Tools in Term Rewriting for Education

Sarah Winkler
Università di Verona, Italy
sarahmaria.winkler@univr.it

Aart Middeldorp
University of Innsbruck, Austria
aart.middeldorp@uibk.ac.at

Term rewriting is a Turing complete model of computation. When taught to students of computer science, key properties of computation as well as techniques to analyze programs on an abstract level are conveyed. This paper gives a swift introduction to term rewriting and presents several automatic tools to analyze term rewrite systems which were developed by the Computational Logic Group at the University of Innsbruck. These include the termination tool $\mathsf{T_TT_2}$, the confluence prover CSI, the completion tools mkbTT and KBCV, the complexity tool TcT, the strategy tool AutoStrat, as well as FORT, an implementation of the decision procedure for the first-order theory for a decidable class of rewrite systems. Besides its applications in research, this software pool has also proved invaluable for teaching, e.g., in multiple editions of the International Summer School on Rewriting.

1 Introduction

Rewriting is a pervasive concept in mathematics, computer science, and other areas: Simplification of expressions constitutes rewriting, the execution of a program can be seen as a rewrite sequence on program states, and in fact probably almost any development according to a set of fixed rules can be considered rewriting. In *term rewriting*, we assume that the objects which are rewritten are terms. This yields a powerful formalism which is crucial for simplification in automated theorem proving, it provides tools to analyze security protocols, it can be used to model the development of RNA structures, but it is also a versatile method in program verification, to name only a few application areas. In fact, term rewriting is a Turing-complete model of computation, and provides methods to investigate important properties of computation and simplification processes on an abstract level [4, 32].

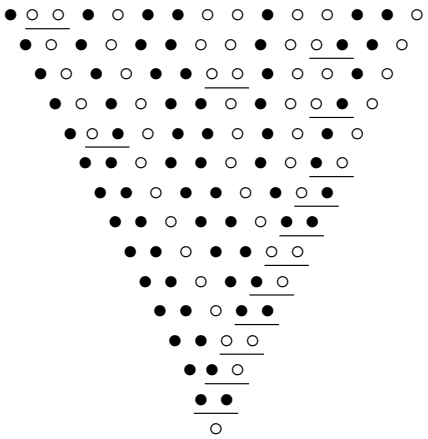
This includes ubiquituous properties related to termination, determinism, and complexity. As a simple but powerful model of computation, term rewriting can in particular also convey program analysis on an abstract level to students of computer science and related fields. We illustrate some properties by means of a simple example.

Example 1.1 (Coffee Bean Game [6]). Coffee beans come in two kinds called black (•) and white (○). A two-player game starts with a random sequence of black and white beans. In a move, a player must take two adjacent beans and put back one bean, according to the following set of rules R_1:

$$\bullet\ \bullet \to \circ \qquad \circ\ \circ \to \circ \qquad \bullet\ \circ \to \bullet \qquad \circ\ \bullet \to \bullet$$

The player who puts the last white bean wins. For instance, the following is a valid game:

P. Quaresma, W. Neuper and J. Marcos (Eds.): Theorem proving components for Educational software 2019 (ThEdu'19)
EPTCS 313, 2020, pp. 54–72, doi:10.4204/EPTCS.313.4



In this case the player who started lost, since the last white bean was put in the 14th move. A number of interesting questions can be asked about such a game: Which moves should the respective players perform to win? Are there game states which are equivalent in the sense that they offer the same opportunities to each of the players? In short, is there a winning strategy for one of the players? While it is obvious that the above game terminates, is this still the case for the modified game using the rules R_2:

$$\bullet\ \bullet \to \circ\ \circ\ \circ\ \circ \qquad \circ\ \circ \to \circ \qquad \bullet\ \circ \to \circ\ \circ\ \circ\ \bullet \qquad \circ\ \bullet \to \bullet$$

and if yes, how many steps are needed?

This paper advocates rewriting to answer these questions and many others that we will motivate by examples. As manual analysis of term rewrite systems often turns out to be tedious, a variety of tools has been developed in the last two decades which perform powerful analysis tasks automatically. We here focus on tools that have been developed at the Computational Logic group at the University of Innsbruck since these are the tools we are most familiar with.

This paper gives a concise introduction to term rewriting. We introduce some of the most widely investigated properties of term rewrite systems, and motivate their relevance by examples from different domains. Rather than elaborating the often complicated methods developed to analyze these properties, we show how tools can effectively be used to inspect term rewrite systems automatically. In this spirit we discuss termination (Section 3), confluence (Section 4), completion as a means to decide the validity problem (Section 5), the first order theory of rewriting (Section 6), evaluation strategies (Section 7), and derivational complexity (Section 8). We conclude in Section 9 with remarks on current research.

2 Preliminaries

We assume basic familiarity with term rewriting [4, 32], but recall some key notions and notation. Given a signature $\mathscr{F}$ and a set of variables $\mathscr{V}$, we consider the set of terms $\mathscr{T}$ built up from $\mathscr{F}$ and $\mathscr{V}$. *Positions* are strings of positive integers which are used to address subterms. We write $t|_p$ for the subterm of t at position p and $t[u]_p$ denotes the term that is obtained from t by replacing its subterm $t|_p$ with u. A *substitution* σ is a mapping from variables to terms such that $\sigma(x) \neq x$ for only finitely many x. An *equation* is a pair of terms $s \approx t$, and a *rewrite rule* is a pair of terms denoted as $\ell \to r$ such that $\ell \notin \mathscr{V}$ and all variables in r also occur in ℓ. An equational system (ES) is a set of equations, while a *term rewrite system* (TRS) refers to a set of rewrite rules.

The *rewrite relation* induced by a TRS R is defined as $s \to_R t$ if and only if $s|_p = \ell\sigma$ and $t = s[r\sigma]_p$ for some position p, substitution σ, and rewrite rule $\ell \to r$ in R. The relations $\leftrightarrow_R$, $\to_R^+$, and $\to_R^*$ denote the symmetric, transitive, and reflexive transitive closure of $\to_R$, respectively, while the reflexive, symmetric, and transitive closure of $\to_R$ is denoted $\leftrightarrow_R^*$ and called *conversion*. Two terms s and t are *convertible* if there exists a conversion $s \leftrightarrow_R^* t$. We further use $\downarrow_R$ as abbreviation for the *joinability relation* ${}_R^*\!\leftarrow \cdot \to_R^*$ and $\uparrow_R$ as abbreviation for the *meetability relation* $\to_R^* \cdot {}_R^*\!\leftarrow$. Here $\cdot$ denotes relation composition. A *normal form* with respect to a TRS R is a term t such that there is no term s with $t \to_R s$. We also write $u \to_R^! t$ if $u \to_R^* t$ and t is a normal form.

Some further concepts will be introduced in later sections when they are needed.

3 Termination

Termination is very often a desired feature of rewrite systems, and thus one of the most studied properties.

Definition 3.1. *A* TRS R *is terminating if there is no infinite rewrite sequence* $t_0 \to_R t_1 \to_R t_2 \to_R \cdots$.

Example 3.1. We revisit Example 1.1 from the introduction. It is obvious that the TRS R_1 terminates since the number of beans decreases by one with every move. Though the case of R_2 is less obvious, it turns out that also this TRS terminates. Many different techniques can be harnessed to show this. Here we use this example to illustrate a popular technique to show termination based on *interpretations*.

Suppose we take as carrier set the natural numbers and use $\circ_A(x) = x+1$ and $\bullet_A(x) = 4x+1$ as interpretations. The terms in the four rewrite rules

$$\bullet(\bullet(x)) \to \circ(\circ(\circ(\circ(x)))) \qquad \circ(\circ(x)) \to \circ(x) \qquad \bullet(\circ(x)) \to \circ(\circ(\circ(\bullet(x)))) \qquad \circ(\bullet(x)) \to \bullet(x)$$

then correspond to the following polynomials, where independent of the value of x the left-hand side is always greater than the right-hand side:

$$16x+5 >_{\mathbb{N}} x+4 \qquad x+2 >_{\mathbb{N}} x+1 \qquad 4x+5 >_{\mathbb{N}} x+4 \qquad 4x+2 >_{\mathbb{N}} 4x+1$$

Since every rewrite step results in a strict decrease, and $>_{\mathbb{N}}$ is a well-founded order, this linear polynomial interpretation shows that no infinite rewrite sequence can possibly exist.

T$_{\text{T}}$T$_2$ [14] is a tool to show termination of TRSs, available both via a web interface and as a stand-alone executable.[1] It is beyond the scope of this paper to describe all implemented techniques; we only mention that a great variety of approaches is supported, including different term orders, interpretations over various domains, modularization of termination problems according to the powerful dependency pair framework, and numerous specialized routines. The tool also provides support for relative termination, as well as means to show termination with respect to strategies (see Section 7) and nontermination. Upon success, T$_{\text{T}}$T$_2$ outputs all details of the (non)termination proof such as interpretations, parameters of orderings, or a counterexample in case termination was disproved. This helps students (as well as researchers) to understand the result.

Specialized support for teaching was added recently [28]. We mention the encoding of the state of the web interface into a URL, which allows examples that are used for teaching to be directly loaded into the web interface by a simple mouse click from the slides. This avoids time-consuming and error-prone manipulations during a lecture or talk. To illustrate this, clicking here opens the web interface of T$_{\text{T}}$T$_2$ in

[1] `http://cl-informatik.uibk.ac.at/software/ttt2/`

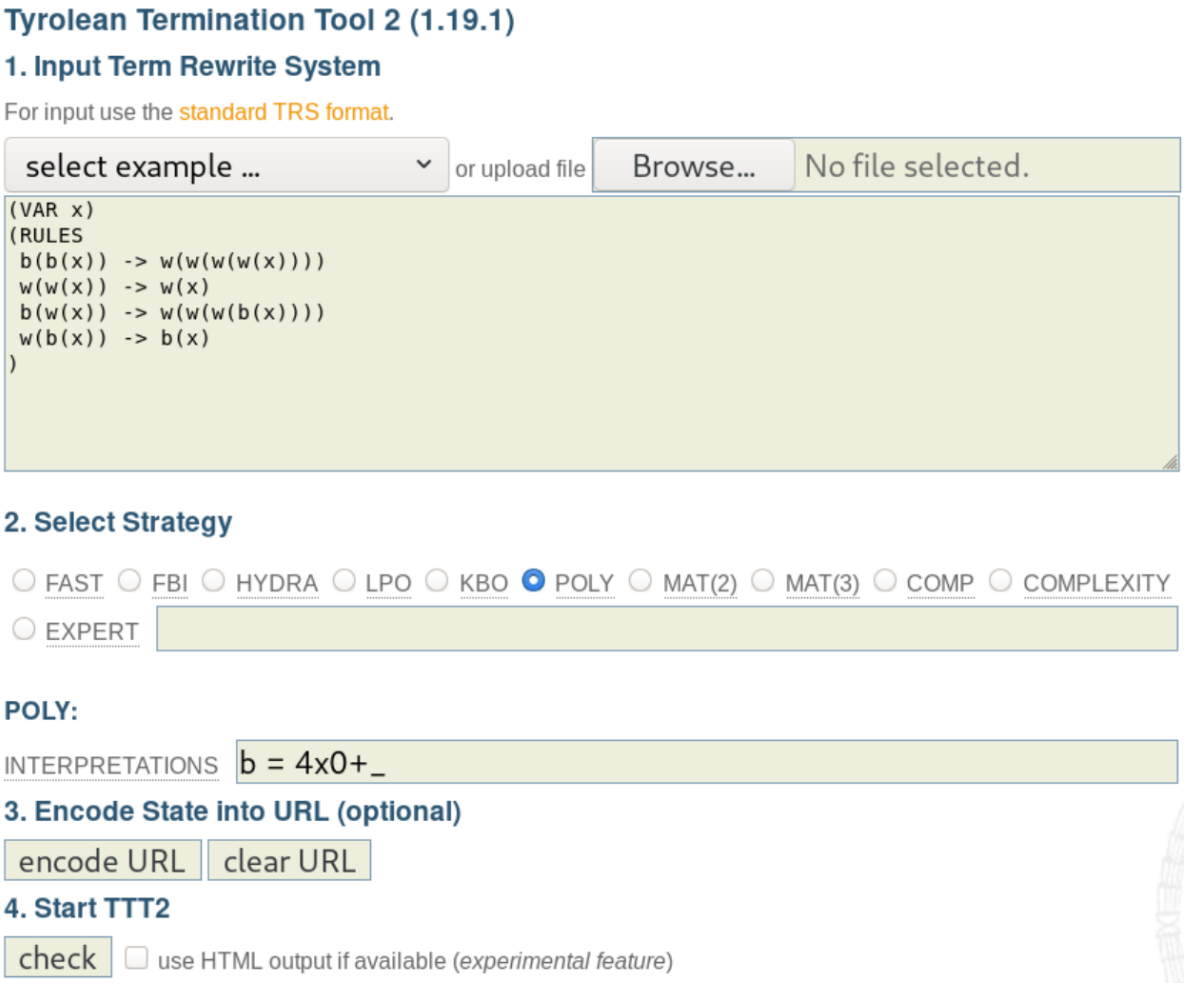


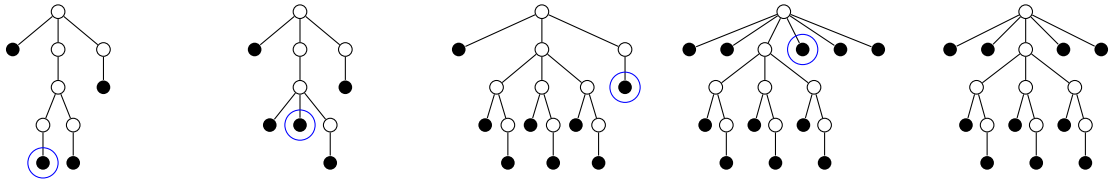
Figure 1: The web interface of $\mathsf{T_T T_2}$.

a browser with the above bean rules and a partial polynomial interpretation, indicating that we look for an interpretation with $\bullet_A(x) = 4x + c$ for some constant c. A screenshot is shown in Figure 1. Guiding termination methods by providing some of the parameters is also supported for the Knuth–Bendix order (KBO), the lexicographic path order (LPO), and matrix interpretations, since these are the termination methods taught in the bachelor course on term rewiting at the University of Innsbruck. This feature is useful for students in multiple respects: Sometimes exercises demand to complete a given partial interpretation, in other cases an interpretation of a particular shape is demanded; in both cases students can check their solutions with this functionality. But it also helps them to refine their own incomplete solutions, and can be used to show that, for instance, a certain precedence relation between two function symbols does *not* work for LPO or KBO.

We conclude this section with another example where termination is less obvious.

Example 3.2 (Battle of Hydra and Hercules). The mythological monster Hydra is a dragon-like creature with multiple heads. Whenever Hercules in his fight chops off a head, more and more new heads can grow instead, since the beast gets increasingly angry. Here we model a Hydra as an unordered tree. If Hercules cuts off a leaf corresponding to a head, the tree is modified in the following way: If the cut-off

node h has a grandparent n, then the branch from n to the parent of h gets multiplied, where the number of copies depends on the number of decapitations so far. Hydra dies if there are no heads left, in that case Hercules wins. The following sequence shows an example fight:



1 2 3 4 5

Though the number of heads can grow considerably in one step, it turns out that the fight always terminates, and Hercules will win independent of his strategy. This can be shown by an argument based on ordinals. Touzet modeled this process as a TRS $\mathscr{H}$ [34]. However, derivations may get very long: their length cannot be described by a multiple recursive function in the size of the initial monster. This was one of the reasons that Touzet's TRS remained out of reach for automatic tools for more than a decade, until *ordinal interpretations* were developed to deal with such systems [39]. Nowadays, T$_{\mathsf{T}}$T$_2$ can show termination of $\mathscr{H}$ automatically, using an implementation of this technique.

4 Confluence

In many applications, it is of interest to know whether the process described by a TRS satisfies properties related to determinism. For instance, in Example 1.1 one would like to know whether different strategies of the players lead to different results. The most studied property in rewriting in this context is confluence, defined below.

Definition 4.1. *A* TRS R *is* locally confluent *if* ${}_R\!\leftarrow \cdot \rightarrow_R \subseteq \downarrow_R$*, and* confluent *if* $\uparrow_R \subseteq \downarrow_R$ *holds.*

According to a famous result by Newman [18], these two properties coincide if a TRS is terminating.

Lemma 4.1. *A terminating and locally confluent* TRS *is confluent.* □

The definiton of confluence imposes a condition on all *peaks*, i.e., rewrite sequences of the form ${}^*_R\!\leftarrow \cdot \rightarrow^*_R$, of which there might be infinitely many, in addition to the fact that $\rightarrow^*_R$ is in general undecidable. Fortunately, it is known that a TRS R is locally confluent if all its critical pairs $\mathsf{CP}(R)$ are joinable, of which there are only finitely many. Thus it turns out that it suffices to consider finitely many peaks for local confluence, as expressed by Lemma 4.2 below.

Definition 4.2. *Let $\ell_1 \to r_1$ and $\ell_2 \to r_2$ be renamings of rewrite rules in R without common variables, such that the following conditions are satisfied:*

- *p is a non-variable position in ℓ_2,*
- *σ is a most general unifier of $\ell_2|_p$ and ℓ_1, and*
- *if $p = \varepsilon$ then $\ell_1 \to r_1$ and $\ell_2 \to r_2$ are not variants.*

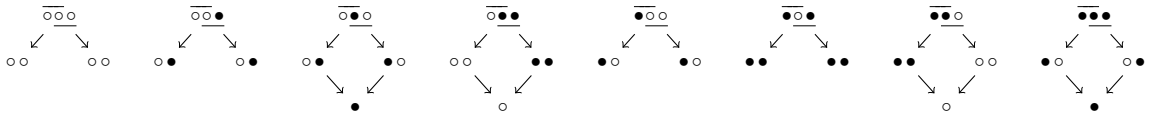
The triple $\langle \ell_1 \to r_1, p, \ell_2 \to r_2 \rangle$ constitutes a critical overlap*, and $\ell_2\sigma[r_1\sigma]_p \approx r_2\sigma$ is a* critical pair *of R.*

Two rewrite steps $s\ {}_R\!\leftarrow \cdot \rightarrow_R t$ are said to form a *critical peak* if $s \approx t$ is a critical pair, and the critical pair is called joinable if $s \downarrow_R t$. In the sequel we denote the set of all critical pairs of R by $\mathsf{CP}(R)$. The following lemma explains the importance of critical pairs for local confluence.

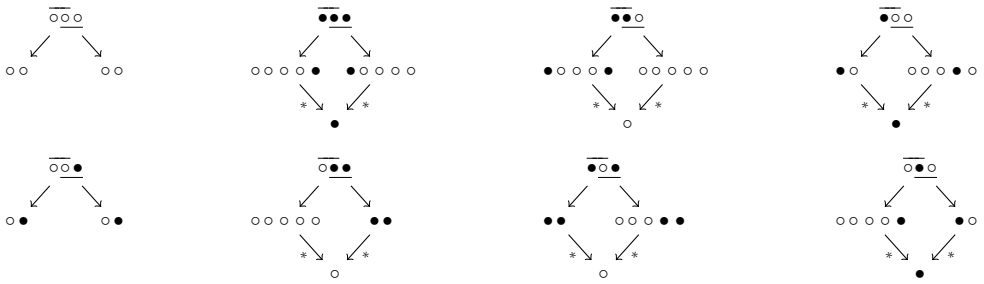
Lemma 4.2 (Critical Pair Lemma [12]). *Consider a* TRS R *and terms* s *and* t*. If there is a peak* $s \mathrel{{}_R\!\leftarrow} \cdot \rightarrow_R t$ *then* $s \downarrow_R t$ *or* $s \leftrightarrow_{\mathsf{CP}(R)} t$. □

In connection with Newman's Lemma, the Critical Pair Lemma implies that confluence is decidable for terminating systems. This result can be used to investigate determinism of the bean game given in the introduction.

Example 4.1. We investigate confluence of the TRS R_1 from Example 1.1. Since the TRS is terminating, confluence and local confluence coincide. We thus analyze the critical pairs of R_1. The following eight diagrams show all critical peaks of R:



In each of these local peaks, the rewrite steps either lead to the same result, or the resulting two terms have a common reduct that is reached in a single step from both. Thus R_1 is locally confluent by Lemma 4.2, and confluent by Lemma 4.1 since R_1 is terminating. This implies that the result of the game only depends on the initial configuration. A similar analysis applies to the TRS R_2 of Example 1.1, although finding a common reduct requires more steps:



For systems that are non-terminating, joinability of critical pairs is insufficient for confluence. By forbidding criticial pairs and imposing the condition that left-hand sides of rules do not contain repeated variables (left-linearity), confluence is guaranteed [27]. This syntactic criterion is called *orthogonality*. We give an example.

Example 4.2. The following TRS R models a functional program to enumerate prime numbers:

$$\begin{aligned}
\mathsf{primes} &\to \mathsf{sieve}(\mathsf{from}(\mathsf{s}(\mathsf{s}(0)))) & \mathsf{sieve}(0 : y) &\to \mathsf{sieve}(y)\\
\mathsf{from}(n) &\to n : \mathsf{from}(\mathsf{s}(n)) & \mathsf{sieve}(\mathsf{s}(n) : y) &\to \mathsf{s}(n) : \mathsf{sieve}(\mathsf{filter}(n, y, n))\\
\mathsf{take}(0, y) &\to \mathsf{nil} & \mathsf{filter}(0, x : y, m) &\to 0 : \mathsf{filter}(m, y, m)\\
\mathsf{take}(\mathsf{s}(n), x : y) &\to x : \mathsf{take}(n, y) & \mathsf{filter}(\mathsf{s}(n), x : y, m) &\to x : \mathsf{filter}(n, y, m)
\end{aligned}$$

It does not terminate as it can, for instance, exhibit the sequence $\mathsf{from}(0) \to 0 : \mathsf{from}(\mathsf{s}(0)) \to 0 : \mathsf{s}(0) : \mathsf{from}(\mathsf{s}(\mathsf{s}(0))) \to \ldots$. However, if the corresponding code is executed using lazy evaluation then non-termination is not a problem for the program. Since the TRS is orthogonal, confluence does hold. As a consequence, every term has at most one normal form. For instance, a call $\mathsf{take}(\mathsf{s}(\mathsf{s}(0)), \mathsf{primes})$ evaluates to the (unique) normal form $\mathsf{s}(\mathsf{s}(0)) : \mathsf{s}(\mathsf{s}(\mathsf{s}(0))) : \mathsf{nil}$, representing the list consisting of the first two prime numbers.

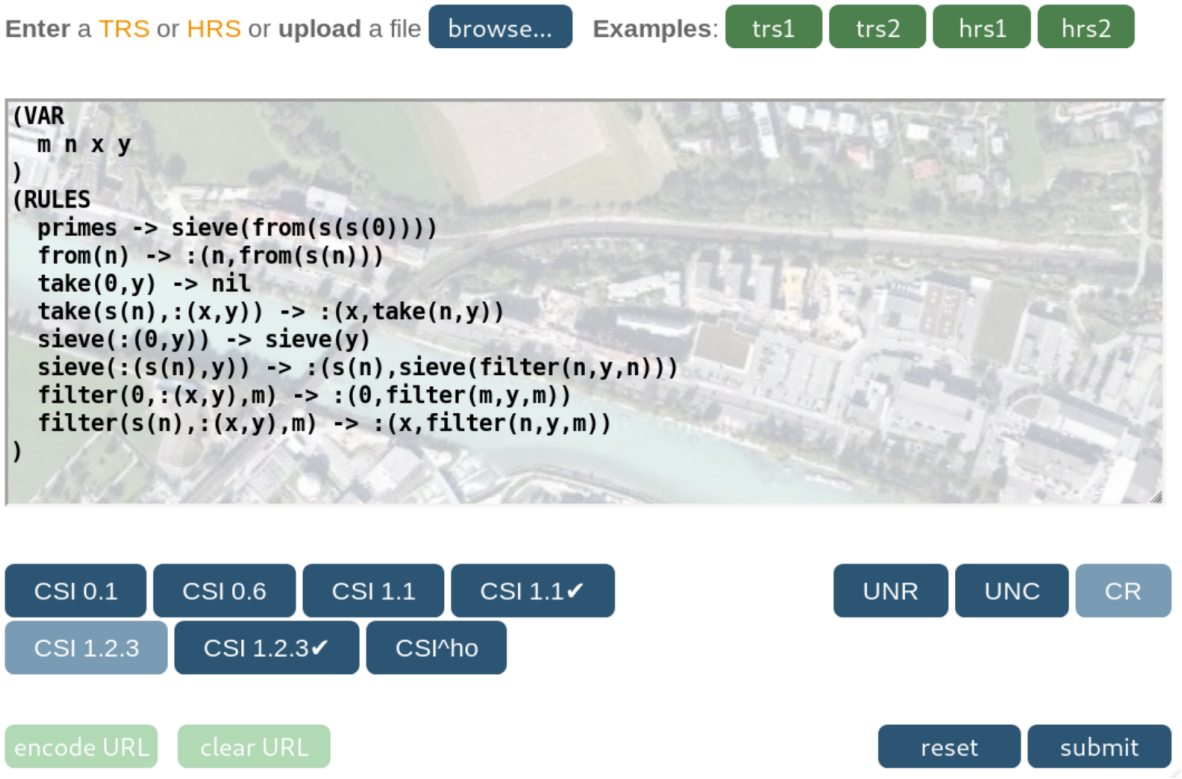


Figure 2: The web interface of CSI.

In a first course on term rewriting the two sufficient conditions described above are typically taught to students and every confluence tool supports these techniques. CSI [17, 38] is developed in Innsbruck. It is built on top of $\mathsf{T_TT_2}$ and available via a web interface and as a stand-alone executable.[2] The web interface is less elaborate than the one of $\mathsf{T_TT_2}$. One reason for this is that the basic sufficient conditions do not have parameters that need to be instantiated. But just like $\mathsf{T_TT_2}$ also CSI outputs all details of the (non)confluence proof to make proof reconstruction for users as easy as possible.

For this paper we added URL encoding. As a consequence, a mouse click suffices to preload the TRS of Example 4.2. The result is shown in Figure 2.

Numerous other techniques have been developed for ensuring confluence and related properties like unique normal forms, some of which are occasionally taught in advanced courses on rewriting. CSI is not the only confluence tool around. All tools for confluence and related properties that participate in the yearly Confluence Competition (CoCo) [15] are available via CoCoWeb,[3] a convenient web interface that provides a single entry point to all tools [9].

5 Completion

Before mentioning any relevant theory, we provide some examples. The first one appeared as a contest in a Dutch popular science magazin [24].

[2] http://cl-informatik.uibk.ac.at/software/csi/
[3] http://cocoweb.uibk.ac.at/

Example 5.1. Genetic engineers in a (hypothetical) research lab want to create cows that produce cola instead of milk. To that end they plan to transform the DNA of the milk gene represented by the sequence TAGCTAGCTAGCT in every fertilized egg into the cola gene, i.e., the sequence CTGACTGACT. The research group already developed techniques to perform the following DNA transformations:

$$\mathsf{TCAT} \leftrightarrow \mathsf{T} \qquad \mathsf{GAG} \leftrightarrow \mathsf{AG} \qquad \mathsf{CTC} \leftrightarrow \mathsf{TC} \qquad \mathsf{AGTA} \leftrightarrow \mathsf{A} \qquad \mathsf{TAT} \leftrightarrow \mathsf{CT}$$

However, recently it has been discovered that the mad cow disease is caused by a retrovirus with the DNA sequence CTGCTACTGACT. Could it happen that accidentally cows with this virus are created?

Example 5.2 (Chameleon Island [6]). A colony of chameleons on a remote island consists of 20 red, 18 blue, and 16 green individuals which continuously walk around. Whenever two chameleons of different color meet, both change to the third color, i.e., they change according to the following rewrite rules:

$$\begin{array}{lll} \mathsf{R}\cdot\mathsf{G} \to \mathsf{B}\cdot\mathsf{B} & \mathsf{B}\cdot\mathsf{R} \to \mathsf{G}\cdot\mathsf{G} & \mathsf{G}\cdot\mathsf{B} \to \mathsf{R}\cdot\mathsf{R} \\ \mathsf{G}\cdot\mathsf{R} \to \mathsf{B}\cdot\mathsf{B} & \mathsf{R}\cdot\mathsf{B} \to \mathsf{G}\cdot\mathsf{G} & \mathsf{B}\cdot\mathsf{G} \to \mathsf{R}\cdot\mathsf{R} \end{array}$$

Some time passes during which no chameleons are born or die nor do any enter or leave the colony. Is it possible that after this period, all 54 chameleons are of the same color?

Both of these examples can be seen as instances of the *validity problem*: Given a set of rewrite rules R and two terms s and t, does $s \leftrightarrow_R^* t$ hold? While this problem is undecidable in general, *Knuth–Bendix completion* [12] is a method to solve some instances.

Definition 5.1. *A* TRS *is* complete *if it is confluent and terminating. A* completion procedure *takes as input an* ES E *and attempts to generate a complete* TRS R *such that* $\leftrightarrow_E^* = \leftrightarrow_R^*$.

If successful, the resulting TRS R can be used to decide the validity problem: by the properties of a completion procedure and because R is complete the following equivalences hold:

$$\leftrightarrow_E^* \;=\; \leftrightarrow_R^* \;=\; \to_R^! \cdot {}_R^!\!\leftarrow$$

Therefore, for any two terms s and t, $s \leftrightarrow_E^* t$ if and only if s and t have the same R-normal form, which is unique since R is confluent. However, since the validity problem is undecidable, completion does not always succeed: it may also fail if some equations cannot be appropriately processed, or run indefinitely.

Applying completion manually often turns out to be a lengthy and tedious process, in particular for students, who lack experience. This observation triggered the development of the Knuth-Bendix Completion Visualizer (KBCV) [31] which is an implementation of a completion procedure providing two different modes: In the automatic mode it attempts to complete the system without further user guidance. But it offers also an interactive mode, where the user can execute a completion procedure step-wise, which is useful for students to get acquainted with completion: All inference rules of the completion inference system taught in the term rewriting course can be applied separately on the present equations and rules to observe their effect, and users can also revert steps that turned out to be disadvantageous. KBCV is available as a Java executable, via a web interface, or as an Android application. Figure 3 shows screenshots from the KBCV Android application run on the gene transformation equations.

Example 5.3. When KBCV is run in automatic mode on the five equations corresponding to possible gene transformations in Example 5.1, it may produce the TRS R consisting of the following six rules:

$$\mathsf{CT} \to \mathsf{T} \qquad \mathsf{TAT} \to \mathsf{T} \qquad \mathsf{AGT} \to \mathsf{AT} \qquad \mathsf{GA} \to \mathsf{A} \qquad \mathsf{ATA} \to \mathsf{A} \qquad \mathsf{TCA} \to \mathsf{TA}$$

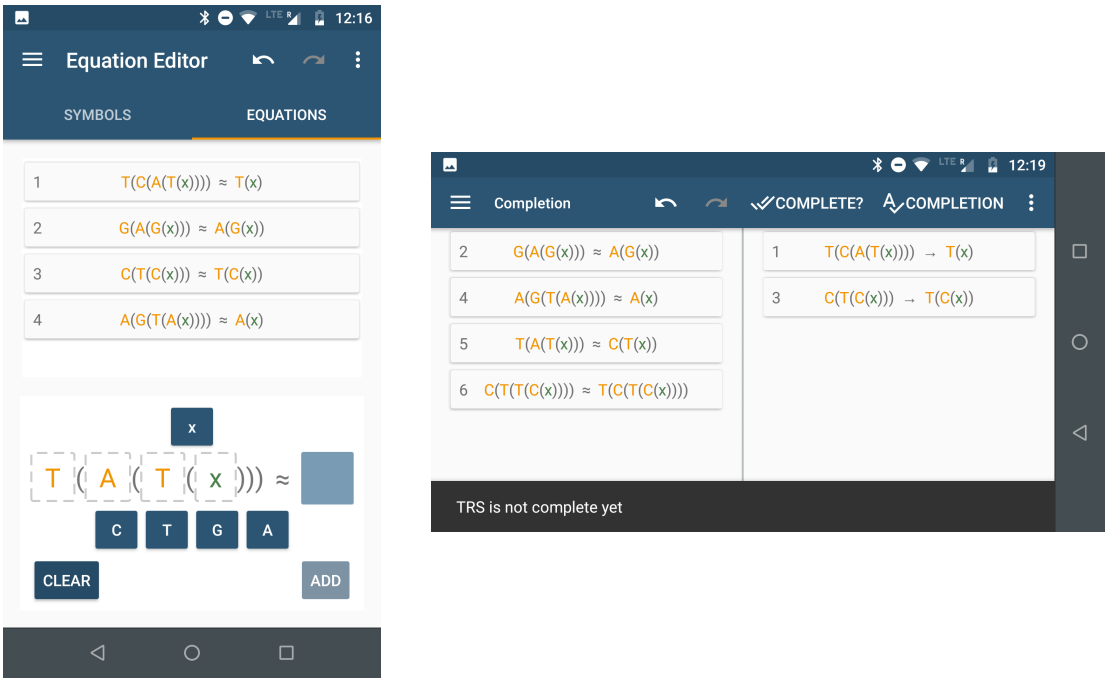


Figure 3: The equation editor and the completion interface of the KBCV Android application.

As R is complete, every two convertible terms have a common normal form. For instance, this is indeed the case for the terms TAGCTAGCTAGCT and CTGACTGACT corresponding to the milk and cola gene, which confirms that the engineers can perform this transformation:

$$\mathsf{TAGCTAGCTAGCT} \to_R^! \mathsf{T} \;{}_R^!\!\leftarrow \mathsf{CTGACTGACT}$$

The milk gene and the mad cow retrovirus, on the other hand, have different normal forms:

$$\mathsf{TAGCTAGCTAGCT} \to_R^! \mathsf{T} \neq \mathsf{TGT} \;{}_R^!\!\leftarrow \mathsf{CTGCTACTGACT}$$

Hence there is no danger that an experiment using the above transformations produces the retrovirus.

The case of the chameleon puzzle in Example 5.2 is more complicated because the six color-changing rules do not suffice to model the problem as a TRS as the animals do not meet in a fixed order. In formal terms, the meeting operator $\cdot$ should be associative and commutative, i.e., satisfy the following equations:

$$(x \cdot y) \cdot z \approx x \cdot (y \cdot z) \qquad\qquad x \cdot y \approx y \cdot x$$

However, any completion procedure will fail when confronted with the second equation since it cannot be oriented into a terminating rewrite rule. Associative and commutative (AC) operators commonly occur in practice, for instance in many algebraic specifications. To deal with such situations, *AC-completion procedures* have been developed which work *modulo* such equations [21]. The tool mkbTT [36] offers both a standard and an AC-completion procedure in an automatic mode, and is available as a binary or via a web interface.[4]

[4] http://cl-informatik.uibk.ac.at/software/mkbtt/

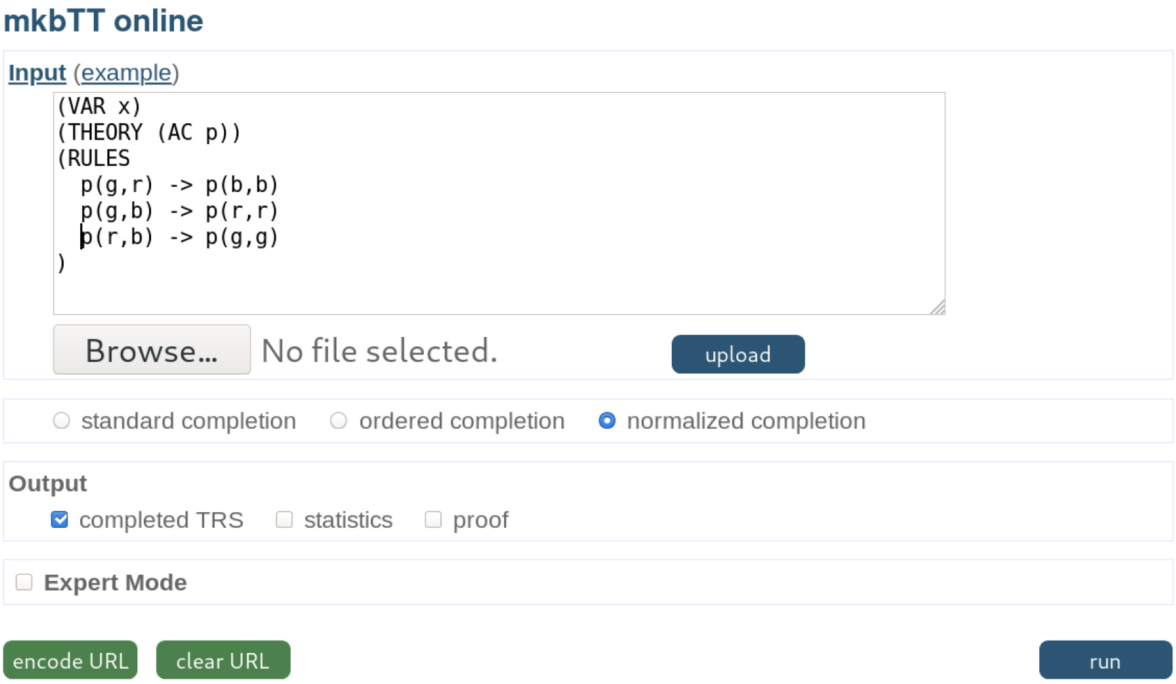


Figure 4: The web interface of mkbTT.

Example 5.4. When mkbTT is run on the following three equations with AC operator $\cdot$:

$$\mathsf{R}\cdot\mathsf{G}\approx\mathsf{B}\cdot\mathsf{B} \qquad \mathsf{B}\cdot\mathsf{R}\approx\mathsf{G}\cdot\mathsf{G} \qquad \mathsf{G}\cdot\mathsf{B}\approx\mathsf{R}\cdot\mathsf{R}$$

it outputs a TRS R that is obtained by reverting one equation and adding one further rule:

$$\mathsf{R}\cdot\mathsf{G}\to\mathsf{B}\cdot\mathsf{B} \qquad \mathsf{G}\cdot\mathsf{G}\to\mathsf{R}\cdot\mathsf{B} \qquad \mathsf{G}\cdot\mathsf{B}\to\mathsf{R}\cdot\mathsf{R} \qquad \mathsf{B}\cdot\mathsf{B}\cdot\mathsf{B}\to\mathsf{R}\cdot\mathsf{R}\cdot\mathsf{R}$$

This TRS is complete modulo AC. We can now rewrite (modulo AC) the terms corresponding to the initial colony and 54 monochromatic chameleons to their respective normal form, where we abbreviate terms of the form $\mathsf{R}\cdot\cdots\cdot\mathsf{R}$ with n occurrences of R by $n\,\mathsf{R}$:

$$20\,\mathsf{R}\cdot 18\,\mathsf{B}\cdot 16\,\mathsf{G}\to^!_{R/\mathrm{AC}} 52\,\mathsf{R}\cdot 2\,\mathsf{B} \qquad 54\,\mathsf{B}\to^!_{R/\mathrm{AC}} 54\,\mathsf{R} \qquad 54\,\mathsf{R}\to^!_{R/\mathrm{AC}} 54\,\mathsf{R} \qquad 54\,\mathsf{G}\to^!_{R/\mathrm{AC}} 54\,\mathsf{R}$$

Since the normal form $52\,\mathsf{R}\cdot 2\,\mathsf{B}$ of the initial colony is different from the normal form $54\,\mathsf{R}$ of 54 monochromatic chameleons these situations are not convertible. Hence it is impossible that all animals turn into the same color. By clicking here the interested reader can test the web interface of mkbTT on this puzzle. The result is displayed in Figure 4.

6 First-Order Theory of Rewriting

An introductory course on term rewriting typically explains basic properties like termination and (local) confluence, together with relationships among these on an abstract level. For instance, local confluence

$$\forall s\,\forall t\,\forall u\,(s\to t\wedge s\to u \implies \exists v\,(t\to^* v\wedge u\to^* v))$$

is a strictly weaker property than confluence

$$\forall s \forall t \forall u \, (s \to^* t \wedge s \to^* u \implies \exists v \, (t \to^* v \wedge u \to^* v)) \tag{1}$$

and the prototype example of a locally confluent rewrite system that is not confluent consists of the four rewrite rules

$$\mathsf{a} \to \mathsf{b} \qquad \mathsf{b} \to \mathsf{a} \qquad \mathsf{a} \to \mathsf{c} \qquad \mathsf{b} \to \mathsf{d}$$

involving only constants. This is an example of an abstract rewrite system (ARS for short), which is a rewrite system over a signature that consists of constants.

Depending on the application area, one can think of a vast number of properties of rewrite systems which are expressible in first-order formulas like (1). Natural questions arising in this context are whether a given property P is satisfiable, valid, or implies a different property P'. Such questions also serve as useful exercises in courses on term rewriting to deepen the understanding of the underlying concepts. Though for many properties of interest such queries are undecidable, certain classes of TRSs turn out to admit decision procedures. Tool support to that end is provided by FORT [25, 26], an implementation of the decision procedure [5] for the first-order theory of rewriting for the class of finite left-linear, right-ground TRSs. This class contains all ARSs. FORT has two different modes.

On the one hand, given a left-linear, right-ground TRS and a formula in the first-order theory of rewriting as input, it decides whether the property expressed by the formula holds for the given TRS. Formulas are first-order logic formulas without function symbols and the predicate symbols include $=$ (equality), $\to$ (one-step rewriting), $\to^*$ (many-step rewriting), $\to^!$ (rewriting to normal form), $\mathrel{+\!\!\!\to}$ (parallel rewriting), and $\leftrightarrow^*$ (conversion). Variables in formulas represent arbitrary ground terms over the signature of the input TRS. Some of the predicate symbols do not increase the expressive power of the language but provide convenient shorthands. For instance, $\to^!$ is such a symbol since $s \to^! t$ if and only if $s \to^* t \wedge \neg \exists u \, (t \to u)$. For expressing termination, FORT supports the unary predicates Fin_R for arbitrary binary regular relations R:

$$\mathsf{Fin}_R(t) \quad \iff \quad (t,u) \in R \text{ for finitely many ground terms } u$$

The formula $\forall t \, (\mathsf{Fin}_{\to^+}(t) \wedge \neg (t \to^+ t))$ states that every term has finitely many reducts and admits no cycle, which is equivalent to termination for finitely-branching TRSs.

On the other hand, FORT provides a synthesis mode in which it tries to synthesize a left-linear, right-ground TRS that satisfies the formula given as input. This is practical only when there exists a small enough witnessing TRS. For instance, when using FORT to synthesize a locally confluent TRS that is not confluent it delivers

$$\mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{c}))) \qquad \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{c}))) \to \mathsf{c}$$

within a few seconds. We can use the decision mode of FORT to confirm the non-confluence of this TRS. Witness generation, a recent extension [26], can be used to find terms in a non-joinable peak:

$$s\colon \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{c})))) \qquad t\colon \mathsf{g}(\mathsf{c}) \qquad u\colon \mathsf{c}$$

Several input parameters allow to guide the search for a suitable TRS. We refer to [25] for further details.

The current version of FORT is written in Java and available as an executable JAR file.[5] The decision procedure implemented in FORT is based on tree automata techniques (ground tree transducers, tree automata operating on encodings of relations on ground terms), which are covered in a graduate course in Innsbruck on selected topics in term rewriting. Since tree automata operate on ground terms, the properties that can be expressed in the first-order theory of rewriting are properties on ground terms. So the earlier formula (1) stands for ground-confluence, which differs from confluence, even for left-linear right-ground TRSs. FORT provides special support to deal with non-ground terms for properties related to confluence. For details we refer to [26].

7 Strategies

In Example 4.2 we have seen an example of a non-terminating confluent TRS. For terms that have a normal form but also admit infinite computations, like $\mathsf{take}(\mathsf{s}(\mathsf{s}(0)),\mathsf{primes})$, it is important to adopt an evaluation strategy that guarantees that the normal form is reached. The study of strategies has a rich history—it goes back to the early days of λ-calculus and combinatory logic—and many deep results have been obtained (see [19]). Students are typically taught the main strategies and their normalization behaviour, without going into the proof details.

Example 7.1. We revisit Example 4.2. If we adopt an eager evaluation strategy like *leftmost-innermost* in which the leftmost of the innermost redexes is selected in each reducible term, we will not reach the normal form of $\mathsf{take}(\mathsf{s}(\mathsf{s}(0)),\mathsf{primes})$, where we use n to denote $\mathsf{s}^n(0)$:

$$\begin{aligned}
\mathsf{take}(2,\underline{\mathsf{primes}}) &\to_R \mathsf{take}(2,\mathsf{sieve}(\underline{\mathsf{from}(2)}))\\
&\to_R \mathsf{take}(2,\mathsf{sieve}(2:\underline{\mathsf{from}(3)}))\\
&\to_R \mathsf{take}(2,\mathsf{sieve}(2:(3:\underline{\mathsf{from}(4)})))\\
&\to_R \cdots
\end{aligned}$$

Adopting the *leftmost-outermost* strategy in which the leftmost of the outermost redexes is selected, the normal form $2:(3:\mathsf{nil})$ is reached:

$$\begin{aligned}
\mathsf{take}(2,\underline{\mathsf{primes}}) &\to_R \mathsf{take}(2,\mathsf{sieve}(\underline{\mathsf{from}(2)}))\\
&\to_R \mathsf{take}(2,\underline{\mathsf{sieve}(2:\mathsf{from}(3))})\\
&\to_R \underline{\mathsf{take}(2,2:\mathsf{sieve}(\mathsf{filter}(1,\mathsf{from}(3),1)))}\\
&\to_R 2:\mathsf{take}(1,\mathsf{sieve}(\mathsf{filter}(1,\underline{\mathsf{from}(3)},1)))\\
&\to_R 2:\mathsf{take}(1,\mathsf{sieve}(\underline{\mathsf{filter}(1,3:\mathsf{from}(4),1)}))\\
&\to_R 2:\mathsf{take}(1,\underline{\mathsf{sieve}(3:\mathsf{filter}(0,\mathsf{from}(4),1))})\\
&\to_R 2:\underline{\mathsf{take}(1,3:\mathsf{sieve}(\mathsf{filter}(2,\mathsf{filter}(0,\mathsf{from}(4),1),2)))}\\
&\to_R 2:(3:\underline{\mathsf{take}(0,\mathsf{sieve}(\mathsf{filter}(2,\mathsf{filter}(0,\mathsf{from}(4),1),2)))})\\
&\to_R 2:(3:\mathsf{nil})
\end{aligned}$$

Other evaluation strategies like the maximal strategy (previously known as full-substitution or Gross–Knuth reduction) are more difficult to apply correctly and this is where the tool AutoStrat,[6] comes in

[5] http://cl-informatik.uibk.ac.at/software/FORT/

[6] http://cl-informatik.uibk.ac.at/software/AutoStrat/

handy. This tool was developed in a bachelor project [16] and also has support for *strategy annotations*. These were introduced in [22, 23] and provide the user with more control over the evaluation strategy. A key notion here is *in-time*. We provide an example.

Example 7.2. Consider the TRS consiting of the rewrite rules

$$\alpha\colon\ x \wedge \mathsf{T} \to x \qquad \gamma\colon\ \mathsf{T} \vee x \to \mathsf{T} \qquad \varepsilon\colon\ \infty \to \infty \qquad \beta\colon\ x \wedge \mathsf{F} \to \mathsf{F} \qquad \delta\colon\ \mathsf{F} \vee x \to x$$

The Greek letters are used to name to the individual rules. A strategy annotation specifies for every function symbol the order in which arguments and potentially matching rewrite rules are applied. Consider the annotation A with

$$A(\wedge) = [2, \alpha, \beta, 1] \qquad A(\vee) = [1, \gamma, \delta, 2] \qquad A(\infty) = [\varepsilon] \qquad A(\mathsf{T}) = A(\mathsf{F}) = [\,]$$

Suppose we want to evaluate the term $t = (\infty \wedge \mathsf{F}) \vee (\mathsf{T} \vee \infty)$.

- The strategy annotation $[1, \gamma, \delta, 2]$ of its root symbol $\vee$ tells us that we first look for a redex in the first argument $\infty \wedge \mathsf{F}$ of t.
- The strategy annotation $[2, \alpha, \beta, 1]$ for $\wedge$ indicates to look for a redex in the second argument F of $\infty \wedge \mathsf{F}$. Since $A(\mathsf{F}) = [\,]$, this will fail. So we discard the first element of $[2, \alpha, \beta, 1]$ and try whether rule α applies. This also fails. Next up is rule β. Since β is applicable, we have found our redex and hence $\infty \wedge \mathsf{F}$ rewrites to F.

So t rewrites to $\mathsf{F} \vee (\mathsf{T} \vee \infty)$.

A strategy annotation defines an evaluation strategy provided the annotation is *full*, which means that no possibilities are omitted. The annotation A in the above example is full. If we change $A(\vee) = [1, \gamma, \delta, 2]$ to $A(\vee) = [\gamma, \delta, 2]$ we lose fullness and, as a consequence, the induced strategy may get stuck on terms which are not yet in normal form. Indeed, the term $(\mathsf{T} \wedge \mathsf{T}) \vee \mathsf{F}$ cannot be reduced since the only redex is in the first argument of $\vee$, which is excluded from the annotation for $\vee$.

After an annotation-guided rewrite step is performed, the process starts all over on the resulting term. This typically results in duplicated efforts to determine the next redex. A function *normalize* can be defined that continues from the position of the last step. If the annotation is not only full but also *in-time*, meaning that argument positions are listed before rules that need them, this function is guaranteed to compute a normal form whenever the annotation-guided strategy that computes the steps separately is normalizing. We refer to [22] for formal definitions.

8 Complexity

While termination is a desirable property, it does not always suffice. For programs in performance-critical contexts, the computational complexity is crucial, for simplification processes fast rewriting to normal form is desired, and frequently the maximal number of rewrite steps needs to be known for theoretical considerations. In term rewriting such considerations gave rise to the research area of complexity. In this section we summarize relevant notions and some results.

A function symbol f is *defined* in a TRS R if R contains a rule $f(\ell_1, \ldots, \ell_n) \to r$. Symbols which are not defined are *constructor* symbols. A term t is *basic* with respect to a TRS R if $t = f(t_1, \ldots, t_n)$ such that f is defined but none of the arguments $t_1, \ldots, t_n$ contain any defined symbols. Complexity analysis in rewriting focuses on the notions defined below.

Definition 8.1. *For a terminating* TRS R,

- *the* derivation height *of a term t is given by* $\mathsf{dh}_R(t) = \max\{\, n \mid t \to_R^n u \text{ for some term } u \,\}$,
- *the* derivational complexity *of R is defined as* $\mathsf{dc}_R(n) = \max\{\, \mathsf{dh}_R(t) \mid |t| = n \,\}$, *and*
- *the* runtime complexity *of R is defined as* $\mathsf{rc}_R(n) = \max\{\, \mathsf{dh}_R(t) \mid t \text{ is basic and } |t| = n \,\}$.

Here $|t|$ denotes the size of the term t.

While the derivation height of a term t asks for the maximal number of rewrite steps that can be performed from t before reaching a normal form, the derivational complexity of the rewrite system relates the derivation height to the size of the starting term. The runtime complexity restricts this notion to basic terms, which correspond to potential input of programs. The different concepts are illustrated by the following example.

Example 8.1. Consider the following TRS R representing a functional program to shuffle a list:

$$\begin{array}{lll} \mathsf{nil} \mathbin{@} ys \to ys & \mathsf{rev}(\mathsf{nil}) \to \mathsf{nil} & \mathsf{shuffle}(\mathsf{nil}) \to \mathsf{nil} \\ (x : xs) \mathbin{@} ys \to x : (xs \mathbin{@} ys) & \mathsf{rev}(x : xs) \to \mathsf{rev}(xs) \mathbin{@} (x : \mathsf{nil}) & \mathsf{shuffle}(x : xs) \to x : \mathsf{shuffle}(\mathsf{rev}(xs)) \end{array}$$

For instance, for the term $t = \mathsf{rev}([1,2])$ we have $\mathsf{dh}_R(t) = 6$ because the following (unique) rewrite sequence to its normal form has six steps:

$$\begin{aligned} \mathsf{rev}([1,2]) &\to_R (\mathsf{rev}([2]) \mathbin{@} [1] \\ &\to_R (\mathsf{rev}(\mathsf{nil}) \mathbin{@} [2]) \mathbin{@} [1] \\ &\to_R (\mathsf{nil} \mathbin{@} [2]) \mathbin{@} [1] \\ &\to_R [2] \mathbin{@} [1] \\ &\to_R 2 : (\mathsf{nil} \mathbin{@} [1]) \\ &\to_R [2,1] \end{aligned}$$

Consider a list xs of length n. Analysis of the TRS R shows that the number of steps in the (unique) rewrite sequence to normal form is

- linear in n for a term of the form $xs \mathbin{@} ys$,
- quadratic for $\mathsf{rev}(xs)$, and
- cubic for $\mathsf{shuffle}(xs)$.

A term of the form $\mathsf{shuffle}^n(xs)$ even needs $\mathcal{O}(n^4)$ steps. Terms of this shape turn out to witness the worst-case as far as derivation length in R is concerned, hence $\mathsf{dc}_R \in \mathcal{O}(n^4)$.

Note that the term $\mathsf{shuffle}(xs)$ is basic, but $\mathsf{shuffle}^n(xs)$ is not. Indeed the latter does not correspond to a run of our shuffle program, where we expect to execute the $\mathsf{shuffle}$ function on some input list. On the other hand, $\mathsf{shuffle}(xs)$ is basic and a witness for the cubic runtime complexity of this program, i.e., we have $\mathsf{rc}_R \in \mathcal{O}(n^3)$.

Example 8.2. We revisit Example 1.1 from the introduction. In the case of the TRS R_1 the number of beans decreases by one with every move. Hence $\mathsf{dc}_{R_1} \in \mathcal{O}(n)$, i.e., the number of rewrite steps is linear in the size of the initial configuration. The TRS R_2 on the other hand admits very long derivations. The rewrite sequence

$$\bullet^n(\circ(x)) \to_{R_2} \bullet^{n-1}(\circ(\circ(\circ(\bullet(x))))) \to_{R_2} \cdots \to_{R_2} \circ^{3^n}(\bullet^n(x))$$

shows that dc_{R_2} is exponential. In fact it is known that a TRS which can be proven terminating by a polynomial interpretation has double exponential derivational complexity in the worst case [10]. If the interpretation is linear as in Example 3.1, the bound is still single exponential.

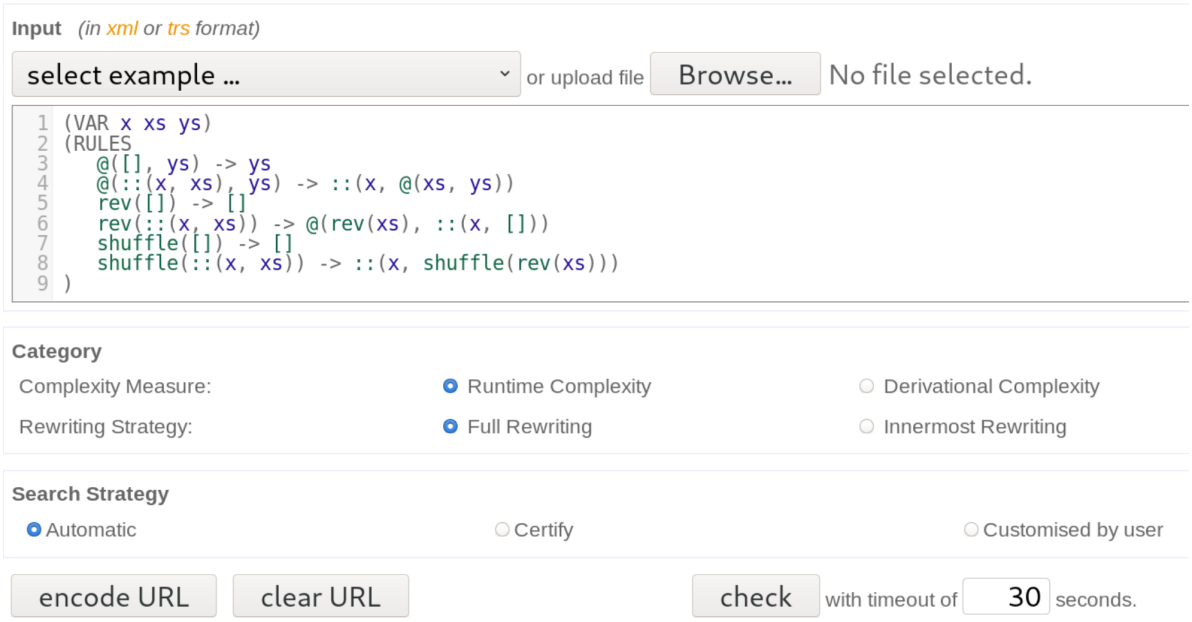


Figure 5: The web interface of TCT.

The Tyrolean Complexity Tool TCT is a fully automatic tool for complexity analysis [3]. For instance, clicking here loads Example 8.1 into the web interface of TCT (with the result shown in Figure 5), and running the tool on this input establishes cubic runtime complexity as remarked above (quartic derivational complexity holds due to the technique from [7]). TCT can not only derive upper bounds on runtime and derivational complexity of TRSs but also provides resource analysis for Java bytecode and functional programs, in the latter case also for higher-order functions, as illustrated by the following example.

Example 8.3. The following Haskell program reverses a list using the higher-order function `fold_left`:

```
let rec fold_left f acc = function
   []    -> acc
 | x::xs -> fold_left f (f acc x) xs ;;

let rev l = fold_left (fun xs x -> x :: xs) [] l ;;
```

TCT transforms such programs into *higher-order* rewrite systems [32], a paradigm whose details are beyond the scope of this paper. Here we contend ourselves by noting that TCT can conclude linear runtime complexity of this implementation of the `rev` function, as one would expect.

9 Conclusion

This paper presented automatic tools to analyze term rewrite systems, developed by the Computational Logic Group at the University of Innsbruck. These tools are not only important in research but also valuable for teaching. In an annual course on term rewriting, as well as in several editions of the International Summer School on Rewriting,[7] they proved highly useful for students as well as teachers to solve and

[7] `http://cbr.uibk.ac.at/ifip-wg1.6/summerschool.html`

prepare homework exercises and exam questions. The tools were mostly developed by (former) graduate students but also benefitted from student feedback after the use in courses.

Related Work. Several other tools support the same TRS analysis tasks as the tools described in this paper. In the following paragraphs, we mention recent tools which are still maintained, and focus on their usability, in particular via web interfaces since these render them more accessible to students. We also restrict ourselves to standard TRSs, for special types of rewrite systems more tools are available.

In the standard category of the Termination Competition 2019 six tools participated; ordered by the number of problems solved these are AProVE [8], NaTT [37], T$_{\text{T}}$T$_2$, mu-term [1], Wanda [13], and NTI [20]. Only AProVE and mu-term have web interfaces. The latter allows the user to control (the shape of) polynomial interpretations, and whether to use RPO and dependency pairs (but neither LPO, KBO, not matrix interpretations are supported). In the AProVE web interface the user cannot control the strategy applied to prove termination (or complexity, which is also supported by AProVE). However, the Java standalone tool offers many options for control.

In the standard category of the Confluence Competition 2019, besides CSI, ACP [2] and CoLL-Saigawa [29] participated. Neither of these has a web interface, but CoCoWeb [9] makes them accessible. Recent completion tools besides mkbTT and KBCV are maxcomp [11], and mædmax [35]. Only the latter has a web interface, but it offers few options for control.

To the best of our knowledge, there are no other tools available which provide similar functionalities as FORT or AutoStrat. In proving runtime and derivational complexity of TRSs, the only recent competitor of T$_{\text{C}}$T is AProVE, already described above.

Outlook. Although the presented tools cover by now all approaches explored in the basic term rewriting course, the software keeps being extended and updated to support new techniques emerging from research. Besides their power as analysis tools, also their user-friendliness and suitability for teaching can still be improved. Among possible extensions are web interfaces for FORT and AutoStrat, mobile-friendly interfaces for or mobile applications of other tools besides KBCV, and more control over the parameters of proof search, for instance in CSI and T$_{\text{C}}$T. A single web interface to access all of the tools is another useful extension. This could include a "meta-analyzer" option which uses the current tools to analyze multiple properties of a given TRS at once.

Finally, we comment on the reliability of the presented tools. Automated reasoning implementations constitute complex pieces of software due to sophisticated deduction techniques, a high degree of optmization, and elaborate heuristics. Hence, implementation errors are to be expected. In order to deal with this problem, trusted proof checkers for rewrite tools have been implemented in the course of the last decade. To that end, a vast amount of rewriting theory has been formalized and proved correct in Isabelle/HOL in the Isabelle Formalization of Rewriting (IsaFoR) project [30, 33].[8] From this formalization the proof checker CeTA is generated automatically, which can validate certificates for the respective properties (like termination, confluence of completeness) output by T$_{\text{T}}$T$_2$, CSI, KBCV, mkbTT, FORT, or T$_{\text{C}}$T. Even though many techniques are already supported by IsaFoR/CeTA, some of the methods implemented in tools remain to be added.

[8] http://cl-informatik.uibk.ac.at/isafor/

References

[1] Beatriz Alarcón, Raúl Gutiérrez, Salvador Lucas & Rafael Navarro-Marset (2011): *Proving Termination Properties with MU-TERM*. In: *Proc. 13thAlgebraic Methodology and Software Technology*, *Lecture Notes in Computer Science* 6486, pp. 201–208, doi:10.1007/978-3-642-17796-5_12.

[2] Takahito Aoto, Junichi Yoshida & Yoshihito Toyama (2009): *Proving Confluence of Term Rewriting Systems Automatically*. In: *Proc. 20th International Conference on Rewriting Techniques and Applications*, *Lecture Notes in Computer Science* 5595, pp. 93–102, doi:10.1007/978-3-642-02348-4_7.

[3] Martin Avanzini, Georg Moser & Michael Schaper (2016): *TcT: Tyrolean Complexity Tool*. In: *Proc. 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes in Computer Science* 9636, Springer, pp. 407–423, doi:10.1007/978-3-662-49674-9_24.

[4] Franz Baader & Tobias Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, doi:10.1017/CBO9781139172752.

[5] Max Dauchet & Sophie Tison (1990): *The Theory of Ground Rewrite Systems is Decidable*. In: *Proc. 5th IEEE Symposium on Logic in Computer Science*, pp. 242–248, doi:10.1109/LICS.1990.113750.

[6] Nachum Dershowitz & David A. Plaisted (2001): *Chapter 9 – Rewriting*. In: *Handbook of Automated Reasoning*, North-Holland, pp. 535–610, doi:10.1016/B978-044450813-3/50011-4.

[7] Carsten Fuhs (2019): *Transforming Derivational Complexity of Term Rewriting to Runtime Complexity*. In: *Proc. 12th International Workshop on Frontiers of Combining Systems*, *Lecture Notes in Computer Science* 11715, Springer, pp. 348–364, doi:10.1007/978-3-030-29007-8_20.

[8] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Strder, Stephanie Swiderski & Ren Thiemann (2017): *Analyzing Program Termination and Complexity Automatically with AProVE*. *Journal of Automated Reasoning* 58(1), pp. 3–31, doi:10.1007/s10817-016-9388-y.

[9] Nao Hirokawa, Julian Nagele & Aart Middeldorp (2018): *Cops and CoCoWeb – Infrastructure for Confluence Tools*. In: *Proc. 9th International Joint Conference on Automated Reasoning*, *Lecture Notes in Artificial Intelligence* 10900, Springer, pp. 346–353, doi:10.1007/978-3-319-94205-6_23.

[10] Dieter Hofbauer & Clemens Lautemann (1989): *Termination Proofs and the Length of Derivations*. In: *Proc. 3rd International Conference on Rewriting Techniques and Applications*, *Lecture Notes in Computer Science* 355, Springer, pp. 167–177, doi:10.1007/3-540-51081-8_107.

[11] Dominik Klein & Nao Hirokawa (2011): *Maximal Completion*. In: *Proc. 22nd International Conference on Rewriting Techniques and Applications*, *Leibniz International Proceedings in Informatics* 10, pp. 71–80, doi:10.4230/LIPIcs.RTA.2011.71.

[12] Donald E. Knuth & Peter B. Bendix (1970): *Simple Word Problems in Universal Algebras*. In J. Leech, editor: *Computational Problems in Abstract Algebra*, Pergamon Press, pp. 263–297, doi:10.1016/B978-0-08-012975-4.50028-X.

[13] Cynthia Kop (2019): *A short overview of Wanda*. In: *Joint Proceedings of the 10th Workshop on Higher-Order Rewriting and the 8th International Workshop on Confluence*, pp. 21–25. Available from `http://cl-informatik.uibk.ac.at/iwc/hor-iwc2019.pdf`.

[14] Martin Korp, Christian Sternagel, Harald Zankl & Aart Middeldorp (2009): *Tyrolean Termination Tool 2*. In: *Proc. 20th International Conference on Rewriting Techniques and Applications*, *Lecture Notes in Computer Science* 5595, Springer, pp. 295–304, doi:10.1007/978-3-642-02348-4_21.

[15] Aart Middeldorp, Julian Nagele & Kiraku Shintani (2019): *Confluence Competition 2019*. In: *Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes in Computer Science* 11429, Springer, pp. 25–40, doi:10.1007/978-3-030-17502-3_2.

[16] Fabian Mitterwallner (2018): *Automating Rewrite Strategies*. bachelor thesis, University of Innsbruck.

[17] Julian Nagele, Bertram Felgenhauer & Aart Middeldorp (2017): *CSI: New Evidence – A Progress Report*. In: *Proc. 26th International Conference on Automated Deduction*, *Lecture Notes in Artificial Intelligence* 10395, Springer, pp. 385–397, doi:10.1007/978-3-319-63046-5_24.

[18] Max H. A. Newman (1942): *On Theories with a Combinatorial Definition of Equivalence*. *Annals of Mathematics* 43(2), pp. 223–243, doi:10.2307/1968867.

[19] Vincent van Oostrom & Roel de Vrijer (2003): *Strategies*. In Terese, editor: *Term Rewriting Systems*, *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press. chapter 9, pp. 475–547.

[20] Étienne Payet & Frdric Mesnard (2006): *Nontermination Inference of Logic Programs*. *ACM Transactions on Programming Languages and Systems* 28(2), pp. 256–289, doi:10.1145/1119479.1119481.

[21] Gerald E. Peterson & Mark E. Stickel (1981): *Complete Sets of Reductions for Some Equational Theories*. *Journal of the ACM* 28(2), pp. 233–264, doi:10.1145/322248.322251.

[22] Jaco van de Pol (2001): *Just-in-time: On Strategy Annotations*. In: *Proc. 1st International Workshop on Reduction Strategies in Rewriting and Programming*, *Electronic Notes in Theoretical Computer Science* 57, pp. 41–63, doi:10.1016/S1571-0661(04)00267-1.

[23] Jaco van de Pol (2002): *JITty: A Rewriter with Strategy Annotations*. In: *Proc. 13th International Conference on Rewriting Techniques and Applications*, *Lecture Notes in Computer Science* 2378, Springer, pp. 367–370, doi:10.1007/3-540-45610-4_26.

[24] Prijsvraag (2015): *Het Cola-gen*. *Natuur, Wetenschap & Techniek* 73(1), p. 65.

[25] Franziska Rapp & Aart Middeldorp (2016): *Automating the First-Order Theory of Left-Linear Right-Ground Term Rewrite Systems*. In: *Proc. 1st International Conference on Formal Structures for Computation and Deduction*, *Leibniz International Proceedings in Informatics* 52, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 36:1–36:12, doi:10.4230/LIPIcs.FSCD.2016.36.

[26] Franziska Rapp & Aart Middeldorp (2018): *FORT 2.0*. In: *Proc. 9th International Joint Conference on Automated Reasoning*, *Lecture Notes in Artificial Intelligence* 10900, Springer, pp. 81–88, doi:10.1007/978-3-319-94205-6_6.

[27] Barry K. Rosen (1973): *Tree-Manipulating Systems and Church-Rosser Theorems*. *Journal of the ACM* 20(1), pp. 160–187, doi:10.1145/321738.321750.

[28] Jonas Schöpf & Christian Sternagel (2018): *TTT2 with Termination Templates for Teaching*. *The Computing Research Repository* abs/1806.05040. Available at `http://arxiv.org/abs/1806.05040`.

[29] Kiraku Shintani & Nao Hirokawa (2019): *CoLL-Saigawa 1.3: A Joint Confluence Tool*. In: *Joint Proceedings of the 10th Workshop on Higher-Order Rewriting and the 8th International Workshop on Confluence*, p. 57. Available from `http://cl-informatik.uibk.ac.at/iwc/hor-iwc2019.pdf`.

[30] Christian Sternagel & René Thiemann (2014): *The Certification Problem Format*. In: *Proc. 11th Workshop on User Interfaces for Theorem Provers (UITP)*, *Electronic Proceedings in Computer Science* 167, pp. 61–72, doi:10.4204/EPTCS.167.8.

[31] Thomas Sternagel & Harald Zankl (2012): *KBCV – Knuth–Bendix Completion Visualizer*. In: *Proc. 6th International Joint Conference on Automated Reasoning*, *Lecture Notes in Artificial Intelligence* 7364, pp. 530–536, doi:10.1007/978-3-642-31365-3_41.

[32] Terese, editor (2003): *Term Rewriting Systems*. *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press.

[33] René Thiemann & Christian Sternagel (2009): *Certification of Termination Proofs using CeTA*. In: *Proc. 22th International Conference on Theorem Proving in Higher Order Logics*, *Lecture Notes in Computer Science* 5674, Springer, pp. 452–468, doi:10.1007/978-3-642-03359-9_31.

[34] Hélène Touzet (1998): *Encoding the Hydra Battle as a Rewrite System*. In: *Proc. 23rd Mathematical Foundations of Computer Science*, *Lecture Notes in Computer Science* 1450, Springer, pp. 267–276, doi:10.1007/BFb0055776.

[35] Sarah Winkler & Georg Moser (2018): *MaedMax: A Maximal Ordered Completion Tool*. In: *Proc. 9th International Joint Conference on Automated Reasoning*, *Lecture Notes in Computer Science* 10900, pp. 472–480, doi:10.1007/978-3-319-94205-6_31.

[36] Sarah Winkler, Haruhiko Sato, Aart Middeldorp & Masahito Kurihara (2013): *Multi-Completion with Termination Tools*. *Journal of Automated Reasoning* 50(3), pp. 317–354, doi:10.1007/s10817-012-9249-2.

[37] Akihisa Yamada, Keiichirou Kusakari & Toshiki Sakabe (2014): *Nagoya Termination Tool*. In: *Proc. 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications*, *Lecture Notes in Computer Science* 8560, pp. 466–475, doi:10.1007/978-3-319-08918-8_32.

[38] Harald Zankl, Bertram Felgenhauer & Aart Middeldorp (2011): *CSI – A Confluence Tool*. In: *Proc. 22nd International Conference on Automated Deduction*, *Lecture Notes in Artificial Intelligence* 6803, Springer, pp. 499–505, doi:10.1007/978-3-642-22438-6_38.

[39] Harald Zankl, Sarah Winkler & Aart Middeldorp (2015): *Beyond Polynomials and Peano Arithmetic – Automation of Elementary and Ordinal Interpretations*. *Journal of Symbolic Computation* 69, pp. 129–158, doi:10.1016/j.jsc.2014.09.033.