, to be applied to rewrite-based program analysis [6].

The integer transition system `Velroyen08-alternKonv.jar-obl-8` from the Termination Problem Database 9.0[4] corresponds to the following LCTRS:

$$\mathsf{f1\_0\_main}(x,y) \to \mathsf{f81\_0}(x',y') \quad [\, x > 0 \wedge y > -1 \wedge y = x' \,] \tag{1}$$
$$\mathsf{f81\_0}(x,y) \to \mathsf{f81\_0}(x',y') \quad [\, x < 0 \wedge x > -3 \wedge x + 2 = x' \,] \tag{2}$$
$$\mathsf{f81\_0}(x,y) \to \mathsf{f81\_0}(x',y') \quad [\, x > 0 \wedge x < 3 \wedge x - 2 = x' \,] \tag{3}$$
$$\mathsf{f81\_0}(x,y) \to \mathsf{f81\_0}(x',y') \quad [\, x < -2 \wedge x < -1 \wedge x < 0 \wedge -x - 2 = x' \,] \tag{4}$$
$$\mathsf{f81\_0}(x,y) \to \mathsf{f81\_0}(x',y') \quad [\, x > 2 \wedge -x + 2 = x' \,] \tag{5}$$
$$\mathsf{init}(x,y) \to \mathsf{f1\_0\_main}(x',y') \tag{6}$$

where the starting term is of the form $\mathsf{init}(x,y)$. It admits the following rewrite steps which contain a loop:

$$\mathsf{init}(1,1) \xrightarrow{(6)} \mathsf{f1\_0\_main}(1,1) \xrightarrow{(1)} \mathsf{f81\_0}(1,-1) \xrightarrow{(3)} \mathsf{f81\_0}(-1,0) \xrightarrow{(2)} \mathsf{f81\_0}(1,-1)$$

(where the arrows are decorated with the applied rule). Ctrl can easily show nontermination within less than 2 seconds by exploiting Lemma 3. This is also the case for the similar system `alternKonv_rec`, while in the Termination Competition 2017[5] both of these problems remained unsolved.

Loops in C Programs

Consider the following C program implementing binary search [10]:

```
int bsearch(int a[], int k, unsigned int lo, unsigned int hi) {
  unsigned int mid;
  while (lo < hi) {
    mid = (lo + hi)/2;
    if (a[mid] < k)
      lo = mid + 1;
    else if (a[mid] > k)
      hi = mid - 1;
    else
      return mid;
  }
  return -1;
}
```

It admits a loop for inputs `lo=1` and `hi=MAXINT` if `a[0] < k`. Abstracting from the array accesses, this program can be represented by the following LCTRS:

[4] `http://termination-portal.org/wiki/TPDB`
[5] `http://www.termination-portal.org/wiki/Termination_Competition_2017`

$$
\begin{aligned}
\mathsf{bsearch}(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1) &\to \mathsf{u}_1(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{rnd}_1)\\
\mathsf{u}_1(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_2(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_2(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_3(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,(\mathsf{lo}_1+\mathsf{hi}_1)\mathbin{/_u}\#\mathsf{x02}) \quad [\,\mathsf{lo}_1 <_u \mathsf{hi}_1\,]\\
\mathsf{u}_3(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_4(\mathsf{k}_1,(\mathsf{mid}_2+\#\mathsf{x01}),\mathsf{hi}_1,\mathsf{mid}_2) \quad [\,\mathsf{mid}_2 <_u \mathsf{k}_1\,]\\
\mathsf{u}_4(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_5(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_3(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_6(\mathsf{k}_1,\mathsf{lo}_1,(\mathsf{mid}_2-\#\mathsf{x01}),\mathsf{mid}_2) \quad [\,\mathsf{mid}_2 \geqslant \mathsf{k}_1 \wedge \mathsf{mid}_2 > \mathsf{k}_1\,]\\
\mathsf{u}_6(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_7(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_3(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{return}(\mathsf{mid}_2) \quad [\,\mathsf{mid}_2 \geqslant \mathsf{k}_1 \wedge \mathsf{mid}_2 \leqslant \mathsf{k}_1\,]\\
\mathsf{u}_7(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_8(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_5(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_9(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_8(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_9(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_9(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_10(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_{10}(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{u}_2(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2)\\
\mathsf{u}_2(\mathsf{k}_1,\mathsf{lo}_1,\mathsf{hi}_1,\mathsf{mid}_2) &\to \mathsf{return}(\#\mathsf{xff}) \quad [\,\mathsf{lo}_1 \geqslant_u \mathsf{hi}_1\,]
\end{aligned}
$$

Ctrl can prove existence of a loop that is reachable from a term of the form $\mathsf{bsearch}(x,y,l,h)$ below one second, using Lemma 3.

6 Conclusion

We presented new criteria to recognize loops in LCTRSs, and implemented these in the constrained rewrite tool Ctrl. In order to demonstrate applicability of such nontermination support, we investigated three example domains.

For the case of LLVM Instcombine optimizations, we confirmed all loops found by the tool `alive-loops` [15], and argued that in contrast to this previous work our criteria do not give rise to false positives. We moreover showed how Ctrl can be used to detect loops in a C program and in integer transition systems.

Extensive work on nontermination detection has been done in the past for both domains, c.f. [2, 5, 10] and [3, 9], for example. A thorough evaluation of our criteria by means of comparison with tools such as [2, 3, 9] is left for future work. Rather than claiming our implementation superior to other tools, we consider the work presented in this paper a proof of concept that nontermination criteria for LCTRSs are applicable to a wide range of domains. In contrast to tools designed for integer transition systems, C programs, or LLVM Instcombine optimizations, we can treat all these applications *uniformly* with our criteria: Due to the generality of LCTRSs, the same implementation can be applied to a variety of background theories such as integer or bitvector arithmetic or arrays.

In future work we want to investigate further application domains such as simplifications performed in the preprocessing phase of SMT solvers [8, 16]. Moreover, it would be interesting to find criteria for nonlooping nontermination of LCTRSs.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. C. Borralleras, M. Brockschmidt, D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Proving termination through conditional termination. In *Proc. 23rd TACAS*, volume 10205 of *LNCS*, pages 99–117, 2017.
3. M. Brockschmidt, B. Cook, S. Ishtiaq, H. Khlaaf, and N. Piterman. T2: temporal property verification. In *Proc. 22nd TACAS*, volume 9636 of *LNCS*, pages 387–393, 2016.
4. S. Falke and D. Kapur. A term rewriting approach to the automated termination analysis of imperative programs. In *Proc. 22nd CADE*, volume 5663 of *LNCS*, pages 277–293, 2009.
5. S. Falke, D. Kapur, and C. Sinz. Termination analysis of C programs using compiler intermediate languages. In *Proc. 22nd RTA*, volume 10 of *Leibniz International Proceedings in Informatics*, pages 41–50, 2011.
6. C. Fuhs, J. Giesl, M. Plücker, P. Schneider-Kamp, and S. Falke. Proving termination of integer term rewriting. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 32–47, 2009.
7. C. Fuhs, C. Kop, and N. Nishida. Verifying procedural programs via constrained rewriting induction. *ACM TOCL*, 18(2):14:1–14:50, 2017.
8. V. Ganesh, S. Berezin, and D. Dill. A decision procedure for fixed-width bit-vectors. Technical report, Stanford University, 2005.
9. J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Analyzing program termination and complexity automatically with AProVE. *JAR*, 58(1):3–31, 2017.
10. A. Gupta, T. Henzinger, R. Majumdar, A. Rybalchenko, and R-.G. Xu. Proving non-termination. *SIGPLAN Not.*, 43(1):147–158, 2008.
11. K. Hoder, Z. Khasidashvili, K. Korovin, and A. Voronkov. Preprocessing techniques for first-order clausification. In *Proc. 12th FMCAD*, pages 44–51, 2012.
12. C. Kop and N. Nishida. Term rewriting with logical constraints. In *Proc. 9th FroCoS*, volume 8152 of *LNAI*, pages 343–358, 2013.
13. C. Kop and N. Nishida. Constrained term rewriting tool. In *Proc. 20th LPAR*, volume 9450 of *LNAI*, pages 549–557, 2015.
14. N. Lopes, D. Menendez, S. Nagarakatte, and J. Regehr. Provably correct peephole optimizations with Alive. In *Proc. 36th PLDI*, pages 22–32, 2015.
15. D. Menendez and S. Nagarakatte. Termination-checking for LLVM peephole optimizations. In *Proc. 38th International Conference on Software Engineering*, pages 191–202, 2016.
16. A. Nadel. Bit-vector rewriting with automatic rule generation. In *Proc. 16th CAV*, pages 663–679, 2014.
17. N. Nishida, M. Sakai, and T. Hattori. On disproving termination of constrained term rewriting systems. In *Proc. 11th WST*, page 5 pages, 2010.
18. É. Payet. Loop detection in term rewriting using the eliminating unfoldings. *Theoretical Computer Science*, 403(2–3):307–327, 2008.
19. C. Rocha, J. Meseguer, and C. A. Muñoz. Rewriting modulo SMT and open system analysis. *J. Log. Algebr. Meth. Program.*, 86(1):269–297, 2017.