



Automated Analysis of Logically Constrained Rewrite Systems Using **crest***

Jonas Schöpf^(✉)  and Aart Middeldorp 

Department of Computer Science, University of Innsbruck, Innsbruck, Austria
{jonas.schoepf,aart.middeldorp}@uibk.ac.at

Abstract. We present **crest**, a tool for automatically proving (non-) confluence and termination of logically constrained rewrite systems. We compare **crest** to other tools for logically constrained rewriting. Extensive experiments demonstrate the promise of **crest**.

Keywords: Automation · Confluence · Termination · Term Rewriting · Logical Constraints.

1 Introduction

Term rewriting is a simple Turing-complete model of computation. Properties like confluence and termination are of key interest and numerous powerful tools have been developed for their analysis. Logically constrained term rewrite systems (LCTRSs for short) constitute a natural extension of term rewrite systems (TRSs) in which rules are equipped with logical constraints that are handled by SMT solvers, thereby avoiding the cumbersome encoding of operations on e.g. integers and bit vectors in term rewriting. LCTRSs, introduced by Kop and Nishida in 2013 [25], are useful for program analysis [8, 13, 23, 40]. They also developed Ctrl [26, 27], a tool for LCTRSs specializing in termination analysis and equivalence testing. Later, techniques for completion [39] and non-termination [33] analysis were added.

In this paper we describe **crest**, the Constrained REwriting Software Tool. The tool **crest** was first announced in [34] with support for a small number of confluence techniques. The new version described here includes numerous extensions:

- more advanced confluence techniques (introduced in [35]),
- automated non-confluence and termination analysis,
- support for fixed-sized bit vectors,
- transformation techniques based on splitting critical pairs and merging constrained rewrite rules, to further boost the confluence proving power.

* This research is funded by the Austrian Science Fund (FWF) project I5943.

Extensive experiments show the strength of *crest*. The tool is open-source and available from <http://cl-informatik.uibk.ac.at/software/crest>.

The remainder of the paper is organized as follows. In the next section we recall important definitions pertaining to LCTRSs. Section 3 summarizes the main confluence and termination techniques implemented in *crest*. Automation details are presented in Section 4. The new transformation techniques are described in Section 5. In Section 6 we present our experiments, before concluding in Section 7 with suggestions for future extensions. We conclude this introductory section with mentioning other tools for LCTRSs.

Related Tools. We already mentioned Ctrl¹ which until 2023 was the only tool capable of analyzing confluence and termination of LCTRSs. It supports termination analysis [24], completion techniques [39], rewriting induction for equivalence testing of LCTRSs [13], and basic confluence analysis [27]. Unfortunately, it is neither actively maintained nor very well documented, which is one reason why the development of *crest* was started. Moreover, a branch² of the automated resource analysis tool TcT [4] performs complexity analysis on LCTRSs based on [40]. RMT by Ciobăcă et al. [7, 8] is a newer tool for program analysis based on a variation of LCTRSs.

In the 2024 edition of the Confluence Competition³ the tool CRaris,⁴ developed by Nishida and Kojima, made its appearance. The tool implements weak orthogonality [25] and the Knuth–Bendix criterion for terminating LCTRSs [34]. For termination, it implements the dependency pair framework [24] and the singleton self-looping removal processor [29] for LCTRSs with bit vectors.

Also in 2024 Guo et al. [14, 15] announced Cora, a new open-source tool for termination analysis of logically constrained *simply-typed* term rewrite systems, which serve as a high-order generalization of LCTRSs. It employs static dependency pairs [28] with several base methods, including a variant of the higher-order recursive path order [18].

2 Logically Constrained Term Rewriting

Familiarity with the basic notions of term rewriting [5] is assumed. We assume a many-sorted signature $\mathcal{F} = \mathcal{F}_{\text{te}} \cup \mathcal{F}_{\text{th}}$ consisting of term and theory symbols together with a countably infinite set of variables $\mathcal{V}$. For every sort ι in $\mathcal{F}_{\text{th}}$ we have a non-empty set $\mathcal{Val}_\iota \subseteq \mathcal{F}_{\text{th}}$ of value symbols, such that all $c \in \mathcal{Val}_\iota$ are constants of sort ι . We demand $\mathcal{F}_{\text{te}} \cap \mathcal{F}_{\text{th}} \subseteq \mathcal{Val}$ where $\mathcal{Val} = \bigcup_\iota \mathcal{Val}_\iota$. The set of terms constructed from function symbols in $\mathcal{F}$ and variables in $\mathcal{V}$ is by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. A term in $\mathcal{T}(\mathcal{F}_{\text{th}}, \mathcal{V})$ is called a *logical* term. Ground logical terms are mapped to values by an interpretation $\mathcal{J}$: $\llbracket f(t_1, \dots, t_n) \rrbracket = f_{\mathcal{J}}(\llbracket t_1 \rrbracket, \dots, \llbracket t_n \rrbracket)$. Logical terms of sort *bool* are called *constraints*. A constraint φ is *valid* if $\llbracket \varphi \rrbracket = \top$ for all

¹ <http://cl-informatik.uibk.ac.at/software/ctrl/>

² <https://github.com/bytekid/tct-lctrs>

³ <https://ari-cops.uibk.ac.at/CoCo/2024/competition/LCTRS/>

⁴ <https://www.trs.css.i.nagoya-u.ac.jp/craris/>

substitutions γ such that $\gamma(x) \in \mathcal{Val}$ for all $x \in \mathcal{Var}(\varphi)$. Positions are sequences of positive integers to indicate subterms. The root of a term is denoted by the empty string ϵ . For a term s , its subterm at position p is given by $s|_p$. The set of positions in $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is denoted by $\mathcal{Pos}(s)$ whereas $\mathcal{Pos}_{\mathcal{F}}(s)$ is restricted to positions with function symbols in s . We write $\mathcal{Var}(s)$ for the set of variables in s . A *constrained rewrite rule* is a triple $\rho: \ell \rightarrow r [\varphi]$ where $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ are terms of the same sort such that $\text{root}(\ell) \in \mathcal{F}_{\text{te}} \setminus \mathcal{F}_{\text{th}}$ and φ is a constraint. We denote the set $\mathcal{Var}(\varphi) \cup (\mathcal{Var}(r) \setminus \mathcal{Var}(\ell))$ of *logical* variables in ρ by $\mathcal{LVar}(\rho)$. We write $\mathcal{EVar}(\rho)$ for the set $\mathcal{Var}(r) \setminus (\mathcal{Var}(\ell) \cup \mathcal{Var}(\varphi))$ of *extra* variables. A set of constrained rewrite rules is called an LCTRS. A substitution σ *respects* a rule $\rho: \ell \rightarrow r [\varphi]$, denoted by $\sigma \models \rho$, if $\text{Dom}(\sigma) \subseteq \mathcal{Var}(\rho)$, $\sigma(x) \in \mathcal{Val}$ for all $x \in \mathcal{LVar}(\rho)$, and $\varphi\sigma$ is valid. Moreover, a constraint φ is respected by σ , denoted by $\sigma \models \varphi$, if $\sigma(x) \in \mathcal{Val}$ for all $x \in \mathcal{Var}(\varphi)$ and $\varphi\sigma$ is valid. We call $f(x_1, \dots, x_n) \rightarrow y [y = f(x_1, \dots, x_n)]$ with a fresh variable y and $f \in \mathcal{F}_{\text{th}} \setminus \mathcal{Val}$ a *calculation rule*. The set of all calculation rules induced by the signature $\mathcal{F}_{\text{th}}$ of an LCTRS $\mathcal{R}$ is denoted by $\mathcal{R}_{\text{ca}}$ and we abbreviate $\mathcal{R} \cup \mathcal{R}_{\text{ca}}$ to $\mathcal{R}_{\text{rc}}$. A rewrite step $s \rightarrow_{\mathcal{R}} t$ satisfies $s|_p = \ell\sigma$ and $t = s[r\sigma]_p$ for some position p , constrained rewrite rule $\rho: \ell \rightarrow r [\varphi]$ in $\mathcal{R}_{\text{rc}}$, and substitution σ such that $\sigma \models \rho$.

A *constrained term* is a pair $s [\varphi]$ consisting 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 $\gamma \models \varphi$ with $\text{Dom}(\gamma) = \mathcal{Var}(\varphi)$ there is some substitution $\delta \models \psi$ with $\text{Dom}(\delta) = \mathcal{Var}(\psi)$ such that $s\gamma = t\delta$, and vice versa. Let $s [\varphi]$ be a constrained term. If $s|_p = \ell\sigma$ for some constrained rewrite rule $\rho: \ell \rightarrow r [\psi] \in \mathcal{R}_{\text{rc}}$, position p , and substitution σ such that $\sigma(x) \in \mathcal{Val} \cup \mathcal{Var}(\varphi)$ for all $x \in \mathcal{LVar}(\rho)$, φ is satisfiable and $\varphi \Rightarrow \psi\sigma$ is valid then $s [\varphi] \rightarrow_{\mathcal{R}} s[r\sigma]_p [\varphi]$. The rewrite relation $\rightarrow_{\mathcal{R}}$ on constrained terms is defined as $\sim \cdot \rightarrow_{\mathcal{R}} \cdot \sim$ and $s [\varphi] \rightarrow_{\mathcal{R}}^p t [\psi]$ indicates that the rewrite step in $\rightarrow_{\mathcal{R}}$ takes place at position p in s . Similarly, we write $s [\varphi] \rightarrow_{\mathcal{R}}^{\geq p} t [\psi]$ if the position in the rewrite step is below position p . We illustrate some of these concepts by means of a simple example which models the computation of the maximum of two integers.

Example 1. Consider the LCTRS $\mathcal{R}$ over the theory Ints with the rules

$$\alpha: \max(x, y) \rightarrow x [x \geq y] \qquad \beta: \max(x, y) \rightarrow y [y \geq x]$$

Here x and y are logical variables in both rules. There are no extra variables. The symbol $\max$ is the only term symbol. The theory symbols depend on the definition of Ints . As the goal is automation this usually consists of non-linear integer arithmetic as specified in the respective SMT-LIB theory.⁵

By applying the calculation rule $x_1 + x_2 \rightarrow y [y = x_1 + x_2]$ with substitution $\{x_1 \mapsto 3, x_2 \mapsto 2, y \mapsto 5\}$ followed by rule α we obtain

$$\max(3 + 2, 3) \rightarrow \max(5, 3) \rightarrow 5$$

⁵ <https://smtlib.cs.uiowa.edu/Theories/Ints.smt2>

An example of constrained rewriting is given by

$$\begin{aligned} \max(3, 3+x) [x \geq 0] &\simeq \max(3, z) [x \geq 0 \wedge z = 3+x] \\ &\simeq z [x \geq 0 \wedge z = 3+x] \end{aligned}$$

One-step rewriting, i.e., rewriting a term using a single rule, was introduced above. The sufficient criteria for confluence, highlighted in the next section, heavily rely on the notation of parallel ($\Rightarrow$) and multi-step ($\Rightarrow^*$) rewriting following [35, Definition 3] and [34, Definition 8]. The former is capable of applying several rules at parallel positions in a step while the latter additionally allows recursive steps within the used matching substitutions of rules. A rewrite sequence consists of consecutive rewrite steps, independent of which kind. The reflexive and transitive closure of $\rightarrow$ is denoted by $\rightarrow^*$. Moreover, for arbitrary terms s and t we write $s \leftrightarrow t$ if $s (\leftarrow \cup \rightarrow) t$ and $s \downarrow t$ if there exists a term u such that $s \rightarrow^* u \leftarrow^* t$.

3 Confluence and Termination

Termination and confluence are well-known properties in static program analysis. Both properties are in general undecidable. With respect to (logically constrained) term rewriting, a program is terminating whenever it does not admit an infinite rewrite sequence. Confluence states that $s \downarrow t$ whenever $t \leftarrow^* s \rightarrow^* u$, for all terms s, t and u . Naively checking the properties is obviously not feasible. In (logically constrained) term rewriting there exist sufficient criteria that guarantee that these properties are satisfied for a given program. In the following we highlight key components of confluence and termination analysis for logically constrained rewrite systems.

The confluence methods implemented in *crest* are based on (parallel) critical pairs. These are defined as follows. Given a constrained rewrite rule ρ , we write $\mathcal{EC}_\rho$ for $\bigwedge\{x = x \mid x \in \mathcal{EVar}(\rho)\}$. An *overlap* of an LCTRS $\mathcal{R}$ is a triple $\langle \rho_1, p, \rho_2 \rangle$ with rules $\rho_1: \ell_1 \rightarrow r_1 [\varphi_1]$ and $\rho_2: \ell_2 \rightarrow r_2 [\varphi_2]$, satisfying the following conditions: (1) ρ_1 and ρ_2 are variable-disjoint variants of rewrite rules in $\mathcal{R}_{rc}$, (2) $p \in \mathcal{Pos}_{\mathcal{F}}(\ell_2)$, (3) ℓ_1 and $\ell_2|_p$ unify with mgu σ such that $\sigma(x) \in \mathcal{Val} \cup \mathcal{V}$ for all $x \in \mathcal{LVar}(\rho_1) \cup \mathcal{LVar}(\rho_2)$, (4) $\varphi_1\sigma \wedge \varphi_2\sigma$ is satisfiable, and (5) if $p = \epsilon$ then ρ_1 and ρ_2 are not variants, or $\mathcal{Var}(r_1) \not\subseteq \mathcal{Var}(\ell_1)$. In this case we call $\ell_2\sigma[r_1\sigma]_p \approx r_2\sigma [\varphi_1\sigma \wedge \varphi_2\sigma \wedge \psi\sigma]$ a *constrained critical pair* (CCP) obtained from the overlap $\langle \rho_1, p, \rho_2 \rangle$. Here $\psi = \mathcal{EC}_{\rho_1} \wedge \mathcal{EC}_{\rho_2}$. The peak

$$\ell_2\sigma[r_1\sigma]_p [\Phi] \leftarrow \ell_2\sigma [\Phi] \rightarrow^\epsilon r_2\sigma [\Phi]$$

with $\Phi = (\varphi_1 \wedge \varphi_2 \wedge \psi)\sigma$, from which the constrained critical pair originates, is called a *constrained critical peak*. The set of all constrained critical pairs of $\mathcal{R}$ is denoted by $\text{CCP}(\mathcal{R})$. A constrained equation $s \approx t [\varphi]$ is *trivial* if $s\sigma = t\sigma$ for every substitution σ with $\sigma \models \varphi$. The trivial equations from $\mathcal{EC}_\rho$ are used in order to prevent losing the information which (extra) variables are logical variables in the underlying rules of a CCP.

Example 2. Let us extend the LCTRS $\mathcal{R}$ from Example 1 with an additional rule modeling the commutativity of $\max$:

$$\max(x, y) \rightarrow x [x \geq y] \quad \max(x, y) \rightarrow y [y \geq x] \quad \max(x, y) \rightarrow \max(y, x)$$

There are six constrained critical pairs, including the following two:

$$x \approx y [x \geq y \wedge y \geq x] \quad x \approx \max(y, x) [x \geq y]$$

The left one is trivial, the one on the right becomes trivial after one rewrite step: $x \approx \max(y, x) [x \geq y] \rightarrow x \approx x [x \geq y]$. The remaining four pairs can be rewritten similarly.

Restricting the way in which constrained critical peaks are rewritten into trivial ones, yields different sufficient conditions for confluence of (left-)linear LCTRSs. We state the conditions below but refer to [25, 34, 35] for precise definitions:

- (C1) (weak) orthogonality ([25, Theorem 4]),
- (C2) joinable critical pairs for terminating LCTRSs ([34, Corollary 4]).
- (C3) strong closedness for linear LCTRSs ([34, Theorem 2]),
- (C4) (almost) parallel closedness for left-linear LCTRSs ([34, Theorem 4]),
- (C5) (almost) development closedness for left-linear LCTRSs ([35, Corollary 1]).

The final confluence criterion implemented in *crest* is based on parallel critical pairs. Let $\mathcal{R}$ be an LCTRS, $\rho: \ell \rightarrow r [\varphi]$ a rule in $\mathcal{R}_{rc}$, and $P \subseteq \text{Pos}_{\mathcal{F}}(\ell)$ a non-empty set of parallel positions. For every $p \in P$ let $\rho_p: \ell_p \rightarrow r_p [\varphi_p]$ be a variant of a rule in $\mathcal{R}_{rc}$. Let $\psi = \mathcal{EC}_{\rho} \wedge \bigwedge_{p \in P} \mathcal{EC}_{\rho_p}$ and $\Phi = \varphi\sigma \wedge \psi\sigma \wedge \bigwedge_{p \in P} \varphi_p\sigma$. The peak $\ell\sigma[r_p\sigma]_{p \in P} [\Phi] \leftarrow \ell\sigma [\Phi] \rightarrow_{\mathcal{R}} r\sigma [\Phi]$ forms a *constrained parallel critical pair* $\ell\sigma[r_p\sigma]_{p \in P} \approx r\sigma [\Phi]$ if the following conditions are satisfied:

1. $\text{Var}(\rho_1) \cap \text{Var}(\rho_2) = \emptyset$ for different rules ρ_1 and ρ_2 in $\{\rho\} \cup \{\rho_p \mid p \in P\}$,
2. σ is an mgu of $\{\ell_p = \ell|_p \mid p \in P\}$ such that $\sigma(x) \in \text{Val} \cup \mathcal{V}$ for all $x \in \mathcal{LVar}(\rho) \cup \bigcup_{p \in P} \mathcal{LVar}(\rho_p)$,
3. $\varphi\sigma \wedge \bigwedge_{p \in P} \varphi_p\sigma$ is satisfiable, and
4. if $P = \{\epsilon\}$ then ρ_{ϵ} is not a variant of ρ or $\text{Var}(r) \not\subseteq \text{Var}(\ell)$.

A constrained peak forming a constrained parallel critical pair is called a *constrained parallel critical peak*. The set of all constrained parallel critical pairs of $\mathcal{R}$ is denoted by $\text{CPCP}(\mathcal{R})$. The following sufficient condition for confluence is reported in ([35, Corollary 2]):

- (C6) parallel closedness of parallel critical pairs for left-linear LCTRSs.

Conditions (C5) and (C6) do not subsume each other. Both generalize conditions (C1) – (C4). All these confluence criteria try to find a specific closing rewrite sequence starting from a constrained (parallel) critical pair—which is seen as a constrained equation—to a trivial constrained equation. For example, parallel closedness in (C4) involves showing that each constrained critical

pair $s \approx t [\varphi]$ can be rewritten into a trivial constrained equation using a single parallel step $s \approx t [\varphi] \not\rightarrow_{\geq 1} s' \approx t [\varphi]$. Note that only the left part (s) is rewritten here and $s' \approx t [\varphi]$ is a trivial constrained equation. For (finite) terminating TRSs, confluence is decided by rewriting critical pairs to normal form [20]. For terminating LCTRSs confluence—even for a decidable theory—is undecidable [35], but rewriting constrained critical pairs to normal forms is still of value. This is used in (C2) above. We need however to adapt the notion of normal form for constrained terms.

Example 3. The LCTRS $\mathcal{R}$ over the theory **Ints** with rewrite rules

$$\begin{array}{lll} f(x) \rightarrow g(x) [x \geq 1] & g(1) \rightarrow a & h(x) \rightarrow a [x \leq 1] \\ f(x) \rightarrow h(x) [x \leq 2] & g(x) \rightarrow b [x \geq 2] & h(x) \rightarrow b [x > 1] \\ & g(x) \rightarrow c [x < 1] & \end{array}$$

admits one (modulo symmetry) constrained critical pair:

$$g(x) \approx h(x) [x \geq 1 \wedge x \leq 2]$$

None of the rules above are applicable, so this non-trivial constrained critical pair is in normal form with respect to $\rightarrow_{\mathcal{R}}$, but it would be wrong to conclude that $\mathcal{R}$ is not confluent; all substitutions σ that satisfy the constraint $x \geq 1 \wedge x \leq 2$ allow us to rewrite $(g(x) \approx h(x))\sigma$ to the trivial equations $a \approx a$ or $b \approx b$.

Definition 1. *Given an LCTRS $\mathcal{R}$, a constrained term $s [\varphi]$ is in normal form if and only if for all substitutions σ with $\sigma \models \varphi$ we have $s\sigma \rightarrow_{\mathcal{R}} t$ for no term t .*

Note that the constrained critical pair in Example 3 is not in normal form according to this definition. We present a simple sufficient condition for non-confluence. The easy proof can be found in the appendix of [37].

Lemma 1. *An LCTRS is non-confluent if there exists a constrained critical pair that rewrites to a non-trivial constrained equation in normal form.*

We will resume the analysis of Example 3 in Section 5. Termination plays an important role in the analysis of LCTRSs. **crest** implements the following methods reported in the papers by Kop and Nishida [24, 25]:

- (T1) dependency graph ([24, Theorems 4 & 5]),
- (T2) recursive path order ([25, Theorem 5]),
- (T3) value criterion ([24, Theorem 10]),
- (T4) reduction pairs ([24, Theorem 12]).

Method (T1) computes the strongly connected components in the dependency graph, and transforms the input LCTRS into so-called DP problems, which can be analyzed independently. It lies at the heart of the dependency pair framework [19] implemented in most termination tools for TRSs. Methods (T2) and (T4) are LCTRS variants of well-known methods for TRSs [3, 9]. Two further methods implemented in **Ctrl** are ported to **crest**:

- (T5) subterm criterion
- (T6) special value criterion

While (T5) is a well-known termination method for DP problems originating from TRSs [17], (T3) and (T6) are specific to LCTRSs. Method (T5) operates on the syntactic structure of dependency pairs and ignores the constraints. In method (T3) dependency pair symbols are also projected to a direct argument but then a strict decrease with respect to the constraint is required. For example, the rule $f(x) \rightarrow f(x-1) [x > 0]$ cannot be handled by (T5), but as $x > 0$ implies the strict decrease $x \succ x-1$ for a suitable well-founded relation $\succ$, (T3) applies. Method (T6) is an extension of (T3) in which linear combinations of arguments are considered. Methods (T3) and (T6) are adapted to the higher-order LCTRS setting in [14, Sections 4.2 & 4.3].

4 Automation

Our tool **crest** is written in Haskell and the current version consists of roughly 12000 lines of code. Core modules like SMT solving use a fork of the **simple-smt** package⁶ and the rewriting modules are inspired by the **term-rewriting** package.⁷ In the following we provide some details of the key components.

Input Format. **crest** operates on LCTRSs in the new ARI format⁸ [1] adopted by the Confluence Competition (CoCo) and also partly by the Termination Competition.⁹ Problems in the ARI database are given a unique number which we will use throughout this paper to address specific LCTRSs. An example problem is given in Fig. 1. The ARI database format requires sort annotations for variables appearing as an argument of a polymorphic predicate. If this sort can be inferred at a different position then this can be ignored for **crest**. For example, consider the rule $f(x = y, x) \rightarrow z [z = x + 1]$ with $f: \text{Bool} \rightarrow \text{Int} \rightarrow \text{Int}, +: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$ and $=: A \rightarrow A \rightarrow \text{Bool}$ with a polymorphic sort A . In the ARI database all variables need concrete sort annotation. For **crest** no sort annotation is necessary as all the sorts of variables can be inferred from the sort of f .

Theory symbols are those that are defined in a specific SMT-LIB theory, however, for fixed-sized bit vectors **crest** additionally supports function symbols defined in the SMT-LIB logic **QF_BV**.¹⁰ In addition to LCTRSs also plain TRSs and many-sorted TRSs are supported.

Pre-Processing. After parsing its input and assigning already known sorts to function symbols and variables we apply a basic type inference algorithm. Some function symbols in the core theory, which provides basic boolean functions, like

⁶ <https://hackage.haskell.org/package/simple-smt>

⁷ <https://hackage.haskell.org/package/term-rewriting>

⁸ <https://project-coco.uibk.ac.at/ARI/lctrs.php>

⁹ https://termination-portal.org/wiki/Termination_Portal

¹⁰ https://smt-lib.org/logics-all.shtml#QF_BV

```

(format LCTRS :smtlib 2.6)
(theory Ints)
(fun f (-> Int Int Int))
(fun g (-> Int Int Int))
(fun c (-> Int Int Int))
(fun h (-> Int Int))
(rule (f x y) (h (g y (* 2 2))) :guard (and (<= x y) (= y 2)))
(rule (f x y) (c 4 x) :guard (<= y x))
(rule (g x y) (g y x))
(rule (c x y) (g 4 2) :guard (not (= x y)))
(rule (h x) x)

```

Fig. 1. ARI file [1528](#) (without sort annotations and meta information).

“=” have a polymorphic sort. Therefore we need to infer unknown sorts in order to obtain a fully sorted LCTRS. This is required as sort information must be present for the declaration of variables in the SMT solver. During the parsing phase *crest* parses the respective theory from an internal representation of the SMT-LIB specification. Currently the theory of integers, reals, fixed-sized bit vectors and a combination of integers and reals are supported. Subsequently *crest* preprocesses the LCTRS by moving values in the left-hand sides of the rewrite rules into the constraints (by applying the transformation described in [34, Definition 13]). Afterwards it merges as many rules as possible following Definition 3 in Section 5.

Rewriting. One of the key components is the rewriting module which provides functionality to perform rewriting on constrained terms. This module computes rewrite sequences of arbitrary length, using single steps, parallel rewrite steps [35, Definition 7] and multisteps [35, Definition 5]. Calculation steps are modeled in an obvious way; whenever we have a term $s[f(s_1, \dots, s_n)] [\varphi]$ with $s_1, \dots, s_n \in \text{Val} \cup \text{Var}(\varphi)$ and $f \in \mathcal{F}_{\text{th}}$, then we produce $s[x] [\varphi \wedge x = f(s_1, \dots, s_n)]$ for a fresh variable x . In some cases single rule steps need more care because of the lack of equivalence steps in rewrite sequences. For rules with variables that do not occur in the left-hand side, the matching substitution of the left-hand side does not provide an instantiation. However, those variables are logical and need to be instantiated with values. This is achieved by adding the constraint of the rule and its extra variables to the resulting constrained term after it has been confirmed that for those variables an instantiation exists. We illustrate this in the following example.

Example 4. Consider the constrained rule $\rho: f(x) \rightarrow y [x \geq 0 \wedge x > y]$, the constrained term $f(z) [z = 2]$ and the matching substitution $\{x \mapsto z\}$ between the left-hand side of ρ and $f(z)$. The variable y is not part of the matching substitution and thus *crest* rewrites $f(z) [z = 2]$ to $y [z = 2 \wedge z \geq 0 \wedge z > y]$. Using the constrained rule $\rho': f(x) \rightarrow y [x \geq 0]$ from the same constrained term would give $y [z = 2 \wedge z \geq 0 \wedge y = y]$.

SMT. SMT solving is a key component in the analysis of LCTRSs and SMT solvers are heavily used during the analysis. In order for SMT solving to not form a bottleneck some care is needed. Again, each different analysis method is equipped with its own SMT solver instance started at the beginning of the analysis. Afterwards such an instance runs until the method has finished. In between, it waits for SMT queries, hence we avoid several restarts of this instance. Constraints are modeled as regular terms of sort boolean and can be checked for satisfiability and validity. Each of those checks runs in its own context (using push and pop commands) in order to avoid any interference with previous queries. Currently *crest* utilizes Z3 [30] as the default SMT solver, as it turned out to be the most reliable during development. Nevertheless, *crest* provides the (experimental) possibility to use Yices [10] and CVC5 [6].

Confluence. The computation of constrained critical pairs follows the definition and constrained parallel critical pairs are computed in a bottom up fashion by collecting all possible combinations of parallel steps. Then the various methods to conclude confluence are applied on those pairs. If a method fails on a constrained critical pair then, using Definition 2, the constrained critical pair is split. The logical constraint used in splitting is taken from a matching rule. The various methods run concurrently in order to prevent starvation of methods because of pending SMT solver queries. The first method which succeeds returns the result and all others, including their SMT solver instances, are terminated. We adopt heuristics to bound the number of rewrite steps in the closing sequences. The method that posed the biggest challenge to automation is the 2-parallel closedness [35, Definition 11] needed for (C6) as we cannot simply use an arbitrary parallel step starting from the right-hand side but need to synthesize a parallel step over a set of parallel positions that adheres to the variable condition present in the definition.

Termination. The choices in the parameters of the subterm criterion (T5) and the recursive path order (T2) are modeled in the SMT encoding. Similarly, for the value criterion (T3) first all possible projections are computed. Then an SMT encoding based on the given rules and theory is constructed and a model of the encoding (if it exists) delivers suitable projections that establish termination. An explicit boolean flag in the SMT encoding determines if a strict or weak decrease is achieved. The special variant with projections to suitable linear combinations (T6) encodes this by attaching unknown constants to the projected arguments and summing them up. Those unknowns are then determined by the SMT solver. The (special) value criterion is currently restricted to the theory of integers as suitable well-founded orderings are required. For the integer theory we use $n \succ m$ if $n > m \wedge n \geq 0$ holds.

Method (T4) receives a DP problem as input and tries to transform it into a smaller one by orienting strictly as many dependency pairs as possible. It is parameterized by a list of termination methods which are applied on the DP problem. The first one which succeeds determines the remaining problem to be

solved. Before trying to solve the latter, (T1) is used to decompose it into smaller problems.

Features. Via the command-line arguments several features of **crest** can be accessed. This includes control over the number of threads in the concurrent setup, the overall timeout of the analysis, or if proof output and debug output should be printed. Furthermore, (parallel) critical pairs or the dependency graph approximation of a given LCTRS problem can be computed. The interface also offers a way to transform an LCTRS into a fully sorted LCTRS in the ARI format. In order to alter the default strategy for the analysis, **crest** offers a very basic strategy language to specify which methods should be used. Detailed information is provided in the usage information of the supplemented artifact.

5 Improving the Analysis via Transformations

In this section we present new transformations which are especially useful for confluence analysis. These transformations operate on either rules or constrained critical pairs and split or unify those based on their constraints.

Splitting Constrained Critical Pairs

If a constrained critical pair has more than one instance, which is almost always the case, and they cannot all be rewritten by a single rule, then we are not able to perform any rewrite step. To overcome this problem we propose a simple method to split constrained critical pairs.

Definition 2. *Given an LCTRS $\mathcal{R}$, a constrained critical pair $\rho: s \approx t [\varphi] \in \text{CCP}(\mathcal{R})$ and a constraint $\psi \in \mathcal{T}(\mathcal{F}_{\text{th}}, \text{Var}(\varphi))$, the set $\text{CCP}(\mathcal{R})_{\rho}^{\psi}$ is defined as $(\text{CCP}(\mathcal{R}) \setminus \{\rho\}) \cup \{s \approx t [\varphi \wedge \psi], s \approx t [\varphi \wedge \neg\psi]\}$.*

The following key lemma states that after splitting critical pairs, all confluence methods are still available. The proof is given in the appendix of [37].

Lemma 2. *If $t \leftarrow_{\mathcal{R}} s \rightarrow_{\mathcal{R}} u$ then $t \downarrow_{\mathcal{R}} u$ or $t \leftrightarrow_{\text{CCP}_{\rho}^{\psi}(\mathcal{R})} u$.*

We illustrate the lemma on the LCTRS in Example 3.

Example 5. Consider the CCP $\mathbf{g}(x) \approx \mathbf{h}(x) [\varphi]$ with $\varphi: x \geq 1 \wedge x \leq 2$ from Example 3. It is neither in normal form nor trivial. Since the subterm $\mathbf{g}(x)$ matches the left-hand side of the rule $\mathbf{g}(x) \rightarrow \mathbf{a} [x = 1]$ (which is how **crest** renders the rule $\mathbf{g}(1) \rightarrow \mathbf{a}$), and the combined constraint $\varphi \wedge x = 1$ is satisfiable, the CCP is split into

$$\mathbf{g}(x) \approx \mathbf{h}(x) [\varphi \wedge x = 1] \quad \text{and} \quad \mathbf{g}(x) \approx \mathbf{h}(x) [\varphi \wedge x \neq 1]$$

The left one rewrites to the trivial constrained equation $\mathbf{a} \approx \mathbf{a} [\varphi \wedge x = 1]$ using the rules $\mathbf{g}(x) \rightarrow \mathbf{a} [x = 1]$ and $\mathbf{h}(x) \rightarrow \mathbf{a} [x \leq 1]$. The right one is rewritten

to $\mathbf{b} \approx \mathbf{b} [\varphi \wedge x \neq 1]$ using the rules $\mathbf{g}(x) \rightarrow \mathbf{b} [x \leq 2]$ and $\mathbf{h}(x) \rightarrow \mathbf{b} [x > 1]$. Hence the LCTRS $\mathcal{R}$ is locally confluent by Lemma 2. Using RPO with the precedence $\mathbf{f} > \mathbf{g} > \mathbf{h} > \mathbf{a} > \mathbf{b} > \mathbf{c}$, termination of $\mathcal{R}$ is easily shown and hence $\mathcal{R}$ is confluent.

The following example shows that constrained critical pairs may be split infinitely often before local confluence can be verified.

Example 6. Consider the LCTRS $\mathcal{R}$ over the theory **Ints** consisting of the rules

$$\begin{array}{ll} \mathbf{a} \rightarrow \mathbf{f}(n) [n \geq 0] & \mathbf{a} \rightarrow \mathbf{g}(n) [n \geq 0] \\ \mathbf{f}(n) \rightarrow \mathbf{b} [n = 0] & \mathbf{g}(n) \rightarrow \mathbf{b} [n = 0] \\ \mathbf{f}(n) \rightarrow \mathbf{f}(m) [n > 0 \wedge 2 * m = n] & \mathbf{g}(n) \rightarrow \mathbf{g}(m) [n > 0 \wedge 2 * m = n] \\ \mathbf{f}(n) \rightarrow \mathbf{f}(m) [n > 0 \wedge 2 * m + 1 = n] & \mathbf{g}(n) \rightarrow \mathbf{g}(m) [n > 0 \wedge 2 * m + 1 = n] \end{array}$$

This LCTRS has a constrained critical pair $\mathbf{f}(n) \approx \mathbf{g}(m) [n \geq 0 \wedge m \geq 0 \wedge n = n \wedge m = m]$ originating from $\mathbf{a}$. To show confluence of $\mathcal{R}$ we would need to split the pair in order to make rules applicable for joining a specific instance. However, there are infinitely many instances with pairwise different joining sequences.

The next example shows that splitting also helps to prove non-confluence.

Example 7. Consider the LCTRS $\mathcal{R}$ in Example 3. By changing the constraint of the rule $\mathbf{f}(x) \rightarrow \mathbf{g}(x) [x \geq 1]$ to $[x \geq 0]$ we obtain a non-confluent LCTRS. This is shown by splitting the constrained critical pair $\mathbf{g}(x) \approx \mathbf{h}(x) [x \geq 0 \wedge x \leq 2]$, and subsequently showing that $\mathbf{g}(x) \approx \mathbf{h}(x) [x < 1 \wedge x \geq 0 \wedge x \leq 2]$ rewrites to the non-trivial normal form $\mathbf{c} \approx \mathbf{a} [x < 1 \wedge x \geq 0 \wedge x \leq 2]$.

Merging Constrained Rewrite Rules

Next we discuss the merging of constrained rewrite rules. The idea here is that rewrite steps may become possible after merging similar rules.

Definition 3. Let $\rho_i: \ell_i \rightarrow r_i [\varphi_i]$ for $i = 1, 2$ be variable-disjoint rewrite rules in an LCTRS $\mathcal{R}$. Suppose there exists a renaming σ such that $\ell_1 = \ell_2\sigma$, $r_1 = r_2\sigma$ and $\text{Var}(\varphi_1) = \text{Var}(\varphi_2\sigma)$. The LCTRS $\mathcal{R}_{\rho_1}^{\rho_2}$ is defined as

$$(\mathcal{R} \setminus \{\rho_1, \rho_2\}) \cup \{\ell_1 \rightarrow r_1 [\varphi_1 \vee \varphi_2\sigma]\}$$

The easy proof of the following lemma is omitted.

Lemma 3. The relations $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{R}_{\rho_1}^{\rho_2}}$ coincide.

Example 8. The LCTRS $\mathcal{R}$ over the theory **Ints** consisting of the rewrite rules

$$\begin{array}{lll} \mathbf{f}(x) \rightarrow 2 [1 \leq x \wedge x \leq 3] & \mathbf{g}(x) \rightarrow \mathbf{h}(x) & \mathbf{h}(x) \rightarrow y [x = 2 \wedge y = x] \\ \mathbf{f}(x) \rightarrow \mathbf{g}(x) [2 \leq x \wedge x \leq 4] & & \mathbf{h}(x) \rightarrow y [x = 3 \wedge y = 2] \end{array}$$

admits the constrained critical pair $2 \approx g(x) [1 \leq x \wedge x \leq 3 \wedge 2 \leq x \wedge x \leq 4]$. After rewriting the subterm $g(x)$ to $h(x)$, no further step is possible because the rewrite rules for h are not applicable. However, if we merge the two rules for h into

$$h(x) \rightarrow y [(x = 2 \wedge y = x) \vee (x = 3 \wedge y = 2)]$$

we can proceed as $1 \leq x \wedge x \leq 3 \wedge 2 \leq x \wedge x \leq 4$ implies $((x = 2 \wedge y = x) \vee (x = 3 \wedge y = 2))\sigma$ for $\sigma(y) = 2$. This is exactly how **crest** operates.

6 Experimental Evaluation

In this section we show the progress of **crest** since the start of its development in early 2023. Initial experiments of an early prototype of **crest** were reported in [34]. In the following tables the prototype of [34] is denoted by **prototype**. Since then more criteria for (non-)confluence and termination were added, and parts of the tool infrastructure were completely revised. Detailed results are available from the [website](#) of **crest** and the artifact of the experiments at the Zenodo repository [36].

All experiments were performed using the **benchexec**¹¹ benchmarking framework which is also used in the StarExec cluster. The benchmark hardware consists of several Intel Xeon E5-2650 v4 CPUs having a base clock speed of 2.20GHz, amounting in total to 64 cores and 128 GB of RAM. As benchmarks we use the problems in the new ARI database¹² in addition to the examples from this paper.

Tool Setup. Each tool receives an ARI benchmark as input and should return either "YES" (property was proved), "NO" (property was disproved) or "MAYBE" (don't know) as the first line of its output. In the tables we represent those with ✓, ✗ and ?, respectively. The fourth category depicted by † denotes that the translation from the ARI format to the input format of the respective tool failed. In order to have a realistic setup, a tool has 4 cores including 8 GB of RAM available for each run. Each tool has 60 seconds to solve a problem before it is killed. Since we have no information about how many threads the other tools use, in the experiments we use CPU time over wall-clock time in order to have a fair comparison.

Our tool **crest** is split into different binaries depending on the analysis. The most important ones are **crest-cr** for confluence and **crest-sn** for termination. We use those two including an additional flag to allow at most 8 threads for the concurrent setup. The default strategy for confluence uses all methods concurrently and where specific methods are tested we restrict to those using our strategy flag. For the default termination setup we use reduction pairs including

¹¹ <https://github.com/sosy-lab/benchexec/>

¹² <https://ari-cops.uibk.ac.at/ARI/>

Table 1. Confluence analysis of examples.

tool	1	2	3	6	7	8
crest	✓	✓	✓	?	✗	✓
CRaris	?	?	?	?	?	?
Ctrl	✓	?	?	?	?	?
prototype	✓	✓	?	?	?	?

dependency graph analysis, recursive path order, (special) value criterion and subterm criterion.

Cora, Ctrl and the prototype of [34] do not accept the ARI format as input. We have developed transformation tools which (try to) transform an ARI benchmark into their respective input. This might not always be possible, hence the transformation tool might fail, which is the reason why we do not distinguish tool (parse) errors from "MAYBE".

Examples. In Table 1 we compare the LCTRS confluence tools on the examples in this paper. **crest** only fails on Example 6, which is a confluent LCTRS, but for which no automatable method is known.

Confluence Competition. The last (2024) Confluence Competition¹³ hosted the first LCTRS category, with **crest** and **CRaris** as participants. The former achieved 67 confluence and 26 non-confluence proofs on a total of 100 selected problems from the ARI database. **CRaris**, which does not (yet) implement techniques for non-confluence achieved 54 confluence proofs. Currently, **crest** is the only tool utilizing a criterion for non-confluence of LCTRSs.

Confluence. All confluence criteria implemented in **crest**, except (C2), require left-linearity. For (C3) right-linearity is also required. Left- and right-linearity is checked only on the non-logical variables. Table 2 presents a summary of the confluence methods implemented in **crest**. The full set of benchmarks consists

¹³ <https://project-coco.uibk.ac.at/2024/index.php>

```
(format LCTRS :smtlib 2.6)
(theory Ints)
(fun f (-> Int Int))
(fun g (-> Int Int Int))
(fun a Int)
(rule (f a) (g 4 4))
(rule a (g (+ 1 1) (+ 3 1)))
(rule (g x y) (f (g z y)) :guard (= z (- x 2)))
```

Fig. 2. ARI file 1529 (without sort annotations and meta information).

Table 2. Confluence analysis using methods in **crest** on 107 LCTRSs.

criterion	solved	time (AVG)	time (total)
termination and joinable critical pairs (C2)	50	4.55 s	487 s
orthogonality (C1)	62	0.10 s	11 s
weak orthogonality (C1)	65	0.12 s	13 s
strongly closed critical pairs (C3)	56	1.21 s	129 s
parallel closed critical pairs (C4)	66	0.44 s	47 s
almost parallel closed critical pairs (C4)	70	11.03 s	1180 s
development closed critical pairs (C5)	66	0.39 s	42 s
almost development closed critical pairs (C5)	71	2.06 s	220 s
parallel closed parallel critical pairs (C6)	71	13.93 s	1490 s
all confluence methods (C1)–(C6)	72	8.40 s	899 s
non-confluence (Lemma 1)	26	1.96 s	210 s
methods (C1)–(C6) + (Lemma 1)	98	1.84 s	197 s
total solved	98	—	—

of the 107 problems in the ARI database. **crest** can prove in a full run with all methods enabled 72 confluent and 26 non-confluent. Of the remaining 9 problems, 2 result in "MAYBE" and 7 in a timeout. Interesting to observe is that (almost) development closedness is way faster than (almost) parallel closedness, which may be due to the fact that less multi-steps than parallel steps are needed to turn a constrained critical pair into a trivial one. The number 72 is explained by the fact that (C5) and (C6) are incomparable: (C5) succeeds on the problem in Fig. 1 but fails on the one in Fig. 2, while the opposite holds for (C6).

In Table 3 we compare all confluence tools on the same 107 LCTRS problems. **Ctrl** supports only weak orthogonality and **CRaris** in addition the Knuth–Bendix criterion. Overall **crest** is able to solve 92 % of the LCTRS problems in the current ARI database and this percentage is reached even if the timeout is restricted to

Table 3. Confluence analysis of LCTRS tools on 107 LCTRSs.

tool	✓	✗	?	†	solved	time (AVG)	time (total)
CRaris	58	0	49	—	54 %	0.13 s	14 s
crest	72	26	9	—	92 %	1.84 s	197 s
Ctrl	54	0	49	4	50 %	0.17 s	18 s
prototype	67	0	37	3	63 %	1.14 s	122 s
total solved	72	26	—	—	92 %	—	—

Table 4. Termination analysis using methods in *crest* on 107 LCTRSs.

method	solved	time (AVG)	time (total)
DP graph (T1)	9	0.08 s	9 s
recursive path order (T2)	27	0.11 s	12 s
recursive path order (T1), (T2)	28	0.11 s	12 s
subterm criterion (T1), (T5)	12	0.12 s	13 s
value criterion (T1), (T3)	34	0.13 s	14 s
special value criterion (T1), (T6)	70	0.12 s	13 s
reduction pairs no SVC (T1)–(T5)	37	0.14 s	15 s
default (T1)–(T6)	74	0.15 s	16 s
total solved	74	—	—

10 seconds. The prototype of [34] supports the methods (C1), (C3), (C4) and proves 67 (63 %) confluent within 122 seconds.

Termination. In Table 4 we compare the different termination methods in *crest*. The "dependency graph" method corresponds to (T1) with a check for the absence of SCCs, "recursive path order" corresponds to (T2), "subterm criterion" to (T5), "(special) value criterion" to (T3) ((T6)) and "reduction pairs" to (T4). The methods annotated with (T1) work on DP problems and are applied after an initial dependency graph analysis. The method "reduction pairs no SVC" uses (T2), (T3) and (T5) and "default" includes additionally (T6). The latter constitutes the current default setup in *crest*.

We continue the evaluation by comparing *crest* to other termination tools for LCTRSs. For this comparison we use the higher-order tool *Cora* and *Ctrl*. The experiments in Table 5 show that the tools are comparable in strength on the LCTRS benchmarks in the ARI database, which is not that surprising as the implemented methods are similar. All tools together prove 73 % of the LCTRSs in the ARI database terminating. All those tools fail on the bit vector problem in Fig. 3 whereas CRaris is able to prove termination (Naoki Nishida, personal

Table 5. Termination analysis of LCTRS tools on 107 LCTRSs.

tool	✓	?	†	solved	time (AVG)	time (total)
Cora	71	30	6	66 %	2.47 s	264 s
crest	74	33	—	69 %	0.15 s	16 s
Ctrl	74	29	4	69 %	0.96 s	103 s
total solved	78	—	—	73 %	—	—

```

(format LCTRS :smtlib 2.6)
(theory FixedSizeBitVectors)
(fun cnt (-> (_ BitVec 4) (_ BitVec 4)))
(fun u1 (-> (_ BitVec 4) (_ BitVec 4) (_ BitVec 4) (_ BitVec 4)))
(rule (cnt x) (u1 x #b0000 #b0000) )
(rule (u1 x i z) (u1 x (bvadd i #b0001) (bvadd z #b0001))
  :guard (bvult i x)))
(rule (u1 x i z) z :guard (not (bvult i x)))

```

Fig. 3. ARI file 1605 (without sort annotations and meta information).

communication). A fork of the official version of Ctrl¹⁴ implements the technique of [33] for non-termination of LCTRSs. Initial experiments reveal that it succeeds to prove non-termination of 8 problems in Table 5.

Term Rewrite Systems. In the final experiment we compare *crest* with the state-of-the-art in automated confluence proving for TRSs. After parsing an input TRS, *crest* attaches a single sort to all function symbols and variables, and adds an empty constraint to all rules. At this point the TRS can be analyzed as an LCTRS. We compare *crest* to the latest winner of the TRS category in the Confluence Competition, CSI [32], on the 566 TRS benchmarks in the ARI database. The results can be seen in Table 6. Keeping in mind that there is some overhead in the analysis of *crest* on TRSs as all its methods are geared towards the constrained setting, the 31 % mark is not a bad result. Here it is important to note that CSI has been actively developed over a ten-year period and utilizes many more confluence methods—there is several decades of research on confluence analysis of TRSs while LCTRS confluence analysis is still in its infancy.

7 Conclusion and Future Work

In this paper we presented *crest*, an open-source tool for automatically proving (non-)confluence and termination of LCTRSs. Detailed experiments were provided to show the power of *crest*.

¹⁴ <https://github.com/bytekid/ctrl>

Table 6. Confluence analysis of *crest* and CSI on 566 TRSs.

tool	✓	✗	?	solved	time (AVG)	time (total)
<i>crest</i>	100	73	393	31 %	15.87 s	8980 s
CSI	259	192	115	80 %	6.25 s	3540 s
total solved	259	192	—	80 %	—	—


```

(format LCTRS :smtlib 2.6)
(theory Reals)
(fun sumroot (-> Real Real))
(fun sqrt (-> Real Real))
(rule (sumroot x) 0.0 :guard (>= 0.0 x))
(rule (sumroot x) (+ (sqrt x) (sumroot (- x 1.0)))
  :guard (not (>= 0.0 x)))

```

Fig. 4. ARI file 1549 (without meta information).

In order to further strengthen the (non-)confluence analysis in *crest* we plan to adapt powerful methods like order-sorted decomposition [12] and redundant rules [31, 38] for plain term rewriting to the constrained setting. Labeling techniques [41] are also on the agenda. The same holds for termination analysis. Natural candidates are matrix interpretations [11] as well as the higher-order methods in [14]. Especially termination problems on real values, like the one in Fig. 4, should be supported in future. Also non-termination analysis of LCTRSs [33] is of interest. Completion, which is supported in Ctrl [39], is another topic for a future release of *crest*. In a recent paper [2] the semantics of LCTRSs is investigated. In that context, concepts like checking consistency of constrained theories are relevant, which are worthy to investigate from an automation viewpoint.

Since constrained rewriting is highly complex [35, Section 3], a formalization of the implemented techniques in a proof assistant like Isabelle/HOL is important. The recent advances in the formalization and subsequent certification of advanced confluence techniques [16, 21, 22] for plain rewriting in connection with the transformation in [35, Section 4] make this a realistic goal.

Finally, to improve the user experience we aim at a convenient web interface and a richer command-line strategy.

Code and Availability Statement. The source code and data that support the contributions of this work are freely available in the Zenodo repository “*crest* - Constrained REwriting Software Tool: Artifact for TACAS 2025” at <https://doi.org/10.5281/zenodo.13969852> [36]. The authors confirm that the data supporting the findings of this study are available within the paper and the artifact.

Acknowledgments. We thank Fabian Mitterwallner for valuable discussions on automation. We are grateful to the authors of the tools used in the experiments for their help in obtaining executables and useful insights about their usage. The insightful comments and suggestions provided by the reviewers greatly improved the presentation of the paper.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aoto, T., Hirokawa, N., Kim, D., Kojima, M., Middeldorp, A., Mitterwallner, F., Nishida, N., Saito, T., Schöpf, J., Shintani, K., Thiemann, R., Yamada, A.: A new format for rewrite systems. In: Proc. 12th International Workshop on Confluence. pp. 32–37 (2023), available at <http://cl-informatik.uibk.ac.at/iwc/iwc2023.pdf>
2. Aoto, T., Nishida, N., Schöpf, J.: Equational theories and validity for logically constrained term rewriting. In: Rehof, J. (ed.) Proc. 9th International Conference on Formal Structures for Computation and Deduction. Leibniz International Proceedings in Informatics, vol. 299, pp. 31:1–31:21 (2024). <https://doi.org/10.4230/LIPIcs.FSCD.2024.31>
3. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. *Theoretical Computer Science* **236**, 133–178 (2000). [https://doi.org/10.1016/S0304-3975\(99\)00207-8](https://doi.org/10.1016/S0304-3975(99)00207-8)
4. Avanzini, M., Moser, G., Schaper, M.: TcT: Tyrolean Complexity Tool. In: Chechik, M., Raskin, J.F. (eds.) Proc. 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 9636, pp. 407–423 (2016). https://doi.org/10.1007/978-3-662-49674-9_24
5. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998). <https://doi.org/10.1017/CBO9781139172752>
6. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Proc. 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 13243, pp. 415–442 (2022). https://doi.org/10.1007/978-3-030-99524-9_24
7. Ciobăcă, S., Lucanu, D., Buruiană, A.S.: Operationally-based program equivalence proofs using LCTRSs. *Journal of Logical and Algebraic Methods in Programming* **135**, 100894 (2023). <https://doi.org/10.1016/j.jlamp.2023.100894>
8. Ciobăcă, S., Lucanu, D.: A coinductive approach to proving reachability properties in logically constrained term rewriting systems. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Proc. 9th International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, vol. 10900, pp. 295–311 (2018). https://doi.org/10.1007/978-3-319-94205-6_20
9. Dershowitz, N.: Orderings for term-rewriting systems. *Theoretical Computer Science* **17**(3), 279–301 (1982). [https://doi.org/10.1016/0304-3975\(82\)90026-3](https://doi.org/10.1016/0304-3975(82)90026-3)
10. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Proc. 26th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 8559, pp. 737–744 (2014). https://doi.org/10.1007/978-3-319-08867-9_49
11. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning* **40**(2-3), 195–220 (2007). <https://doi.org/10.1007/s10817-007-9087-9>
12. Felgenhauer, B., Middeldorp, A., Zankl, H., van Oostrom, V.: Layer systems for proving confluence. *ACM Transactions on Computational Logic* **16**(2:14), 1–32 (2015). <https://doi.org/10.1145/2710017>
13. Fuhs, C., Kop, C., Nishida, N.: Verifying procedural programs via constrained rewriting induction. *ACM Transactions on Computational Logic* **18**(2), 14:1–14:50 (2017). <https://doi.org/10.1145/3060143>

14. Guo, L., Hagens, K., Kop, C., Vale, D.: Higher-order constrained dependency pairs for (universal) computability. In: Kráľovič, R., Kučera, A. (eds.) Proc. 49th International Symposium on Mathematical Foundations of Computer Science. Leibniz International Proceedings in Informatics, vol. 306, pp. 57:1–57:15 (2024). <https://doi.org/10.4230/LIPIcs.MFCS.2024.57>
15. Guo, L., Kop, C.: Higher-order LCTRSs and their termination. In: Weirich, S. (ed.) Proc. 33rd European Symposium on Programming. Lecture Notes in Computer Science, vol. 14577, pp. 331–357 (2024). https://doi.org/10.1007/978-3-031-57267-8_13
16. Hirokawa, N., Kim, D., Shintani, K., Thiemann, R.: Certification of confluence- and commutation-proofs via parallel critical pairs. In: Timany, A., Traytel, D., Pientka, B., Blazy, S. (eds.) Proc. 13th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 147–161. ACM (2024). <https://doi.org/10.1145/3636501.3636949>
17. Hirokawa, N., Middeldorp, A.: Dependency pairs revisited. In: van Oostrom, V. (ed.) Proc. 15th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 3091, pp. 249–268 (2004). https://doi.org/10.1007/978-3-540-25979-4_18
18. Jouannaud, J.P., Rubio, A.: The higher-order recursive path ordering. In: Proc. 14th Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 402–411 (1999). <https://doi.org/10.1109/LICS.1999.782635>
19. Jürgen, Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) Proc. 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. Lecture Notes in Computer Science, vol. 3452, pp. 301–331 (2005). https://doi.org/10.1007/978-3-540-32275-7_21
20. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press (1970). <https://doi.org/10.1016/B978-0-08-012975-4.50028-X>
21. Kohl, C., Middeldorp, A.: A formalization of the development closedness criterion for left-linear term rewrite systems. In: Krebbers, R., Traytel, D., Pientka, B., Zdancewic, S. (eds.) Proc. 12th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 197–210 (2023). <https://doi.org/10.1145/3573105.3575667>
22. Kohl, C., Middeldorp, A.: Formalizing almost development closed critical pairs. In: Naumowicz, A., Thiemann, R. (eds.) Proc. 14th International Conference on Interactive Theorem Proving. Leibniz International Proceedings in Informatics, vol. 268, pp. 38:1–38:8 (2023). <https://doi.org/10.4230/LIPIcs.ITP.2023.38>
23. Kojima, M., Nishida, N.: From starvation freedom to all-path reachability problems in constrained rewriting. In: Hanus, M., Inclezan, D. (eds.) Proc. 25th International Symposium on Practical Aspects of Declarative Languages. Lecture Notes in Computer Science, vol. 13880, pp. 161–179 (2023). https://doi.org/10.1007/978-3-031-24841-2_11
24. Kop, C.: Termination of LCTRSs. CoRR **abs/1601.03206** (2016). <https://doi.org/10.48550/ARXIV.1601.03206>
25. Kop, C., Nishida, N.: Term rewriting with logical constraints. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) Proc. 9th International Symposium on Frontiers of Combining Systems. Lecture Notes in Artificial Intelligence, vol. 8152, pp. 343–358 (2013). https://doi.org/10.1007/978-3-642-40885-4_24

26. Kop, C., Nishida, N.: Automatic constrained rewriting induction towards verifying procedural programs. In: Garrigue, J. (ed.) Proc. 12th Asian Symposium on Programming Languages and Systems. Lecture Notes in Computer Science, vol. 8858, pp. 334–353 (2014). https://doi.org/10.1007/978-3-319-12736-1_18
27. Kop, C., Nishida, N.: Constrained Term Rewriting tool. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) Proc. 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. Lecture Notes in Artificial Intelligence, vol. 9450, pp. 549–557 (2015). https://doi.org/10.1007/978-3-662-48899-7_38
28. Kusakari, K., Sakai, M.: Enhancing dependency pair method using strong computability in simply-typed term rewriting. *Applicable Algebra in Engineering, Communication and Computing* **18**(5), 407–431 (2007). <https://doi.org/10.1007/S00200-007-0046-9>
29. Matsumi, A., Nishida, N., Kojima, M., Shin, D.: On singleton self-loop removal for termination of LCTRSs with bit-vector arithmetic. *CoRR* **abs/2307.14094** (2023). <https://doi.org/10.48550/arXiv.2307.14094>
30. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 4963, pp. 337–340 (2008). https://doi.org/10.1007/978-3-540-78800-3_24
31. Nagele, J., Felgenhauer, B., Middeldorp, A.: Improving automatic confluence analysis of rewrite systems by redundant rules. In: Fernández, M. (ed.) Proc. 26th International Conference on Rewriting Techniques and Applications. *Leibniz International Proceedings in Informatics*, vol. 36, pp. 257–268 (2015). <https://doi.org/10.4230/LIPIcs.RTA.2015.257>
32. Nagele, J., Felgenhauer, B., Middeldorp, A.: CSI: New evidence — a progress report. In: de Moura, L. (ed.) Proc. 26th International Conference on Automated Deduction. Lecture Notes in Artificial Intelligence, vol. 10395, pp. 385–397 (2017). https://doi.org/10.1007/978-3-319-63046-5_24
33. Nishida, N., Winkler, S.: Loop detection by logically constrained term rewriting. In: Piskac, R., Rümmer, P. (eds.) Proc. 10th International Conference on Verified Software: Theories, Tools, and Experiments. Lecture Notes in Computer Science, vol. 11294, pp. 309–321 (2018). https://doi.org/10.1007/978-3-030-03592-1_18
34. Schöpf, J., Middeldorp, A.: Confluence criteria for logically constrained rewrite systems. In: Pientka, B., Tinelli, C. (eds.) Proc. 29th International Conference on Automated Deduction. Lecture Notes in Artificial Intelligence, vol. 14132, pp. 474–490 (2023). https://doi.org/10.1007/978-3-031-38499-8_27
35. Schöpf, J., Mitterwallner, F., Middeldorp, A.: Confluence of logically constrained rewrite systems revisited. In: Benz Müller, C., Heule, M.J., Schmidt, R.A. (eds.) Proc. 12th International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, vol. 14740, pp. 298–316 (2024). https://doi.org/10.1007/978-3-031-63501-4_16
36. Schöpf, J., Middeldorp, A.: *crest* - Constrained REwriting Software Tool: Artifact for TACAS 2025 (Oct 2024). <https://doi.org/10.5281/zenodo.13969852>
37. Schöpf, J., Middeldorp, A.: Automated analysis of logically constrained rewrite systems using *crest*. *CoRR* **abs/2501.05240** (2025). <https://doi.org/10.48550/arXiv.2501.05240>
38. Shintani, K., Hirokawa, N.: Compositional confluence criteria. *Logical Methods in Computer Science* **20**(1) (2024). [https://doi.org/10.46298/lmcs-20\(1:6\)2024](https://doi.org/10.46298/lmcs-20(1:6)2024)

39. Winkler, S., Middeldorp, A.: Completion for logically constrained rewriting. In: Kirchner, H. (ed.) Proc. 3rd International Conference on Formal Structures for Computation and Deduction. Leibniz International Proceedings in Informatics, vol. 108, pp. 30:1–30:18 (2018). <https://doi.org/10.4230/LIPICs.FSCD.2018.30>
40. Winkler, S., Moser, G.: Runtime complexity analysis of logically constrained rewriting. In: Fernández, M. (ed.) Proc. 30th International Symposium on Logic-Based Program Synthesis and Transformation. Lecture Notes in Computer Science, vol. 12561, pp. 37–55 (2021). https://doi.org/10.1007/978-3-030-68446-4_2
41. Zankl, H., Felgenhauer, B., Middeldorp, A.: Labelings for decreasing diagrams. Journal of Automated Reasoning **54**(2), 101–133 (2015). <https://doi.org/10.1007/s10817-014-9316-y>

Open Access. This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

