CDIPROVER3: A TOOL FOR PROVING DERIVATIONAL COMPLEXITIES OF TERM REWRITING SYSTEMS

Andreas Schnabl
University of Innsbruck
andreas.schnabl@uibk.ac.at

Abstract. This paper describes `cdiprover3` a tool for proving termination of term rewrite systems by polynomial interpretations and context dependent interpretations. The methods used by `cdiprover3` induce small bounds on the derivational complexity of the considered system. We explain the tool in detail, and give an overview of the employed proof methods.

1. Introduction

Term rewriting is a Turing complete model of computation, which is conceptually closely related to declarative and (first-order) functional programming. One of its most studied properties, termination, is also a central problem in computer science. This property is undecidable in general, but many partial decision methods have been developed in the last decades. Beyond showing termination of a given rewriting system, some of these methods can also give bounds on different measures of its complexity. As suggested in (Hofbauer & Lautemann 1989), a natural way of measuring the complexity of a term rewrite system is to analyze its *derivational complexity*. The derivational complexity is a function which relates the size of a term and the maximal number of rewrite steps that can be executed starting from any term of that size in the given rewrite system . We are particularly interested in small, i.e. polynomial upper bounds on this function. In contrast to our approach of measuring derivational complexity, the *constructor discipline* is mentioned in (Lescanne 1995). In this field, we look at the complexity of the function that is encoded by a constructor system. It is either measured by the number of rewrite steps needed to bring the term into normal form (Bonfante, et al. 2001, Avanzini & Moser 2008), or by counting the number of steps needed by some evaluation mechanism different from standard term rewriting (Marion 2003, Bonfante, et al. 2007).

In this paper, we describe `cdiprover3`, a tool which uses polynomial and context-dependent interpretations in order to prove termination and complexity bounds of term rewrite systems. The tool, its predecessors, and full experimental data are available at

`http://cl-informatik.uibk.ac.at/~aschnabl/experiments/cdi/` .

s Polynomial interpretations, introduced in (Lankford 1979), are a standard *direct* termination proof method. Besides showing termination of rewrite systems, they also provide

an easy way to extract upper bounds on the derivational complexity (Hofbauer & Lautemann 1989). However, as noticed in (Hofbauer 2001), this often heavily overestimates the derivational complexity. Context dependent interpretations, also introduced in (Hofbauer 2001), are an effort to improve these upper bounds.

The remainder of this paper is organised as follows: Section 2. outlines the basics of term rewriting needed to state all relevant results. In Section 3., we briefly describe polynomial and context dependent interpretations, which are used by `cdiprover3`. Section 4. describes the implementation of `cdiprover3`, and mentions some experimental results. In Section 5., we explain the input and output of `cdiprover3` in detail. Last, in Section 6., we state conclusions and potential future work.

2. Term Rewriting

In this section, we review some basics of term rewriting. We only cover the concepts which are relevant to this paper. A general introduction to term rewriting can be found in (Baader & Nipkow 1998, TeReSe 2003), for instance.

A *term rewrite system* (TRS) $\mathcal{R}$ consists of a *signature* $\mathcal{F}$, a countably infinite set of *variables* $\mathcal{V}$ disjoint from $\mathcal{F}$, and a finite set of *rewrite rules* $l \to r$, where l and r are terms such that $l \notin \mathcal{V}$ and all variables which occur in r also occur in l. The signature $\mathcal{F}$ defines a set of function symbols, and assigns to each function symbol f its arity. We assume that every signature contains at least one function symbol of arity 0. The set of terms built from $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. The set of terms $\mathcal{T}(\mathcal{F})$ without any variables is called the set of *ground terms* over $\mathcal{F}$. A function symbol is *defined* if it occurs at the root of a left hand side of a rewrite rule. All non-defined function symbols are called *constructors*. A *constructor based term* is a term containing exactly one defined function symbol, which appears at the root of that term. We call the total number of function symbol and variable occurrences in a term t its *size*, denoted by $|t|$. A *substitution* is a mapping $\sigma : \mathrm{Dom}(\sigma) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$, where $\mathrm{Dom}(\sigma)$ is a finite subset of $\mathcal{V}$. The result of replacing all occurrences of variables $x \in \mathrm{Dom}(\sigma)$ in a term t by $\sigma(x)$ is denoted by $t\sigma$. A *context* is a term $C[\Box]$ containing a single occurrence of a fresh function symbol $\Box$ of arity 0. If we replace $\Box$ with a term t, we denote the resulting term by $C[t]$. Given a TRS $\mathcal{R}$ and two terms s, t, we say that s *rewrites* to t ($s \to_{\mathcal{R}} t$) if there exist a context C, a substitution σ and a rewrite rule $l \to r$ in $\mathcal{R}$ such that $s = C[l\sigma]$ and $t = C[r\sigma]$. The transitive closure of this relation is $\to^+_{\mathcal{R}}$. The reflexive and transitive closure is $\to^*_{\mathcal{R}}$. We write $\to^n_{\mathcal{R}}$ to express n-fold composition of $\to_{\mathcal{R}}$. A TRS $\mathcal{R}$ is *terminating* if there exists no infinite chain of terms $t_0, t_1, \ldots$ such that $t_i \to_{\mathcal{R}} t_{i+1}$ for each $i \in \mathbb{N}$. For a terminating TRS $\mathcal{R}$, the *derivation length* of a ground term t is defined as $\mathsf{dl}_{\mathcal{R}}(t) = \max\{n \mid \exists s : t \to^n_{\mathcal{R}} s\}$. The *derivational complexity* is the function $\mathsf{dc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N}$ which maps n to $\max\{\mathsf{dl}_{\mathcal{R}}(t) \mid |t| = n\}$.

3. Used Termination Proof Methods

3.1. Polynomial Interpretations

An $\mathcal{F}$-*algebra* $\mathcal{A}$ for some signature $\mathcal{F}$ consists of a *carrier* A and *interpretation functions* $\{f_{\mathcal{A}} : A^n \to A \mid f \in \mathcal{F}, n = \mathsf{arity}(f)\}$. Given an *assignment* $\alpha : \mathcal{V} \to A$, we denote the

evaluation of a term t into $\mathcal{A}$ by $[\alpha]_{\mathcal{A}}(t)$. It is defined inductively as follows:

$$\begin{aligned} [\alpha]_{\mathcal{A}}(x) &= \alpha(x) && \text{for } x \in \mathcal{V} \\ [\alpha]_{\mathcal{A}}(f(t_1, \ldots, t_n)) &= f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n)) && \text{for } f \in \mathcal{F} \end{aligned}$$

A *well-founded monotone $\mathcal{F}$-algebra* is a pair $(\mathcal{A}, >)$ where $\mathcal{A}$ is an $\mathcal{F}$-algebra and $>$ is a well-founded proper order such that for every function symbol $f \in \mathcal{F}$, $f_{\mathcal{A}}$ is monotone with respect to $>$. It is *compatible* with a TRS $\mathcal{R}$ if for every rewrite rule $l \to r$ in $\mathcal{R}$ and every assignment α, $[\alpha]_{\mathcal{A}}(l) > [\alpha]_{\mathcal{A}}(r)$ holds. It is a well-known fact that a TRS $\mathcal{R}$ is terminating if and only if there exists a well-founded monotone algebra that is compatible with $\mathcal{R}$. A *polynomial interpretation* (Lankford 1979) is an interpretation into a well-founded monotone algebra $(\mathcal{A}, >)$ such that $A \subseteq \mathbb{N}$, $>$ is the standard order on the natural numbers, and $f_{\mathcal{A}}$ is a polynomial for every function symbol f. If a polynomial interpretation is compatible with a TRS $\mathcal{R}$, then we clearly have $\mathsf{dl}_{\mathcal{R}}(t) \leqslant [\alpha]_{\mathcal{A}}(t)$ for all terms t.

Example 1. Consider the TRS $\mathcal{R}$ with the following rewrite rules over the signature containing the function symbols 0 (arity 0), s (arity 1), + and - (arity 2). The system is example `SK90/2.11.trs` in the termination problems database[1] (TPDB), which is the standard benchmark for termination provers:

$$\begin{array}{lll} +(0, y) \to y & -(0, y) \to 0 & -(\mathsf{s}(x), \mathsf{s}(y)) \to -(x, y) \\ +(\mathsf{s}(x), y) \to \mathsf{s}(+(x, y)) & -(x, 0) \to x & \end{array}$$

The following interpretation functions build a compatible polynomial interpretation $\mathcal{A}$ over the carrier $\mathbb{N}$:

$$+_{\mathcal{A}}(x, y) = 2x + y \qquad -_{\mathcal{A}}(x, y) = 3x + 3y \qquad \mathsf{s}_{\mathcal{A}}(x) = x + 2 \qquad 0_{\mathcal{A}} = 1$$

A *strongly linear interpretation* is a polynomial interpretation such that every interpretation function $f_{\mathcal{A}}$ has the form $f_{\mathcal{A}}(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i + c$, $c \in \mathbb{N}$. A surprisingly simple property is that compatibility with a strongly linear interpretation induces a linear upper bound on the derivational complexity (Schnabl 2007).

A *linear polynomial interpretation* is a polynomial interpretation where each interpretation function $f_{\mathcal{A}}$ has the shape $f_{\mathcal{A}}(x_1, \ldots, x_n) = \sum_{i=1}^{n} a_i x_i + c$, $a_i \in \mathbb{N}$, $c \in \mathbb{N}$. For instance, the interpretation given in Example 1 is a linear polynomial interpretation. Because of their simplicity, this class of polynomial interpretations is the one most commonly used in automatic termination provers. As illustrated by Example 2 below, if only a single one of the coefficients a_i in any of the functions $f_{\mathcal{A}}$ is greater than 1, there might already exist derivations whose length is exponential in the size of the starting term.

Example 2. Consider the TRS $\mathcal{S}$ with the following single rule over the signature containing the function symbols a, b (arity 1), and c (arity 0). The system is example `SK90/2.50.trs` in the TPDB:

$$\mathsf{a}(\mathsf{b}(x)) \to \mathsf{b}(\mathsf{b}(\mathsf{a}(x)))$$

[1] `http://www.lri.fr/~marche/tpdb/`.

The following interpretation functions build a compatible linear polynomial interpretation $\mathcal{A}$ over $\mathbb{N}$:

$$\mathsf{a}_{\mathcal{A}}(x) = 2x \qquad \mathsf{b}_{\mathcal{A}}(x) = x + 1 \qquad \mathsf{c}_{\mathcal{A}} = 0$$

If we start a rewrite sequence from the term $\mathsf{a}^n(\mathsf{b}(\mathsf{c}))$, we reach the normal form $\mathsf{b}^{2^n}(\mathsf{a}^n(\mathsf{c}))$ after $2^n - 1$ rewriting steps. Therefore, the derivational complexity of $\mathcal{S}$ is at least exponential.

3.2. Context Dependent Interpretations

Even though polynomial interpretations provide an easy way to obtain an upper bound on the derivational complexity of a TRS, they are not very suitable for proving polynomial derivational complexity. Strongly linear interpretations only capture linear derivational complexity, but even a slight generalization admits already examples of exponential derivational complexity, as illustrated by Example 2. In (Hofbauer 2001), *context dependent interpretations* are introduced. They use an additional parameter (usually denoted by Δ) in the interpretation functions, which changes in the course of evaluating the interpretation of a term, thus making the interpretation dependent on the context. This way of computing interpretations also allows us to bridge the gap between linear and polynomial derivational complexity.

Definition 3. *A* context-dependent interpretation $\mathcal{C}$ *for some signature* $\mathcal{F}$ *consists of functions* $\{f_{\mathcal{C}}[\Delta] : (\mathbb{R}_0^+)^n \to \mathbb{R}_0^+ \mid f \in \mathcal{F}, n = \mathsf{arity}(f), \Delta \in \mathbb{R}^+\}$ *and* $\{f_{\mathcal{C}}^i : \mathbb{R}^+ \to \mathbb{R}^+ \mid f \in \mathcal{F}, i \in \{1, \ldots, \mathsf{arity}(f)\}\}$. *Given a* Δ-assignment $\alpha : \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}_0^+$, *the evaluation of a term* t *by* $\mathcal{C}$ *is denoted by* $[\alpha, \Delta]_{\mathcal{C}}(t)$. *It is defined inductively as follows:*

$$\begin{aligned} [\alpha, \Delta]_{\mathcal{C}}(x) &= \alpha(\Delta, x) && \text{for } x \in \mathcal{V} \\ [\alpha, \Delta]_{\mathcal{C}}(f(t_1, \ldots, t_n)) &= f_{\mathcal{C}}[\Delta]([\alpha, f_{\mathcal{C}}^1(\Delta)]_{\mathcal{C}}(t_1), \ldots, [\alpha, f_{\mathcal{C}}^n(\Delta)]_{\mathcal{C}}(t_n)) && \text{for } f \in \mathcal{F} \end{aligned}$$

Definition 4. *For each* $\Delta \in \mathbb{R}^+$, *let* $>_\Delta$ *be the order defined by* $a >_\Delta b \iff a - b \geqslant \Delta$. *A context-dependent interpretation* $\mathcal{C}$ *is* compatible *with a TRS* $\mathcal{R}$ *if for all rewrite rules* $l \to r$ *in* $\mathcal{R}$, *all* $\Delta \in \mathbb{R}^+$, *and every* Δ*-assignment* α, *we have* $[\alpha, \Delta]_{\mathcal{C}}(l) >_\Delta [\alpha, \Delta]_{\mathcal{C}}(r)$.

Definition 5. *A* Δ-linear interpretation *is a context dependent interpretation* $\mathcal{C}$ *whose interpretation functions have the form*

$$f_{\mathcal{C}}[\Delta](z_1, \ldots, z_n) = \sum_{i=1}^{n} a_{(f,i)} z_i + \sum_{i=1}^{n} b_{(f,i)} z_i \Delta + c_f \Delta + d_f \quad f_{\mathcal{C}}^i(\Delta) = \frac{\Delta}{a_{(f,i)} + b_{(f,i)} \Delta}$$

with $a_{(f,i)}, b_{(f,i)}, c_f, d_f \in \mathbb{N}, a_{(f,i)} + b_{(f,i)} \neq 0$ *for all* $f \in \mathcal{F}$, $1 \leqslant i \leqslant n$. *If we have* $a_{(f,i)} \in \{0, 1\}$ *for all* f, i, *we also call it a* Δ-restricted interpretation

We consider Δ-linear interpretations because of the similarity between the functions $f_{\mathcal{C}}[\Delta]$ and the interpretation functions of linear polynomial interpretations. Another point of interest is that the simple syntactical restriction to Δ-restricted interpretations yields a quadratic upper bound on the derivational complexity. Moreover, because of the special shape of Δ-linear interpretations, we need no additional monotonicity criterion for our main theorems:

Theorem 6 ((Moser & Schnabl 2008)). *Let $\mathcal{R}$ be a TRS and suppose that there exists a compatible Δ-linear interpretation. Then $\mathcal{R}$ is terminating and $\mathsf{dc}_{\mathcal{R}}(n) = 2^{\mathcal{O}(n)}$.*

Theorem 7 ((Schnabl 2007)). *Let $\mathcal{R}$ be a TRS and suppose that there exists a compatible Δ-restricted interpretation. Then $\mathcal{R}$ is terminating and $\mathsf{dc}_{\mathcal{R}}(n) = \mathcal{O}(n^2)$.*

Example 8. Consider the TRS given in Example 1 again. A compatible Δ-restricted (and Δ-linear) interpretation $\mathcal{C}$ is built from the following interpretation functions:

$$
\begin{array}{lll}
+_{\mathcal{C}}[\Delta](x,y) = (1+\Delta)x + y + \Delta & +^1_{\mathcal{C}}(\Delta) = \dfrac{\Delta}{1+\Delta} & +^2_{\mathcal{C}}(\Delta) = \Delta \\
\text{-}_{\mathcal{C}}[\Delta](x,y) = x + y + \Delta & \text{-}^1_{\mathcal{C}}(\Delta) = \Delta & -^2_{\mathcal{C}}(\Delta) = \Delta \\
\mathsf{s}_{\mathcal{C}}[\Delta](x) = x + \Delta + 1 & \mathsf{s}^1_{\mathcal{C}}(\Delta) = \Delta & \mathsf{0}_{\mathcal{C}}[\Delta] = 0
\end{array}
$$

Note that this interpretation gives a quadratic upper bound on the derivational complexity. However, from the polynomial interpretation given in Example 1, we can only infer an exponential upper bound (Hofbauer & Lautemann 1989). Consider the term $P_{n,n}$, where we define $P_{0,n} = \mathsf{s}^n(\mathsf{0})$ and $P_{m+1,n} = +(P_{m,n}, \mathsf{0})$. We have $|P_{n,n}| = 3n + 1$. For every $m, n \in \mathbb{N}$, $P_{m+1,n}$ rewrites to $P_{m,n}$ in $n+1$ steps. Therefore, $P_{n,n}$ reaches its normal form $s^n(0)$ after $n(n+1)$ rewriting steps. Hence, the derivational complexity is also $\Omega(n^2)$ for this example, so the inferred bound $\mathcal{O}(n^2)$ is tight.

4. Implementation

`cdiprover3` is written fully in `OCaml`[2]. It employs the libraries of the termination prover TTT2[3]. From these libraries, functionality for handling TRSs and SAT encodings, and an interface to the SAT solver `MiniSAT`[4] are used. Without counting this, the tool consists of about 1700 lines of OCaml code. About 25% of that code are devoted to the manipulation of polynomials and extensions of polynomials that stem from our use of the parameter Δ. Another 35% are used for constructing parametric interpretations and building suitable Diophantine constraints (see below) which enforce the necessary conditions for termination. Using TTT2's library for propositional logic and its interface to `MiniSAT`, 15% of the code deal with encoding Diophantine constraints into SAT. The remaining code is used for parsing input options and the given TRS, generating output, and controlling the program flow.

In order to find polynomial interpretations automatically, Diophantine constraints are generated according to the procedure described in (Contejean, et al. 2005). Putting an upper bound on the coefficients makes the problem finite. Essentially following (Fuhs, et al. 2007), we then encode the (finite domain) constraints into a propositional satisfiability problem. This problem is given to `MiniSAT`. From a satisfying assignment for the SAT problem, we construct a polynomial interpretation which is monotone and compatible with the given TRS.

This procedure is also the basis of the automatic search for Δ-linear and Δ-restricted interpretations. The starting point of that search is an interpretation with uninstantiated

[2] `http://caml.inria.fr`.
[3] `http://colo6-c703.uibk.ac.at/ttt2`.
[4] `http://minisat.se`.

Table 1: Performance of `cdiprover3`

Method	SL	SL+Δ-restricted	Δ-linear	Δ-restricted			
`-i -b X`	31	31	31	3	7	15	31
# success	41	87	83	83	86	86	86
average success time	20	3010	5527	3652	4041	4008	3986
# timeout	0	237	797	144	189	221	238

coefficients. If we want to be able to apply Theorem 6 or 7, we need to find coefficients which make the resulting interpretation compatible with the given TRS. Furthermore, we need to make sure that no divisions by zero occur in the interpretation functions. Again, we encode these properties into Diophantine constraints on the coefficients of a Δ-linear or Δ-restricted interpretation. The encoding is an adaptation of the procedure in (Contejean et al. 2005) to context-dependent interpretations. It is described in detail in (Schnabl 2007, Moser & Schnabl 2008). Once we have built the constraints, we continue using the same techniques as for searching polynomial interpretations: we encode the constraints in a propositional satisfiability problem, apply the SAT solver, and use a satisfying assignment to construct a context-dependent interpretation.

Table 1 shows experimental results of applying `cdiprover3` on the 957 known terminating examples of the TPDB. The tests were performed single-threaded on a 2.40 GHz Intel® Core™ 2 Duo with 2 GB of memory. For each system, `cdiprover3` was given a timeout of 60 seconds. All times in the table are given in milliseconds. The method SL denotes strongly linear interpretations. In all tests, we called `cdiprover3` with the options `-i -b X` (see Section 5. below), where `X` is specified in the second row of the table. As we can see, `cdiprover3` is currently able to prove polynomial derivational complexity for 87 of the 368 known terminating non-duplicating rewrite systems of the TPDB (duplicating rewrite systems have at least exponential derivational complexity, so this restriction is harmless here). The results indicate that an upper bound of 7 on the coefficient variables suffices to capture all examples on our test set. Therefore, 3 and 7 seem to be good candidates for default values of the `-b` option. However, it should be noted that our handling of the divisions introduced by the functions f_C^i is computationally rather expensive, which is indicated by the number of timeouts and the average time needed for successful proofs. This also explains the slight decrease in performance when we extend the search space to Δ-linear interpretations. However, there is one system which can be handled by Δ-linear interpretations, but not by Δ-simple interpretations: system `SK90/2.50` in the TPDB, which we mentioned in Example 2.

5. Using cdiprover3

`cdiprover3` is called from command line. The basic usage pattern for `cdiprover3` is

```
$ ./cdiprover3 <options> <filename> <timeout>
```

- `<timeout>` specifies the maximum number of seconds until `cdiprover3` stops looking for a suitable interpretation.

- <filename> specifies the path to the file which contains the considered TRS.
- For <options>, the following switches are available:

  -c <class> defines the desired subclass of the searched polynomial or context-dependent interpretation. The following values of <class> are legal:

  linear, simple, simplemixed, quadratic These classes correspond to the respective subclasses of polynomial interpretations, as defined in (Steinbach 1992). Linear polynomial interpretations imply an exponential upper bound on the derivational complexity. The other classes imply a double exponential upper bound, cf. (Hofbauer & Lautemann 1989).

  pizerolinear, pizerosimple, pizerosimplemixed, pizeroquadratic For these values, cdiprover3 tries to find a polynomial interpretation with the following restrictions: defined function symbols are interpreted by linear, simple, simple-mixed, or quadratic polynomials, respectively. Constructors are interpreted by strongly linear polynomials. These interpretations guarantee that the derivation length of all constructor based terms is polynomial (Bonfante et al. 2001).

  sli This option corresponds to strongly linear interpretations. As mentioned in Section 3., they induce a linear upper bound on the derivational complexity of a compatible TRS.

  deltalinear This value specifies that the tool should search for a Δ-linear interpretation. By Theorem 6, compatibility with such an interpretation implies an exponential upper bound on the derivational complexity.

  deltarestricted This option corresponds to Δ-restricted interpretations. By Theorem 7, they induce a quadratic upper bound.

  -b <bound> sets the upper bound for the coefficient variables. The default value for this bound is 3.

  -i This switch activates an incremental strategy for handling the upper bound on the coefficient variables. First, cdiprover3 tries to find a solution using an intermediate upper bound of 1 (which corresponds to encoding each coefficient variable by one bit). Whenever the tool fails to find a proof for some upper bound b, it is checked whether b is equal to the bound specified by the -b option. If that is the case, then the search for a proof is given up. Otherwise, b is set to the minimum of the bound specified by the -b option and $2(b+1)-1$ (which corresponds to increasing the number of bits used for each coefficient variable by 1).

If the -c switch is not specified, then the standard strategy for proving polynomial derivational complexity is employed. First, cdiprover3 looks for a strongly linear interpretation. If that is not successful, then a suitable Δ-restricted interpretation is searched. The input TRS files are expected to have the same format as the files in the TPDB. The format specification for this database is available at http://www.lri.fr/~marche/tpdb/format.html.

The output given by cdiprover3, as exemplified by Example 9, is structured as follows. The first line contains a short answer to the question whether the given TRS

is terminating: `YES`, `MAYBE`, or `TIMEOUT`. The latter means that `cdiprover3` was still busy after the specified timeout. `MAYBE` means that a termination proof could not be found, and `cdiprover3` gave up before time ran out. The answer `YES` indicates that an interpretation of the given class has been found which guarantees termination of the given TRS. It is followed by the inferred bound on the derivational complexity and a listing of the interpretation functions. After the interpretation functions, the elapsed time between the call of `cdiprover3` and the output of the proof is given. In all cases, the answer is concluded by statistics stating the total number of monomials in the constructed Diophantine constraints, and the upper bound for the coefficients that was used in the last call to `MiniSAT`.

Example 9. Given the TRS shown in Example 1, `cdiprover3` produces the output shown in Figure 1. The interpretations in Example 8 and in the output are equivalent. Note that the parameter Δ in the interpretation functions $f_{\mathcal{C}}[\Delta]$ is treated like another argument of the function. The interpretation functions $f^i_{\mathcal{C}}$ are represented by f_tau_i in the output.

6. Conclusion

In this paper, we have presented the (as far as we know) first tool which is specifically designed for automatically proving polynomial derivational complexity of term rewriting. We have also given a brief introduction into the applied proof methods. With our current implementation, we are able to prove polynomial derivational complexity for 87 of the 368 known terminating non-duplicating rewrite systems of the TPDB. By adding new termination methods to our tool which can prove polynomial derivational complexity of rewrite systems, we could extend the range of problems that the prover can solve. The *matchbounds* technique comes to mind here, which induces a linear upper bound on the derivational complexity of the considered system (Geser, et al. 2007, Korp & Middeldorp 2007). Another avenue for future work is the search for other subclasses of context-dependent interpretations which imply non-quadratic and non-linear, but polynomial upper bounds on the derivational complexity. A further possibility would be to find more efficient ways of handling the divisions introduced by the functions $f^i_{\mathcal{C}}$. Results in this area would help to further improve the power of `cdiprover3`.

References

M. Avanzini & G. Moser (2008). 'Complexity Analysis by Rewriting'. In *Proc. 9th FLOPS*, vol. 4989 of *LNCS*, pp. 130–146.

F. Baader & T. Nipkow (1998). *Term Rewriting and All That*. Cambridge University Press.

G. Bonfante, et al. (2001). 'Algorithms with Polynomial Interpretation Termination Proof'. *J. Funct. Program.* **11**(1):33–53.

G. Bonfante, et al. (2007). 'Quasi-interpretation Synthesis by Decomposition'. In *Proc. 4th ICTAC*, vol. 4711 of *LNCS*, pp. 410–424.

E. Contejean, et al. (2005). 'Mechanically Proving Termination Using Polynomial Interpretations.'. *J. Autom. Reason.* **34**(4):325–363.

C. Fuhs, et al. (2007). 'SAT Solving for Termination Analysis with Polynomial Interpretations'. In *Proc. SAT 2007*, vol. 4501 of *LNCS*, pp. 340–354.

A. Geser, et al. (2007). 'On Tree Automata that Certify Termination of Left-Linear Term Rewriting Systems'. *Inf. Comput.* **205**(4):512–534.

D. Hofbauer (2001). 'Termination Proofs by Context-Dependent Interpretations'. In *Proc. 12th RTA*, vol. 2051 of *LNCS*, pp. 108–121.

D. Hofbauer & C. Lautemann (1989). 'Termination Proofs and the Length of Derivations'. In *Proc. 3rd RTA*, vol. 355 of *LNCS*, pp. 167–177.

M. Korp & A. Middeldorp (2007). 'Proving Termination of Rewrite Systems using Bounds'. In *Proc. 18th RTA*, vol. 4533 of *LNCS*, pp. 273–287.

D. Lankford (1979). 'On proving Term-Rewriting Systems are Noetherian'. Tech. Rep. MTP-2, Math. Dept., Louisiana Tech. University.

P. Lescanne (1995). 'Termination of Rewrite Systems by Elementary Interpretations'. *Formal Aspects of Computing* **7**(1):77–90.

J.-Y. Marion (2003). 'Analysing the Implicit Complexity of Programs'. *Inf. Comput.* **183**(1):2–18.

G. Moser & A. Schnabl (2008). 'Proving Quadratic Derivational Complexities using Context Dependent Interpretations'. In *Proc. 19th RTA*. Accepted for publication.

A. Schnabl (2007). 'Context Dependent Interpretations[5]'. Master's thesis, Universität Innsbruck.

J. Steinbach (1992). 'Proving Polynomials Positive'. In *Proc. 12th FSTTCS*, vol. 652 of *LNCS*, pp. 191–202.

TeReSe (2003). *Term Rewriting Systems*, vol. 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

[5]Available online at `http://cl-informatik.uibk.ac.at/~aschnabl/`

Figure 1: Output produced by `cdiprover3`.

```
$ cat tpdb-4.0/TRS/SK90/2.11.trs
(VAR x y)
(RULES
+(0,y) -> y
+(s(x),y) -> s(+(x,y))
-(0,y) -> 0
-(x,0) -> x
-(s(x),s(y)) -> -(x,y)
)
(COMMENT Example 2.11 (Addition and Subtraction) in \cite{SK90})
$ ./cdiprover3 -i tpdb-4.0/TRS/SK90/2.11.trs 60
YES
QUADRATIC upper bound on the derivational complexity

This TRS is terminating using the deltarestricted interpretation
-(delta, X1, X0) = + 1*X0 + 1*X1 + 0 + 0*X0*delta + 0*X1*delta + 1*delta
s(delta, X0) = + 1*X0 + 1 + 0*X0*delta + 1*delta
0(delta) = + 0 + 0*delta
+(delta, X1, X0) = + 1*X0 + 1*X1 + 0 + 0*X0*delta + 1*X1*delta + 1*delta
-_tau_1(delta) = delta/(1 + 0 * delta)
-_tau_2(delta) = delta/(1 + 0 * delta)
s_tau_1(delta) = delta/(1 + 0 * delta)
+_tau_1(delta) = delta/(1 + 1 * delta)
+_tau_2(delta) = delta/(1 + 0 * delta)

Time: 0.024418 seconds
Statistics:
Number of monomials: 187
Last formula building started for bound 1
Last SAT solving started for bound 1
```