





Loop Detection by Logically Constrained Term Rewriting

Naoki Nishida¹  and Sarah Winkler² 

¹ Department of Computing and Software Systems, Graduate School of Informatics, Nagoya University, Nagoya, 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 executions of C programs, and to establish nontermination of integer transition systems.

Keywords: Constrained rewriting · Nontermination · Loops

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, 15, 17]. 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 term 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 employ an arbitrary first-order logic which contains propositional logic and equality, though their application for practical analysis tasks requires decidability of the logic under consideration. 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].

This work is partially supported by JSPS KAKENHI Grant Number JP18K11160 and FWF (Austrian Science Fund) project T789.

© Springer Nature Switzerland AG 2018

R. Piskac and P. Rümmer (Eds.): VSTTE 2018, LNCS 11294, pp. 309–321, 2018.

https://doi.org/10.1007/978-3-030-03592-1_18

Termination is a key property of simplification and computation processes, and loops are 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 [14, 15]. 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 defined by a sequence of variable assignments. The last variable on each side—in this case `%r`—identifies the pattern to be replaced. This simplification can also be represented by the following LCTRS rule, using a side constraint over bitvector arithmetic:

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

The `Instcombine` optimization suite is community-maintained, and unintended interference of rules may occur. For instance, for 16-bit integers where `#x8000` is the smallest representable integer value, Rule (1) in combination with constant folding admits the following loop since `abs(#x8000)` evaluates to `#x8000`:

$$\text{mul}(\text{sub}(x, x), \#x8000) \rightarrow \text{mul}(\text{sub}(x, x), \text{abs}(\#x8000)) \rightarrow \text{mul}(\text{sub}(x, x), \#x8000)$$

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 our 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 Sect. 2 we recall preliminaries about logically constrained rewrite systems. We present our nontermination criteria in Sect. 3. Afterwards, we outline our implementation within the tool `Ctrl` in Sect. 4, and report on detecting loops in some example application areas in Sect. 5. In Sect. 6 we conclude.

2 Preliminaries

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

We consider an infinite set of variables \mathcal{V} and 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. Symbols in $\mathcal{F}_{\text{terms}}$ are called *term symbols*, while $\mathcal{F}_{\text{theory}}$ contains *theory symbols*. A term in $\mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ is called a theory term. For a non-variable term $t = f(t_1, \dots, t_n)$, we write $\text{root}(t)$ to obtain the top-most symbol f . A position p is an integer sequence used to identify subterms of a given term. The *subterm* of t at position p is defined as $t|_e = t$, and if $t = f(t_1, \dots, t_n)$ then $t|_{ip} = t_i|_p$. The result of replacing the subterm of a term t at position p by s is denoted $t[s]_p$. A *context* C is a term with a single occurrence of a designated constant \square , and we write $C[t]$ to denote the term obtained by replacing \square in C by t . A *substitution* σ is a mapping from variables to terms. We write $\text{Dom}(\sigma)$ and $\text{Ran}(\sigma)$ for its domain and range, while $t\sigma$ denotes the application of σ to a term t .

Terms over logical symbols are associated with 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 $f_{\mathcal{J}}$. For every sort ι occurring in $\mathcal{F}_{\text{theory}}$ we assume a set $\mathcal{Val}_{\iota} \subseteq \mathcal{F}_{\text{theory}}$ of *value* symbols, such that all $c \in \mathcal{Val}_{\iota}$ are constants of sort ι and \mathcal{J} constitutes a bijective mapping between \mathcal{Val}_{ι} and $\mathcal{I}(\iota)$. Hence there exists a constant symbol for every value in the carrier set. We write \mathcal{Val} for $\bigcup_{\iota} \mathcal{Val}_{\iota}$. The interpretation \mathcal{J} naturally extends to theory terms without variables by setting $[f(t_1, \dots, t_n)]_{\mathcal{J}} = f_{\mathcal{I}}([t_1]_{\mathcal{J}}, \dots, [t_n]_{\mathcal{J}})$. Theory symbols and term symbols are supposed to overlap only on values, i.e., $\mathcal{F}_{\text{terms}} \cap \mathcal{F}_{\text{theory}} \subseteq \mathcal{Val}$ holds. We assume a sort **bool** such that $\mathcal{I}(\text{bool}) = \mathbb{B} = \{\top, \perp\}$ with values $\mathcal{Val}_{\text{bool}} = \{\text{true}, \text{false}\}$ such that $\text{true}_{\mathcal{J}} = \top$, and $\text{false}_{\mathcal{J}} = \perp$. Moreover we consider a theory symbol \approx for equality. Theory terms of sort **bool** are called *constraints*. A substitution σ which satisfies $\sigma(x) \in \mathcal{Val}$ for all $x \in \text{Dom}(\sigma)$ is also called an *assignment*. A constraint φ is *valid* if $[\varphi\gamma]_{\mathcal{J}} = \top$ for all assignments γ , and *satisfiable* if $[\varphi\gamma]_{\mathcal{J}} = \top$ for some assignment γ .

Logically Constrained Rewriting. We consider constrained rewriting as developed in [7, 12]. A *constrained rewrite rule* is a triple $\ell \rightarrow r \ [\varphi]$ where $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $\ell \notin \mathcal{V}$, φ is a constraint, and $\text{root}(\ell) \in \mathcal{F}_{\text{terms}} \setminus \mathcal{F}_{\text{theory}}$. If $\varphi = \text{true}$ then the constraint is omitted, and the rule denoted as $\ell \rightarrow r$. A set of constrained rewrite rules is called a *logically constrained term rewrite system* (LCTRS for short).

In order to define rewriting using constrained rewrite rules, a substitution σ is said to *respect* a constraint φ if $\varphi\sigma$ is valid and $\sigma(x) \in \mathcal{Val}$ for all $x \in \text{Var}(\varphi)$. A *calculation step* $s \rightarrow_{\text{calc}} t$ satisfies $s = C[f(s_1, \dots, s_n)]$ for some $f \in \mathcal{F}_{\text{theory}} \setminus \mathcal{Val}$, $t = C[u]$, $s_i \in \mathcal{Val}$ for all $1 \leq i \leq n$, and $u \in \mathcal{Val}$ is the value symbol of $[f(s_1, \dots, s_n)]_{\mathcal{J}}$. In this case $f(x_1, \dots, x_n) \rightarrow y \ [y \approx f(x_1, \dots, x_n)]$ is a *calculation rule*, where y is a variable different from x_1, \dots, x_n . A *rule step* $s \rightarrow_{\ell \rightarrow r \ [\varphi]} t$ satisfies $s = C[\ell\sigma]$, $t = C[r\sigma]$, and σ respects φ . For an LCTRS

\mathcal{R} , we also write $\rightarrow_{\text{rule}, \mathcal{R}}$ to refer to the relation $\{\rightarrow_{\alpha}\}_{\alpha \in \mathcal{R}}$, and denote $\rightarrow_{\text{calc}} \cup \rightarrow_{\text{rule}, \mathcal{R}}$ by $\rightarrow_{\mathcal{R}}$. The subscript \mathcal{R} is dropped if clear from the context.

Example 2. Consider the sorts `int` and `bool`, and let $\mathcal{F}_{\text{theory}}$ consist of symbols \cdot , $+$, $-$, \leq , and \geq as well as values n for all $n \in \mathbb{Z}$, with the usual interpretations on \mathbb{Z} . Let $\mathcal{F}_{\text{terms}} = \text{Val} \cup \{\text{fact}\}$. The LCTRS \mathcal{R} consisting of the rules

$$\text{fact}(x) \rightarrow 1 \quad [x \leq 0] \quad \text{fact}(x) \rightarrow \text{fact}(x-1) \cdot x \quad [x-1 \geq 0]$$

admits the following rewrite steps:

$$\begin{aligned} \text{fact}(2) &\xrightarrow{\text{rule}} \text{fact}(2-1) \cdot 2 && (\text{as } 2-1 \geq 0 \text{ is valid}) \\ &\xrightarrow{\text{calc}} \text{fact}(1) \cdot 2 &\xrightarrow{\text{rule}} (\text{fact}(1-1) \cdot 1) \cdot 2 && (\text{as } 1-1 \geq 0 \text{ is valid}) \\ &\xrightarrow{\text{calc}} (\text{fact}(0) \cdot 1) \cdot 2 &\xrightarrow{\text{rule}} (1 \cdot 1) \cdot 2 && (\text{as } 0 \leq 0 \text{ is valid}) \\ &\xrightarrow{\text{calc}}^+ 2 \end{aligned}$$

An LCTRS \mathcal{R} is *terminating* if $\rightarrow_{\mathcal{R}}$ is well-founded. A *loop* is a rewrite sequence of the form $t \xrightarrow{\text{rule}}^+ C[t\sigma]$. Due to the sequence $t \xrightarrow{\text{rule}}^+ C[t\sigma] \xrightarrow{\text{rule}}^+ C^2[t\sigma^2] \xrightarrow{\text{rule}}^+ \dots$ existence of a loop implies nontermination. For example, a rewrite rule $f(x, y) \rightarrow h(f(-x, g(y)))$ $[x \geq 0]$ gives rise to the loop where $t = f(0, y)$, $C = h(\square)$, and $\sigma = \{y \mapsto g(y)\}$:

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

Rewriting Constrained Terms. The notion of rewriting for *unconstrained* terms considered so far is used to model the actual simplification and computation processes in practice. But for the sake of analysis it is convenient to also define a notion of rewriting on *constrained* terms, for instance to capture the composition of rewrite rules.

To that end, 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, $\text{fact}(x) \cdot x$ $[x = 1 \wedge x < y] \sim \text{fact}(1) \cdot y$ $[y > 0 \wedge y < 2]$ holds, but these terms are not equivalent to $\text{fact}(x) \cdot y$ $[x = y]$ or $\text{fact}(1)$ $[\text{true}]$. Next we define rewriting on constrained terms.

Definition 1

- A *calculation step* $s[\varphi] \xrightarrow{\text{calc}} t[\varphi \wedge x \approx f(s_1, \dots, s_n)]$ needs to satisfy $s = C[f(s_1, \dots, s_n)]$ for some $f \in \mathcal{F}_{\text{theory}} \setminus \mathcal{F}_{\text{terms}}$ and $t = C[x]$ such that $s_1, \dots, s_n \in \text{Var}(\varphi) \cup \text{Val}$ and x is a fresh variable.
- A constrained rewrite rule $\alpha: \ell \rightarrow r$ $[\psi]$ admits a *rule step* $s[\varphi] \rightarrow_{\alpha} t[\varphi]$ if φ is satisfiable, $s = C[\ell\sigma]$, $t = C[r\sigma]$, $\sigma(x) \in \text{Val} \cup \text{Var}(\varphi)$ for all $x \in \text{Var}(\psi)$, and $\varphi \Rightarrow \psi\sigma$ is valid.

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

For example, the LCTRS from Example 2 and the constraint $\varphi = x \geq 1 \wedge y \geq 0$ admit the rule step $\mathbf{fact}(x + y) [\varphi] \rightarrow_{\text{rule}} \mathbf{fact}(x + y - 1) \cdot (x + y) [\varphi]$, while $\mathbf{fact}(x + y) [\varphi] \rightarrow_{\text{calc}} \mathbf{fact}(z) [\varphi \wedge z \approx x + y]$ is a possible calculation step.

We next define narrowing on constrained terms (cf. the notion of *chains* [4]).

Definition 2. A constrained rewrite rule $\alpha: \ell \rightarrow r [\psi]$ admits a narrowing step $s [\varphi] \rightsquigarrow_{\alpha,p}^{\mu} t [\varphi']$ 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.

We also write $s [\varphi] \overset{\mu}{\alpha} \leftarrow t [\varphi']$ if $\alpha: \ell \rightarrow r [\psi]$ admits a step $t [\varphi'] \rightsquigarrow_{r \rightarrow \ell}^{\mu} s [\varphi]$. The following lemma shows the crucial correspondence between narrowing and rewriting, which ensures correctness of our loop detection shown in Sect. 4.

Lemma 1 (Lifting Lemma). *Suppose $\alpha: \ell \rightarrow r [\psi]$ admits a narrowing step $s [\varphi] \rightsquigarrow_{\alpha,p}^{\mu} t [\varphi']$, where $\varphi' = (\varphi \wedge \psi)\mu$. Then $s\mu [\varphi'] \rightarrow_{\alpha,p} t [\varphi']$.*

Proof. We have $s\mu|_p = \ell\mu$ and can perform a rewrite step because $\varphi' = (\varphi \wedge \psi)\mu$ is satisfiable, and $\varphi' \Rightarrow \psi\mu$ is valid. The result is indeed $s\mu[r\mu] = (s[r]_p)\mu = t$. \square

3 Loop Criteria

Our aim is to detect loops in LCTRSs. More precisely, given an LCTRS \mathcal{R} we want to find rewrite sequences $t \rightarrow_{\mathcal{R}}^+ C[t\sigma]$ on *unconstrained* terms. A natural approach to this end from standard rewriting is *unfolding* [19]: one tries to compose (instances of) rewrite rules such that the final term of the resulting rewrite sequence contains (an instance of) the initial term. For our setting, this requires to rewrite *constrained* terms. But a rewrite sequence $t [\varphi] \rightarrow_{\mathcal{R}}^+ C[t\sigma] [\psi]$ on constrained terms where the final term contains the initial term need not imply a loop: this depends on whether the constraints can remain satisfied after repeated execution of the respective rewrite steps. In this section we consider a rewrite sequence $t [\psi] \rightarrow_{\mathcal{R}}^+ C[t\sigma] [\psi]$ and look for sufficient criteria such that these steps give rise to a loop. If there exists a ψ as above then we abbreviate this by $t \rightarrow_{\psi, \mathcal{R}}^+ C[t\sigma]$ and call it a *loop candidate*.

The following criterion was presented in [18, Theorem 2].

Lemma 2. *Let \mathcal{R} be an LCTRS, and ψ a constraint. Suppose $t \rightarrow_{\psi, \mathcal{R}}^+ C[t\sigma]$ for a term t , context C , and substitution σ such that $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ for all $x \in \text{Var}(\psi)$, ψ is satisfiable, and $\psi \Rightarrow \psi\sigma$ valid. Then \mathcal{R} is nonterminating.*

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 constraint. Suppose that $t \rightarrow_{\psi, \mathcal{R}}^+ C[t\sigma]$ for some term t , context C , and substitution σ such that $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ for all $x \in \text{Var}(\psi)$, and $\psi \wedge \bigwedge_{y \in \text{Dom}(\sigma)} y \approx y\sigma$ is a constraint satisfied by some assignment α . Then \mathcal{R} is nonterminating because of the loop $t\alpha \rightarrow_{\mathcal{R}}^+ C[t\sigma\alpha]$.*

Proof. If $\psi \wedge \bigwedge_{y \in \text{Dom}(\sigma)} y \approx y\sigma$ is satisfied by an assignment α then $\psi\alpha$ is valid, and $[y\alpha]_{\mathcal{J}} = [y\sigma\alpha]_{\mathcal{J}}$ for all $y \in \text{Dom}(\sigma)$. Thus $t\sigma\alpha \xrightarrow{*}_{\text{calc}} t\alpha$ such that there is a loop $t\alpha \xrightarrow{+}_{\mathcal{R}} C[t\sigma\alpha] \xrightarrow{*}_{\text{calc}} C[t\alpha] \xrightarrow{+}_{\mathcal{R}} \dots$. \square

Example 3. Returning to Example 1, the two rewrite steps

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

constitute a loop candidate, where $\varphi = c <_s \#x0000 \wedge \text{isPowerOf2}(\text{abs}(c))$ and $\psi = \varphi \wedge c' = \text{abs}(c)$. We thus have $t[\psi] \xrightarrow{+}_{\mathcal{R}} C[t\sigma] [\psi]$ for $t = \text{mul}(\text{sub}(y, x), c)$, $C = \square$, and $\sigma = \{y \mapsto x, c \mapsto c'\}$, such that $\sigma(z)$ is a logical term for all z in ψ . The formula $\psi \wedge x = y \wedge c = c'$ is satisfiable by any assignment such that $\alpha(x) = \alpha(y)$ and $\alpha(c) = \alpha(c') = \#x8000$, which exhibits the loop in Example 1:

$$\text{mul}(\text{sub}(x, x), \#x8000) \rightarrow \text{mul}(\text{sub}(x, x), \text{abs}(\#x8000)) \rightarrow \text{mul}(\text{sub}(x, x), \#x8000)$$

The criterion of Lemma 3 is rather restrictive in that it demands the starting term to occur again as a subterm after some (calculation) steps. The next criterion adds some flexibility in this respect.

Lemma 4. *Let \mathcal{R} be an LCTRS, and ψ a constraint. Suppose that $t \xrightarrow{+}_{\psi, \mathcal{R}} C[t\sigma]$ for some term t , context C , and substitution σ such that $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\text{theory}}, \mathcal{V})$ for all $x \in \text{Var}(\psi)$. Suppose $\text{Dom}(\sigma) = \{y_1, \dots, y_n\}$, and let $\rho = \{y_1 \mapsto z_1, \dots, y_n \mapsto z_n\}$ be a renaming to fresh variables z_1, \dots, z_n .*

If $\forall y_1 \dots y_n. (\psi \implies \psi\sigma) \wedge \psi\rho$ is satisfiable by α then \mathcal{R} is nonterminating because of the loop $t\rho\alpha \xrightarrow{+}_{\mathcal{R}} C[t\sigma\rho\alpha]$.

Proof. We write \bar{y} for $y_1 \dots y_n$ and assume that $\chi = \forall \bar{y}. (\psi \implies \psi\sigma) \wedge \psi\rho$ is satisfied by some assignment α , so $\text{Ran}(\alpha) \subseteq \text{Val}$. We can assume $\text{Dom}(\alpha) \cap \{\bar{y}\} = \emptyset$ since there are no free occurrences of y_i in χ . There must be some assignment β such that $\alpha = \beta \uplus \alpha|_{\{z_i\}}$, and we abbreviate $\gamma = \rho\alpha|_{\{z_i\}}$. By assumption $\psi\rho\alpha$ holds, which coincides with $\psi\beta\gamma$ because $\text{Ran}(\beta) \subseteq \text{Val}$ and $\text{Dom}(\beta) \cap \{\bar{y}\} = \emptyset$. Moreover $\forall \bar{y}. (\psi \implies \psi\sigma)\beta\alpha|_{\{z_i\}}$ holds, and we have

$$\begin{aligned} (\forall \bar{y}. (\psi \implies \psi\sigma)\beta\alpha|_{\{z_i\}}) &= (\forall \bar{y}. (\psi\beta \implies \psi\sigma\beta))\alpha|_{\{z_i\}} && \text{as } \text{Dom}(\beta) \cap \{\bar{y}\} = \emptyset \\ &= (\forall \bar{y}. (\psi\beta \implies \psi\sigma\beta)) && \text{because } \bar{z} \text{ are fresh} \\ &= (\forall \bar{y}. (\psi\beta \implies \psi\beta\sigma)) && \text{as } \text{Dom}(\beta) \cap \{\bar{y}\} = \emptyset \end{aligned}$$

Thus $\psi\beta\gamma'$ implies $\psi\beta\sigma\gamma'$ for all substitutions γ' with $\text{Dom}(\gamma') = \{\bar{y}\}$. Since $\text{Dom}(\sigma^k\gamma) = \{\bar{y}\}$ the constraint $\psi\beta\sigma^k\gamma = \psi\sigma^k\beta\gamma = \psi\sigma^k\rho\alpha$ holds for all $k \geq 0$. Hence we have the loop

$$t\rho\alpha \xrightarrow{+}_{\mathcal{R}} C[t\sigma\rho\alpha] \xrightarrow{+}_{\mathcal{R}} C^2[t\sigma^2\rho\alpha] \xrightarrow{+}_{\mathcal{R}} \dots \quad \square$$

Example 4. Consider the following LCTRS \mathcal{R}_0 with constraints over the integers:

$$f(x, y) \rightarrow f(x + 1 - y, y) - 1 \quad [y \neq 1 \wedge x \geq 0]$$

The rule constitutes a loop candidate: We have $t \rightarrow_{\psi, \mathcal{R}}^+ C[t\sigma]$ for $t = f(x, y)$, $C = \square - 1$, and $\sigma = \{x \mapsto x + 1 - y\}$ with $\text{Dom}(\sigma) = \{x\}$. The formula

$$\forall x (y \neq 1 \wedge x \geq 0 \implies (y \neq 1 \wedge (x + 1 - y) \geq 0)) \wedge y \neq 1 \wedge z \geq 0$$

is satisfied e.g. by the assignment $\alpha(y) = \alpha(z) = 0$. Thus we can detect the loop

$$f(0, 0) \rightarrow_{\mathcal{R}} f(0 + 1 - 0, 0) \xrightarrow{+}_{\text{calc}} f(1, 0) \rightarrow_{\mathcal{R}} f(1 + 1 - 0, 0) \xrightarrow{+}_{\text{calc}} f(2, 0) \rightarrow \dots$$

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

It is clear that Lemma 4 subsumes Lemma 2—satisfiability of ψ and validity of $\psi \implies \psi\sigma$ in Lemma 2 implies satisfiability of $\forall y_1 \dots y_n. (\psi \implies \psi\sigma) \wedge \psi\rho$ in Lemma 4. The LCTRS \mathcal{R}_0 from Example 4 indicates the existence of an example for which Lemma 4 can detect a loop but Lemmas 2 or 3 do not. The following example shows the remaining relationship between Lemmas 2, 3, and 4.

Example 5. A loop of the LCTRS $\mathcal{R}_1 = \{ f(x) \rightarrow f(x) \ [x \geq 0] \}$ can be detected by Lemmas 2, 3, and 4. A loop of the LCTRS $\mathcal{R}_2 = \{ f(x) \rightarrow f(x + 1) \ [x \geq 0] \}$ can be detected by Lemmas 2 and 4 but not by Lemma 3. A loop of the LCTRS $\mathcal{R}_3 = \{ f(x, y) \rightarrow f(x + y, y) \ [x \geq 0] \}$ can be detected by Lemmas 3 and 4 but not by Lemma 2. A loop of the LCTRS $\mathcal{R}_4 = \{ f(x, y) \rightarrow f(y + 1, x - 1) \ [x \geq 0 \wedge y \geq 0] \}$ can be detected by Lemma 3 but not by Lemmas 2 or 4.

The relationship between the different criteria is summarized in Fig. 1.

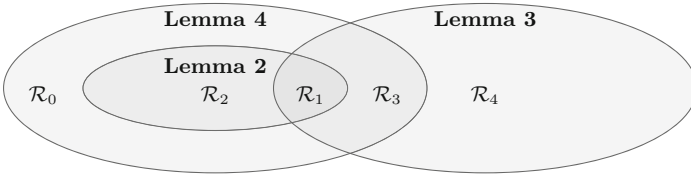


Fig. 1. Relationship between the criteria implied by Lemmas 2, 3, and 4.

4 Implementation

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

- (a) Given an LCTRS \mathcal{R} , find a loop $t \rightarrow_{\mathcal{R}}^+ C[t\sigma]$.
- (b) Given an LCTRS \mathcal{R} and a starting term u , find a loop reachable from u , i.e., a sequence $u \rightarrow_{\mathcal{R}}^* t \rightarrow_{\mathcal{R}}^+ C[t\sigma]$.

To that end our implementation searches loop candidates $t \xrightarrow{+}_{\varphi, \mathcal{R}} C[t\sigma]$ which satisfy the criteria in Lemmas 2–4. An input file in the `ctrls` format specifies the logical theory to be used, the signature, the rewrite rules, and a query to fix the problem statement for `Ctrl`, i.e., the requested analysis or transformation task. To support nontermination analysis, we provide `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 for this work. Alternatively, a user-defined theory specification can be used.

We next describe how our implementation detects loops. Following the idea of unfolding [19], we construct *sequence tuples* $(s \rightarrow t [\psi], S)$ where $s \rightarrow t [\psi]$ is a constrained rewrite rule, $S = [(\alpha_1, p_1), \dots, (\alpha_k, p_k)]$, α_i is a rule of the form $\ell_i \rightarrow r_i [\varphi_i]$ and p_i are positions for all $0 \leq i \leq k$ such that there is the rewrite sequence $s [\psi] \rightarrow_{\alpha_1, p_1} \dots \rightarrow_{\alpha_k, p_k} t [\psi]$. In either of the modes (a) and (b), we proceed in five steps as follows.

- (1) 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:

- (2) The set of initial sequence tuples T_0 is determined. In case of (a), we take the set of all single-step sequences $(\ell \rightarrow r [\varphi], [(\ell \rightarrow r [\varphi], \epsilon)])$ such that $\ell \rightarrow r [\varphi] \in \mathcal{P}$. In case of (b), this set is restricted to those tuples where a rewrite sequence $u \xrightarrow{\varphi, \mathcal{R}} v[\ell]$ was found.
- (3) Given tuples T_i , we define T_{i+1}^f for forward and T_{i+1}^b for backward unfolding:

$$T_{i+1}^f = \{(s\tau \rightarrow u [\chi], S_f) \mid (s \rightarrow t [\psi], S) \in T_i, \beta \in \mathcal{Q} \text{ and } t [\varphi] \rightsquigarrow_{\beta, q}^{\tau} u [\chi]\}$$

$$T_{i+1}^b = \{(u\tau \rightarrow t [\chi], S_b) \mid (s \rightarrow t [\psi], S) \in T_i, \beta \in \mathcal{Q} \text{ and } u [\varphi] \rightsquigarrow_{\beta, q}^{\tau} s [\chi]\}$$

Here \mathcal{Q} abbreviates $\mathcal{P} \cup \mathcal{R}$, $S_f = S ++ [(\beta, q)]$ and $S_b = [(\beta, q)] ++ S$, where $++$ denotes list concatenation.

- (4) Let $T = \bigcup_{i \leq n} T_i$ for some n . By the construction of T_i and Lemma 1, we have $s [\psi] \xrightarrow{+}_{\mathcal{R} \cup \mathcal{P}} t [\psi]$ for all $(s \rightarrow t [\psi], S) \in T$. If $t = C[s']$ for some C and s' such that s and s' are unifiable with mgu μ and $\psi\mu$ is satisfiable, then $s\mu [\psi\mu] \xrightarrow{+}_{\mathcal{R} \cup \mathcal{P}} C[s\mu] [\psi\mu]$ is a loop candidate.
- (5) 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 [19], both methods are supported. The tool as well as input files corresponding to the examples used in this paper can be found on-line¹.

¹ http://cl-informatik.uibk.ac.at/users/swinkler/lctrls_loops.

5 Applications

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

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 [16] 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.

Table 1. Instcombine loops found via forward (`fw`) and backward (`bw`) unfolding.

	# rules	add-sub	mul-div	shift	and-or	select	loops	all
		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

A dedicated tool `alive-loops` to detect loops in the Instcombine optimizations was presented in [16]. 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 $t \rightarrow_{\psi, \mathcal{R}}^+ C[t\sigma]$ 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 $\text{and}(\#x0, x) \rightarrow \text{and}(\#x0, x \gg_u \#x1) [x > \#x0]$. It gives rise to a loop candidate $t \rightarrow_{\psi, \mathcal{R}}^+ t\sigma$ where $\psi = x > \#x0$, $t = \text{and}(\#x0, x)$, and $\sigma = \{x \mapsto x \gg_u \#x1\}$. The constraint $\psi \implies \psi\sigma$ is satisfiable. But logically shifting x to the right will eventually result in a bit vector `#x0000`, hence no such loop exists. Indeed `alive-loops` finds a spurious loop in this example, but Ctrl does not.

By the correctness proofs of Lemmas 3 and 4, such false positives can be excluded for Ctrl. Moreover `alive-loops` is limited in that it restricts to loop candidates which are not size-increasing.

We remark that not all loops found by Ctrl or `alive-loops` 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.

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-ob1-8` from the Termination Problem Database 9.0² corresponds to the following LCTRS:

$$\text{f1_0_main}(x, y) \rightarrow \text{f81_0}(x', y') \quad [x > 0 \wedge y > -1 \wedge y = x'] \quad (1)$$

$$\text{f81_0}(x, y) \rightarrow \text{f81_0}(x', y') \quad [x < 0 \wedge x > -3 \wedge x + 2 = x'] \quad (2)$$

$$\text{f81_0}(x, y) \rightarrow \text{f81_0}(x', y') \quad [x > 0 \wedge x < 3 \wedge x - 2 = x'] \quad (3)$$

$$\text{f81_0}(x, y) \rightarrow \text{f81_0}(x', y') \quad [x < -2 \wedge x < -1 \wedge x < 0 \wedge -x - 2 = x'] \quad (4)$$

$$\text{f81_0}(x, y) \rightarrow \text{f81_0}(x', y') \quad [x > 2 \wedge -x + 2 = x'] \quad (5)$$

$$\text{init}(x, y) \rightarrow \text{f1_0_main}(x', y') \quad (6)$$

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

$$\text{init}(1, 1) \xrightarrow{(6)} \text{f1_0_main}(1, 1) \xrightarrow{(1)} \text{f81_0}(1, -1) \xrightarrow{(3)} \text{f81_0}(-1, 0) \xrightarrow{(2)} \text{f81_0}(1, -1)$$

(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 2017³ 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;
}
```

² <http://termination-portal.org/wiki/TPDB>.

³ http://www.termination-portal.org/wiki/Termination_Competition_2017.

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

$$\begin{aligned}
 & bsearch(k_1, lo_1, hi_1) \rightarrow u_2(k_1, lo_1, hi_1, rnd_1) \\
 & u_2(k_1, lo_1, hi_1, mid_2) \rightarrow u_3(k_1, lo_1, hi_1, (lo_1 + hi_1) /_u \#x02) \quad [lo_1 <_u hi_1] \\
 & u_3(k_1, lo_1, hi_1, mid_2) \rightarrow u_5(k_1, (mid_2 + \#x01), hi_1, mid_2) \quad [mid_2 <_u k_1] \\
 & u_3(k_1, lo_1, hi_1, mid_2) \rightarrow u_6(k_1, lo_1, (mid_2 - \#x01), mid_2) \quad [mid_2 \geqslant k_1 \wedge mid_2 > k_1] \\
 & u_6(k_1, lo_1, hi_1, mid_2) \rightarrow u_9(k_1, lo_1, hi_1, mid_2) \\
 & u_3(k_1, lo_1, hi_1, mid_2) \rightarrow return(mid_2) \quad [mid_2 \geqslant k_1 \wedge mid_2 \leqslant k_1] \\
 & u_5(k_1, lo_1, hi_1, mid_2) \rightarrow u_9(k_1, lo_1, hi_1, mid_2) \\
 & u_9(k_1, lo_1, hi_1, mid_2) \rightarrow u_{10}(k_1, lo_1, hi_1, mid_2) \\
 & u_{10}(k_1, lo_1, hi_1, mid_2) \rightarrow u_2(k_1, lo_1, hi_1, mid_2) \\
 & u_2(k_1, lo_1, hi_1, mid_2) \rightarrow return(\#xff) \quad [lo_1 \geqslant_u hi_1]
 \end{aligned}$$

Ctrl can prove existence of a loop that is reachable from a term of the form $bsearch(x, y, l, h)$ 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` [16], 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, 17]. Moreover, it would be interesting to find criteria for nonlooping nontermination of LCTRSs.

Acknowledgements. The authors thank the anonymous referees for their helpful comments.

References

1. Baader, F., Nipkow, T.: *Term Rewriting and All That*. Cambridge University Press, Cambridge (1998)
2. Borralleras, C., Brockschmidt, M., Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving termination through conditional termination. In: Legay, A., Margaria, T. (eds.) *TACAS 2017*. Heidelberg, vol. 10205, pp. 99–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-662-54577-5_6
3. Brockschmidt, M., Cook, B., Ishtiaq, S., Khlaaf, H., Piterman, N.: T2: Temporal property verification. In: Chechik, M., Raskin, J.-F. (eds.) *TACAS 2016*. LNCS, vol. 9636, pp. 387–393. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_22
4. Falke, S., Kapur, D.: A term rewriting approach to the automated termination analysis of imperative programs. In: Schmidt, R.A. (ed.) *CADE 2009*. LNCS (LNAI), vol. 5663, pp. 277–293. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_22
5. Falke, S., Kapur, D., Sinz, C.: Termination analysis of C programs using compiler intermediate languages. In: *Proceedings of the 22nd RTA, Leibniz International Proceedings in Informatics*, vol. 10, pp. 41–50 (2011). <https://doi.org/10.4230/LIPIcs.RTA.2011.41>
6. Fuhs, C., Giesl, J., Plücker, M., Schneider-Kamp, P., Falke, S.: Proving termination of integer term rewriting. In: Treinen, R. (ed.) *RTA 2009*. LNCS, vol. 5595, pp. 32–47. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02348-4_3
7. Fuhs, C., Kop, C., Nishida, N.: Verifying procedural programs via constrained rewriting induction. *ACM TOCL* **18**(2), 14:1–14:50 (2017). <https://doi.org/10.1145/3060143>
8. Ganesh, V., Berezin, S., Dill, D.: A decision procedure for fixed-width bit-vectors. Technical report, Stanford University (2005)
9. Giesl, J., et al.: Analyzing program termination and complexity automatically with AProVE. *JAR* **58**(1), 3–31 (2017). <https://doi.org/10.1007/s10817-016-9388-y>
10. Gupta, A., Henzinger, T., Majumdar, R., Rybalchenko, A., Xu, R.G.: Proving non-termination. *SIGPLAN Not.* **43**(1), 147–158 (2008). <https://doi.org/10.1145/1328897.1328459>
11. Hoder, K., Khasidashvili, Z., Korovin, K., Voronkov, A.: Preprocessing techniques for first-order clausification. In: *Proceedings of the 12th FMCAD*, pp. 44–51 (2012)
12. Kop, C., Nishida, N.: Term rewriting with logical constraints. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) *FroCoS 2013*. LNCS (LNAI), vol. 8152, pp. 343–358. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40885-4_24
13. Kop, C., Nishida, N.: Constrained term rewriting tool. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) *LPAR 2015*. LNCS, vol. 9450, pp. 549–557. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_38
14. Lopes, N., Menendez, D., Nagarakatte, S., Regehr, J.: Provably correct peephole optimizations with Alive. In: *Proceedings of the 36th PLDI*, pp. 22–32 (2015). <https://doi.org/10.1145/2737924.2737965>
15. Lopes, N., Menendez, D., Nagarakatte, S., Regehr, J.: Practical verification of peephole optimizations with Alive. *Commun. ACM* **61**(2), 84–91 (2018). <https://doi.org/10.1145/3166064>
16. Menendez, D., Nagarakatte, S.: Termination-checking for LLVM peephole optimizations. In: *Proceedings of the 38th International Conference on Software Engineering*, pp. 191–202 (2016). <https://doi.org/10.1145/2884781.2884809>

17. Nadel, A.: Bit-vector rewriting with automatic rule generation. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 663–679. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_44
18. Nishida, N., Sakai, M., Hattori, T.: On disproving termination of constrained term rewriting systems. In: Proceedings of the 11th WST (2010)
19. Payet, É.: Loop detection in term rewriting using the eliminating unfoldings. Theor. Comput. Sci. **403**(2–3), 307–327 (2008). <https://doi.org/10.1016/j.tcs.2008.05.013>