POLO — A System for Termination Proofs using Polynomial Orderings* 

Jürgen Giesl
FB Informatik, Technische Hochschule Darmstadt,
Alexanderstr. 10, 64283 Darmstadt, Germany
Email: giesl@inferenzsysteme.informatik.th-darmstadt.de

Abstract
POLO is a system for both semi-automatic and fully automated termination proofs of term rewriting systems using polynomial orderings. In this paper we describe the system's user interface and illustrate its performance with some examples.

1 Introduction

In [Gie95a] and [Gie95b] we presented a method for automated termination proofs of term rewriting systems using polynomial orderings. This technique has been implemented in the POLO system. It runs under Common Lisp and is available by anonymous ftp from kirmes.inferenzsysteme.informatik.th-darmstadt.de under pub/termination.

To run the system please start your Lisp, compile the file polo.lisp, load the compiled version and change the package to polo. More precisely, you have to enter the following commands. User inputs are printed in bold face.

```
> (compile-file "polo.lisp")
> (load "polo.sbin")
> (in-package 'polo)
```

The following section explains the user interface of the POLO system. In section 3 we provide a table of experiments run with the system to illustrate its performance and we end up with a conclusion in section 4. In the appendix we present a digest of some term rewriting systems whose termination could be proved with the POLO system.

*Technical Report IBN 95/24, Technische Hochschule Darmstadt
2 The User Interface

To start the system you have to call the function start.

\[ > \text{(start)} \]

Then the system prints out the following start message.

\[ \text{POLO} \]

A system for termination proofs with POLynomial Orderings.

Hit H for help.

\[ >> \]

The \[ >> \] prompt indicates that you are currently on the top level of the POLO program. You can now enter commands by typing them in and hitting Return afterwards.

By entering the command \[ H \] (for Help) the current menu of possible commands will be printed.

\[ >> \text{H} \]

[\[L\]] Load trs
[\[M\]] Change method for generating polynomial orderings
[\[Q\]] Quit
[\[H\]] Help

To quit the program you must hit Q.

In the following we will explain how to load term rewriting systems from files (section 2.1), how to prove their termination (section 2.2) and how to inspect and to change the termination proof method (section 2.3).

2.1 Loading Files

To prove the termination of a term rewriting system (trs) you first have to load the trs from a file. For that purpose you have to enter the command L.

\[ >> \text{L} \]

Enter a filename:

Accompanying the system there is a directory examples of example files. One of the easiest example in these files is NestedFunctionSymbols.
The term rewriting system in the example file must begin with the rules of the trs. The left hand side and the right hand side of a rule are separated by "\rightarrow". Rules must be separated by commas and there has to be a semicolon after the last rule. Following the semicolon there must be the word functions followed by a ";" and a list of all function symbols in the trs. The function symbols in the list are separated by commas and after the last function symbol there must be a dot. Please note that every symbol in the trs which is not declared as a function symbol will be regarded as a variable. The system will ignore everything following the dot. For example the file Nested_Function_Symbols has the following contents.

\begin{verbatim}
f(f(x)) \rightarrow g(g(x)),
g(g(f(x))) \rightarrow f(g(g(x)));
\end{verbatim}

functions: f, g.

-----------------------------

As x is not declared as a function symbol it will be regarded as a variable.

\subsection{2.2 Proving Termination}

By entering \texttt{H} again you will notice that the menu of possible command has changed.

\begin{verbatim}
>> H
\end{verbatim}

Termination proof with
\begin{verbatim}
[1] polynomials of degree 1
[2] simple-mixed polynomials
[V] provided values
[G] given polynomials
\end{verbatim}
Apart from loading term rewriting systems (by entering the command L again) you can now prove termination of the current trs by entering one of the four commands 1, S, V or G.

### 2.2.1 Polynomials of Degree 1

If you enter the command 1 the system will try to prove termination of the current trs using polynomials of degree 1. First it will print the approach applied, i.e. the used polynomial interpretation with variable coefficients. Variable coefficients have the names V0, V1 etc. Here, μ stands for μ.

Then the system tries to find an instantiation of these variable coefficients with numbers such that the current trs is compatible with the resulting polynomial ordering. The run time spent for this search is printed out to the user.

If the system has found a solution it will print the resulting polynomial interpretation.

```plaintext
>> 1

Approach:
μ is mapped to the polynomial V0
f(X1) is mapped to the polynomial V1 + V2 X1
g(X1) is mapped to the polynomial V3 + V4 X1

Begin of the Termination Proof

Run time: 0.03 seconds

Termination Proof succeeded!

Solution:
g(X1) is mapped to the polynomial 1 + 2 X1
f(X1) is mapped to the polynomial 2 + 2 X1
μ is mapped to the polynomial 2

>>
```
2.2.2 The Actual Polynomial Ordering

If you hit H again you will notice that the top level menu now contains two additional commands A and I.

```
>> H

Termination proof with
[A] actual polynomial ordering
[1] polynomials of degree 1
[S] simple-mixed polynomials
[V] provided values
[G] given polynomials
[I] Inspect current polynomial ordering
[1] Load new trs (current one is Nested.Function.Symbols)
[X] Change method for generating polynomial orderings
[Q] Quit
[H] Help
```

The actual (or current) polynomial ordering is the polynomial ordering used in the last approach for termination proofs (i.e. it can contain variable coefficients). To have a look at this polynomial ordering please enter the command I.

```
>> I

Approach:
mu is mapped to the polynomial V0
f(X1) is mapped to the polynomial V1 + V2 X1
g(X1) is mapped to the polynomial V3 + V4 X1
```

If you enter the command A another termination proof with the same approach will be tried, i.e. the same polynomial ordering (possibly with variable coefficients) will be used. This command is useful if you have changed the method for generating polynomial orderings and want to re-attempt a proof using the same polynomial ordering, cf. section 2.3.
Begin of the Termination Proof

Run time: 0.02 seconds

Termination Proof succeeded!

Solution:
g(x) is mapped to the polynomial 1 + 2 x
f(x) is mapped to the polynomial 2 + 2 x
mu is mapped to the polynomial 2

Note that repeated proof attempts for the same trs using the same polynomial ordering (and the same method for the generation of the solution) may have different run times. As the system uses time limits and certain steps are aborted if certain time limits are exceeded, it is possible that repeated proof attempts lead to different solutions. It is even possible that one proof attempt fails while the next one succeeds.

2.2.3 Simple-Mixed Polynomials

If you enter the command S the system will try to prove termination of the current trs using simple-mixed polynomials. An exception is made for unary function symbols. Here, instead of simple-mixed polynomials the system uses polynomials of degree 2 (x^2 stands for x^2). If you also want to use simple-mixed polynomials for unary function symbols you have to use one of the options that are described in the next two sections.

Approach:
mu is mapped to the polynomial v0
f(x) is mapped to the polynomial v1 + v2 x + v3 x^2
g(x) is mapped to the polynomial v4 + v5 x + v6 x^2

Begin of the Termination Proof

Run time: 0.87 seconds

Termination Proof succeeded!
Solution:
g(X1) is mapped to the polynomial 2 + X1^2
f(X1) is mapped to the polynomial 2 + 2 X1 + 2 X1^2
mu is mapped to the polynomial 2

>>

2.2.4 Polynomials with Provided Values

The command V is also used to prove termination with a simple-mixed polynomial ordering, but in contrast to the command S you can now set variable coefficients to special (numeric) values. For each function symbol you are asked if you want to replace some variable coefficients by numbers. If you hit Y, then you can enter equations of the form “V0 = -1” etc. After each equation you must hit Return. If you do not want to enter any more equations for the current function symbol hit one more Return. The numbers can also be negative and/or rational (e.g. “V0 = 2/3”).

For instance you can perform the following termination proof where the value of mu is fixed (mu = -1) and where g(x) is only mapped to a polynomial of the form 1 + vx.

>> V

mu is mapped to the polynomial V0
Replace some variable coefficients [y/n]? Y
V0 = -1
f(X1) is mapped to the polynomial V1 + V2 X1 + V3 X1^2
Replace some variable coefficients [y/n]? N
g(X1) is mapped to the polynomial V4 + V5 X1 + V6 X1^2
Replace some variable coefficients [y/n]? Y
V4 = 1
V6 = 0

Approach:
mu is mapped to the polynomial -1
f(X1) is mapped to the polynomial V1 + V2 X1 + V3 X1^2
g(X1) is mapped to the polynomial 1 + V5 X1
Begin of the Termination Proof

Run time: 0.06 seconds

Termination Proof succeeded!

Solution:
\[
g(X_1) \text{ is mapped to the polynomial } 1 + 2X_1
\]
\[
f(X_1) \text{ is mapped to the polynomial } 2 + 2X_1
\]
\[
\mu \text{ is mapped to the polynomial } -1
\]

For a comment on the used termination criterion the reader is referred to section 2.3.

The system immediately checks whether the polynomial is dependent on all its formal parameters. If not, it will force the user to redefine the polynomial for the current function symbol.

After the proof the actual polynomial ordering is the one mentioned as approach. You can hit I to inspect it and A to re-attempt a termination proof with it.

2.2.5 Given Polynomials

With the commands described up to now only termination proofs using simple-mixed polynomials are possible. If you want to use arbitrary polynomial orderings you should enter the command G. Then for each function symbol f the system prints "f is mapped to the polynomial" after which you must enter a polynomial for the function symbol f. These polynomial interpretations can have arbitrary degree and can contain variable coefficients as well as numerical coefficients. For a variable coefficient you have to enter "V". Please note that different occurrences of "V" denote different variable coefficients.

\[
\begin{align*}
&>> \text{G} \\
&\text{Enter a polynomial for } \mu : 2 \\
&\text{Enter a polynomial for } f(X_1) : VX_1 + X_1^3 \\
&\text{Enter a polynomial for } g(X_1) : V + X_1 + X_1^3 \\

\text{Approach:} \\
&\mu \text{ is mapped to the polynomial } 2 \\
&f(X_1) \text{ is mapped to the polynomial } V0X_1 + X_1^3 \\
&g(X_1) \text{ is mapped to the polynomial } V1 + X_1 + X1^3
\end{align*}
\]
Begin of the Termination Proof

Run time: 1.45 seconds

Termination Proof succeeded!

Solution:
g(X1) is mapped to the polynomial 1 + X1 + X1^3
f(X1) is mapped to the polynomial 2 X1 + X1^3
mu is mapped to the polynomial 2

>>

2.2.6 Polynomials from Files

It is also possible to define a polynomial ordering in the file containing the trs. In this case the list of the function symbols must not end with a dot, but with a semicolon. It should be followed by the word polynomials and a “;”. Then a polynomial ordering can be defined by a list of equations of the form “f(x1,…, xn) = polynomial”. Here f must be a function symbol, the names x1,…, xn of the formal parameters can be chosen arbitrarily and the polynomial can contain both numerical and variable coefficients. The equations must be separated by commas and after the last equation there must be a dot. As an example consider the file Running Example.

f(x, y) -> x,
g(a) -> h(a, b, a),
i(x) -> f(z, x),
h(x, x, y) -> g(x);

functions: a, g, i, h, f, b;

polynomials:
mu = 1,
a = v,
g(x) = v x + v x^2,
i(x) = v + v x,
h(x, y, z) = v x + v y + v z + v x y + v x z + v y z + v x y z,
f(x, y) = x + y,
b = 1.
Running Example 6.1 in (Steinbach, 91)
(Middeldorp, 89)

The above approach for the polynomial ordering is the one Steinbach uses in his system for the generation of polynomial orderings.

After loading this file (using the command L) a new command is included in the menu:

>> H

Termination proof with [F] ordering from file
...

If you hit F then the polynomial ordering defined in the file will be used for the termination proof.

2.3 Changing the Method for Generating Polynomial Orderings

In this section we explain how to change the method applied for termination proofs. If you type the command M the system informs you which method is currently used to find solutions for the variable coefficients. In POLO there are two possible methods for this purpose: TESTING and the modified COLLINS algorithm. In the following these methods will be discussed in more detail.

Following the command M the system enters a sub-menu for the inspection and for changing the termination proof method. This is indicated by a new prompt symbol =>.

>> M

Currently, solutions are generated by TESTING.
=>

By hitting H you can see the list of commands available in the sub-menu. Moreover, the currently used method is printed again.
Currently, solutions are generated by TESTING.

If you only wanted to know which of the two methods (TESTING or COLLINS) was currently used, but do not want to change the method or its parameters, then enter the command B to go back to the top-level menu.

In the following two sections we briefly explain the two methods implemented in the system and illustrate which parameters can be changed.

2.4 Testing

The method TESTING uses a “generate and test” approach to find solutions for the variable coefficients of the polynomial ordering. For this purpose all instantiations of the variables with numbers from a given list are generated until one of these instantiations leads to a polynomial ordering the TRS is compatible with.

If you are currently in the “change method” mode of the system (which is indicated by the prompt =>) then you can enter the command T to select the TESTING method. You will then be informed about the current values of the parameters of this method and you have the possibility to change them.

The TESTING method has two parameters. The first parameter is the list of numbers the variable coefficients can be instantiated with. When examining term rewriting systems occurring in the literature we noticed that most termination proofs only use polynomials whose coefficients are 0, 1 or 2. Therefore initially this list is (2 1 0). You can now either enter a new list or simply hit Return to keep the old values.

The second parameter is the amount of approximately allowed execution time. If this time limit is exceeded then the search for a solution will be aborted. Again it is possible to change the value by entering a new time limit or to keep the current time limit by simply hitting Return.

After setting (resp. inspecting) the parameters, the system goes back to the top-level. In the next termination proof the TESTING method with the selected parameters will be used.
You can change the values to be tested and the time limit. Hit Return to keep the old values.

Values to be tested (current values are (2 1 0)): (3 1 0 -1)
Time limit (current value is 20 seconds):

Note that the order of the numbers in the list of values to be tested is not irrelevant. The system uses the heuristic that the first numbers in the list are more likely to yield a successful polynomial interpretation than the last numbers in the list. Therefore most of the allowed execution time is spent to test instantiations with numbers at the beginning of the list.

The system uses the additional heuristic that if there is only few allowed execution time left then at least the numbers 0 and 1 should be tested. Therefore the found solution may contain the coefficients 0 and 1 even if these numbers are not included in the list of values to be tested.

Moreover, the order of the declared function symbols (functions: f1, f2, ...) in the file is also important, as the time spent searching for suited polynomials is not equally shared among the function symbols. Function symbols with a small arity get more time than those with higher arity and if $f$ and $g$ have the same arity and $f$ precedes $g$ in the list of declared function symbols, then $f$ gets more time than $g$.

2.5 The modified Collins algorithm

As described in [Gie95a], [Gie95b] the solutions can also be computed using an incomplete modification of Collins' decision algorithm [Col75], [ACM84]. To choose this method you have to enter the command C in the "change method" mode of the system.

Again there are several parameters you can inspect and change. If you want to keep an old value simply hit Return.

- The first parameter is the approximate time limit after which the search for a solution will be aborted.
- The next parameter is the percentage of the allowed execution time that may be spent for the projection phase of Collins' algorithm.
- You can decide whether the algorithm should also examine solutions with negative coefficients.
- Moreover, you can decide whether the algorithm should also examine rational coefficients.
You can change the time limit and the percentage of time spent for projection. Moreover, you can decide whether the algorithm should also examine negative and rational numbers. Hit Return to keep the old values.

Time limit (current value is 40 seconds): 60
Percentage of time spent for projection (current value is 1/3):
Examination of negative numbers [Y/N] (current value is N):
Examination of rational numbers [Y/N] (current value is Y): N

Depending on the method and the selected parameters different termination criteria are used. If all numerical coefficients of the actual polynomial interpretation and all possible solutions for the variable coefficients are integers, then the system uses the termination criterion of theorem 2 in [Gie95b]. Otherwise the termination criterion of theorem 3 in [Gie95b] (i.e. theorem 1 in [Gie95a]) is used.

3 Experimental Results

In this section we illustrate the performance of the POLO system with some examples. Table 1 summarizes these results (run on a Sun SPARC-2). The files containing the examined term rewriting systems are included in the appendix and can be found in the directory examples.

If the POLO system is used in a semi-automatic way then checking whether a term rewriting system is compatible with a given polynomial ordering (i.e. an ordering without variable coefficients) can be done extremely quickly (usually in significantly less than a second).

But for many term rewriting systems it is possible to generate a suited simple-mixed polynomial ordering completely automatically in 1 or 2 seconds. (Here, unary function symbols are mapped to polynomials of the form \(a + bx + cx^2\).) Such examples are Nested Function Symbols, Endomorphism Associativity, Distributivity Associativity etc. For such systems one should use the TESTING method with the values (2 1 0). The use of COLLINS' method is not advisory here, as it is significantly more time-consuming.

For larger systems (such as Running Example or Symbolic Differentiation) POLO can also generate a suited polynomial ordering very quickly, if the value of a few variable coefficients is fixed before. For instance, for Running Example we took the approach used by Steinbach in his system for the generation of polynomial orderings [Ste91] and for Symbolic Differentiation we set the values of constants to 2.
<table>
<thead>
<tr>
<th>File</th>
<th>Approach</th>
<th>Method</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested Function Symbols [Ste91, Example 8.1]</td>
<td>1</td>
<td>TESTING</td>
<td>0.03 sec.</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>TESTING</td>
<td>0.9 sec.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>COLLINS</td>
<td>19.8 sec.</td>
</tr>
<tr>
<td>Flatten</td>
<td>1</td>
<td>TESTING</td>
<td>0.04 sec.</td>
</tr>
<tr>
<td>Flatten</td>
<td>S</td>
<td>TESTING</td>
<td>0.8 sec.</td>
</tr>
<tr>
<td>Flatten</td>
<td>V</td>
<td>COLLINS</td>
<td>20.4 sec.</td>
</tr>
<tr>
<td>Stack [DJ90, p. 253]</td>
<td>1</td>
<td>TESTING</td>
<td>0.1 sec.</td>
</tr>
<tr>
<td>Boolean Ring_1 [HD83], [Der87, p. 102]</td>
<td>S</td>
<td>TESTING</td>
<td>0.3 sec.</td>
</tr>
<tr>
<td>Boolean Ring_2 [Hsi82], [BL87, p. 152]</td>
<td>S</td>
<td>TESTING</td>
<td>0.4 sec.</td>
</tr>
<tr>
<td>Neutral Elements Distributivity [Pau84], [Ste91, Example 8.13]</td>
<td>S</td>
<td>TESTING</td>
<td>0.4 sec.</td>
</tr>
<tr>
<td>Plus</td>
<td>S</td>
<td>TESTING</td>
<td>0.6 sec.</td>
</tr>
<tr>
<td>Endomorphism Associativity [Bc84], [BL87]</td>
<td>V</td>
<td>TESTING</td>
<td>0.1 sec.</td>
</tr>
<tr>
<td>Endomorphism Associativity [Bc84], [BL87]</td>
<td>S</td>
<td>TESTING</td>
<td>0.8 sec.</td>
</tr>
<tr>
<td>Endomorphism Associativity [Bc84], [BL87]</td>
<td>V</td>
<td>COLLINS</td>
<td>35.5 sec.</td>
</tr>
<tr>
<td>Distributivity Associativity [Der87, p. 78]</td>
<td>S</td>
<td>TESTING</td>
<td>1.9 sec.</td>
</tr>
<tr>
<td>Distributivity Associativity [Der87, p. 78]</td>
<td>V</td>
<td>COLLINS</td>
<td>25.1 sec.</td>
</tr>
<tr>
<td>File</td>
<td>Approach</td>
<td>Method</td>
<td>Time</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>Binomial Coefficients</strong></td>
<td>S</td>
<td>TESTING</td>
<td>1.6 sec.</td>
</tr>
<tr>
<td>[Ste91, Example 8.8], [Ste92, Example 13]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Running Example</strong></td>
<td>F</td>
<td>TESTING</td>
<td>0.2 sec.</td>
</tr>
<tr>
<td>[Mid89], [Ste91, Example 6.1]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Boolean Ring_3</strong></td>
<td>F</td>
<td>TESTING</td>
<td>2.6 sec.</td>
</tr>
<tr>
<td>[Ste91, Example 8.5]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Reverse</strong></td>
<td>F</td>
<td>TESTING</td>
<td>1.7 sec.</td>
</tr>
<tr>
<td>[Ste91, Example 8.6]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Symbolic Differentiation</strong></td>
<td>F</td>
<td>TESTING</td>
<td>3.4 sec.</td>
</tr>
<tr>
<td>[Knu73], [Der87, p. 79]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Symbolic Differentiation</strong></td>
<td>F</td>
<td>TESTING</td>
<td>0.6 sec.</td>
</tr>
<tr>
<td>[Knu73], [Der87, p. 79]</td>
<td></td>
<td>(1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Symbolic Differentiation_long</strong></td>
<td>F</td>
<td>TESTING</td>
<td>0.2 sec.</td>
</tr>
<tr>
<td>[Knu73], [Der87, p. 79]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Groups</strong></td>
<td>F</td>
<td>TESTING</td>
<td>0.9 sec.</td>
</tr>
<tr>
<td>[Hue80], [BL87, p. 151]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Taussky Group</strong></td>
<td>F</td>
<td>TESTING</td>
<td>5.5 sec.</td>
</tr>
<tr>
<td>[KB70], [BL87, p. 155], [Ste91, Example 8.4]</td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Fibonacci Group</strong></td>
<td></td>
<td>V</td>
<td>4.8 sec.</td>
</tr>
<tr>
<td>[Ste91, Example 8.2]</td>
<td></td>
<td>(μ = 0, (\text{comb}(x, y) = vx + vy))</td>
<td></td>
</tr>
<tr>
<td><strong>Fibonacci Group</strong></td>
<td>F</td>
<td>TESTING</td>
<td>0.3 sec.</td>
</tr>
<tr>
<td>[Ste91, Example 8.2]</td>
<td></td>
<td>(4 2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Fibonacci Group</strong></td>
<td>F</td>
<td>TESTING</td>
<td>0.8 sec.</td>
</tr>
<tr>
<td>[Ste91, Example 8.2]</td>
<td></td>
<td>(4 3 2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Fibonacci Group</strong></td>
<td>F</td>
<td>COLLINS</td>
<td>8.1 sec.</td>
</tr>
<tr>
<td>[Ste91, Example 8.2]</td>
<td></td>
<td>40 sec., 1/3, <em>neg., rat.</em></td>
<td></td>
</tr>
<tr>
<td><strong>Negative Coefficient</strong></td>
<td>F</td>
<td>TESTING</td>
<td>0.01 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2 1 0), 20 sec.</td>
<td></td>
</tr>
<tr>
<td><strong>Negative Coefficient</strong></td>
<td>F</td>
<td>V</td>
<td>7.3 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(plus(x, y) = x + y)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Experiments run with POLO.
Note that a manual termination proof of these term rewriting systems is not at all trivial. Proving termination of Symbolic Differentiation was one of the problems on a qualifying exam given at Carnegie-Mellon University in 1967 [Der87].

The POLO system can also be used for term rewriting systems which require non simple-mixed polynomial orderings (such as Taussky Group [KB70]) or for systems where a polynomial ordering with coefficients different from 2, 1, 0 is needed (e.g. Fibonacci Group [Ste91, Example 8.2]). For such examples the use of the COLLINS method may be useful, especially if one already knows the polynomials associated with certain function symbols and searches polynomials for the remaining function symbols. Finally, POLO also allows the use of negative coefficients (e.g. in the example Negative Coefficient).

4 Conclusion

POLO is a system for termination proofs of term rewriting systems using polynomial orderings. Virtually all other systems for this purpose are semi-automatic, i.e. the polynomial ordering is given by the user and the system has to check whether the trs is compatible with this polynomial ordering. If used in such a semi-automatic way POLO is extremely powerful (more efficient and slightly more powerful than all preceding systems, cf. [Gie95a], [Gie95b]).

Moreover, POLO can also be used to determine the values of variable coefficients. If the number of unknown coefficients is not too high, then it is likely that a solution will be found quickly. For most of the commonly used term rewriting systems the TESTING method with the coefficients (2 1 0) will suffice. Efficient application of this method is only possible, because the technique in [Gie95a], [Gie95b] makes the test whether the trs is compatible with a certain polynomial ordering trivial.

If there are only few variable coefficients and a solution with commonly used values cannot be found, then it is possible to use the modified incomplete COLLINS method to search for an instantiation. For instance, this may be useful if a trs compatible with a polynomial ordering is extended by some new rules introducing a new function symbol. Now the polynomial interpretation must also be extended for this function symbol. For the soundness of the incomplete COLLINS method the elimination of rule variables in the technique of [Gie95a], [Gie95b] is necessary.

The POLO system is only a prototype which may be refined in future implementations. Nevertheless it demonstrates how the method in [Gie95a], [Gie95b] leads to an efficient, powerful and easy to implement algorithm which can be used in both a semi-automatic and a fully automated way.
A Example Files

A.1 Nested Function Symbols

\[ \begin{align*} 
  f(f(x)) & \rightarrow g(g(x)), \\
  g(g(f(x))) & \rightarrow f(g(g(x))) \; ;
\end{align*} \]

functions: \( f, g \).

Nested Function Symbols
(Steinbach 91, Example 8.1)

A.2 Flatten

\[ \begin{align*} 
  \text{flatten}(\text{nil}) & \rightarrow \text{nil}, \\
  \text{flatten}(\text{cons}(\text{nil}, y)) & \rightarrow \text{cons}(\text{nil}, \text{flatten}(y)), \\
  \text{flatten}(\text{cons}(\text{cons}(u, v), w)) & \rightarrow \text{flatten}(\text{cons}(u, \text{cons}(v, w))) ;
\end{align*} \]

functions: \( \text{flatten}, \text{cons}, \text{nil} \).

Flatten

A.3 Stack

\[ \begin{align*} 
  \text{top}(\text{push}(x, y)) & \rightarrow x, \\
  \text{pop}(\text{push}(x, y)) & \rightarrow y, \\
  \text{alternate}(\text{empty}, z) & \rightarrow z, \\
  \text{alternate}(\text{push}(x, y), z) & \rightarrow \text{push}(x, \text{alternate}(z, y)) ;
\end{align*} \]

functions: \( \text{top}, \text{push}, \text{pop}, \text{alternate}, \text{empty} \).

Stack
(Dershowitz & Jouannaud, 90, p. 253)
A.4 Boolean_Ring_1

and(x, T) → x,
and(x, F) → F,
and(x, x) → x,
xor(x, F) → x,
xor(x, x) → F,
and(xor(x, y), c) → xor(and(x, c), and(y, c));

functions: and, xor, T, F.

=================================================================

Boolean_Ring_1
(Hsiang & Dershowitz, 83), (Dershowitz 87, p. 102)

A.5 Boolean_Ring_2

xor(z, F) → z,
xor(z, neg(x)) → F,
and(z, T) → x,
and(x, x) → x,
and(xor(x, y), z) → xor(and(x, z), and(y, z)),
xor(x, x) → F;

functions: and, xor, T, F, neg.

=================================================================

Boolean_Ring_2
(Hsiang, 82), (Ben Cherifa & Lescanne, 87, p. 152)

A.6 Neutral_Elements_Distributivity

plus(x, zero) → x,
times(x, one) → x,
times(x, zero) → zero,
times(x, plus(y, z)) → plus(times(x, y), times(x, z));

functions: times, plus, one, zero.

=================================================================
Neutral Elements, Distributivity
(Paul, 84), (Steinbach, 91, Example 8.13)

A.7 Plus

plus(x, zero) -> x,
plus(x, s(y)) -> s(plus(x, y)),
plus(plus(x, y), z) -> plus(x, plus(y, z));

functions: plus, s, zero.

Endomorphism and Associativity
(Bellegarde 84, Ben Cherifa & Lescanne 87)

A.8 Endomorphism, Associativity

composition(composition(x, y), z) -> composition(x, composition(y, z)),
composition(map(x), map(y)) -> map(composition(x, y)),
composition(map(x), composition(map(y), z)) ->
composition(map(composition(x, y)), z);

functions: map, composition.

Distributivity, Associativity

times(x, plus(y, z)) -> plus(times(x, y), times(x, z)),
times(plus(x, y), z) -> plus(times(x, z), times(y, z)),
plus(plus(x, y), z) -> plus(x, plus(y, z));

functions: plus, times.
Distributivity & Associativity
(Dershowitz 87, p.78)

A.10 Binomial Coefficients
bin(x, zero) -> s(zero),
bin(zero, s(y)) -> zero,
bin(s(x), s(y)) -> plus(bin(x, s(y)), bin(x, y));

functions: zero, s, plus, bin.

A.11 Running Example
f(x, y) -> x,
g(a) -> h(a, b, a),
i(x) -> f(x, x),
h(x, x, y) -> g(x);

functions: a, g, i, h, f, b;

polynomials:
m = 1,
a = v,
g(x) = v x + v x^2,
i(x) = v + v x,
h(x, y, z) = v x + v y + v z + v x y + v x z + v y z + v x y z,
f(x, y) = x + y,
b = 1.

Running Example 6.1 in (Steinbach, 91)
(Middeldorp, 89)
The above approach for the polynomial ordering is the one
Steinbach uses in his system for the generation of polynomial
orderings.
A.12 Boolean_Ring_3

\[
\begin{align*}
\text{impl}(x, y) & \rightarrow \text{xor}(\text{and}(x, y), \text{xor}(x, \text{true})) , \\
\text{or}(x, y) & \rightarrow \text{xor}(\text{and}(x, y), \text{xor}(x, y)) , \\
\text{equiv}(x, y) & \rightarrow \text{xor}(x, \text{xor}(y, \text{true})) , \\
\text{neg}(x) & \rightarrow \text{xor}(x, \text{true}); \\
\end{align*}
\]

functions: impl, true, or, equiv, and, xor, neg;

polynomials:
\[
\begin{align*}
\mu &= 1 , \\
\text{true} &= 1 , \\
\text{impl}(x, y) &= v + v x + v y , \\
\text{equiv}(x, y) &= v + v x + v y , \\
\text{or}(x, y) &= v + v x + v y , \\
\text{neg}(x) &= v + v x , \\
\text{and}(x, y) &= v + v x + v y , \\
\text{xor}(x, y) &= v + v x + v y . \\
\end{align*}
\]

A.13 Reverse

\[
\begin{align*}
\text{append}(\text{nil}, y) & \rightarrow y , \\
\text{append}(\text{add}(x, y), z) & \rightarrow \text{add}(x, \text{append}(y, z)) , \\
\text{append}(\text{append}(x, y), z) & \rightarrow \text{append}(x, \text{append}(y, z)) , \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} , \\
\text{reverse}(\text{add}(x, y)) & \rightarrow \text{append}(\text{reverse}(y), \text{add}(x, \text{nil})) , \\
\text{reviter}(\text{nil}, y) & \rightarrow y , \\
\text{reviter}(\text{add}(x, y), z) & \rightarrow \text{reviter}(y, \text{add}(x, z)) , \\
\text{append}(\text{reverse}(x), y) & \rightarrow \text{reviter}(x, y) , \\
\text{reverse}(x) & \rightarrow \text{reviter}(x, \text{nil}); \\
\end{align*}
\]

functions: reverse, reviter, add, append, nil;

polynomials:
\[
\begin{align*}
\mu &= 2 , \\
\text{nil} &= 2 , \\
\text{reverse}(x) &= v + v x^2 , \\
\text{add}(x, y) &= v + v x + v y + v x y , \\
\text{reviter}(x, y) &= v x + v x y , \\
\end{align*}
\]
append(x, y) = v x + v x y.

Boolean_Ring_3
(Steinbach 91, Example 8.6)

A.14 Symbolic_Differentiation
Dx(x) -> one,
Dx(a) -> zero,
Dx(plus(alpha, beta)) -> plus(Dx(alpha), Dx(beta)),
Dx(times(alpha, beta)) ->
    plus(times(beta, Dx(alpha)), times(alpha, Dx(beta))),
Dx(minus(alpha, beta)) -> minus(Dx(alpha), Dx(beta)),
Dx(neg(alpha)) -> neg(Dx(alpha));

functions: one, zero, x, a, Dx, plus, times, minus, neg;

polynomials:
mu = 2,
zero = 2,
one = 2,
a = 2,
x = 2,
Dx(alpha) = v + v alpha^2,
plus(alpha, beta) = v + v alpha + v beta,
times(alpha, beta) = v + v alpha + v beta,
minus(alpha, beta) = v + v alpha + v beta,
neg(alpha) = v + v alpha.

Symbolic Differentiation
(Knuth, 73), (Dershowitz 87, p. 79)

A.15 Symbolic_Differentiation_long
Dx(x) -> one,
Dx(a) -> zero,
Dx(plus(alpha, beta)) -> plus(Dx(alpha), Dx(beta)),

22
\[
\begin{align*}
Dx(\times(\alpha, \beta)) & \rightarrow \\
& \quad \text{plus}(\times(\beta, Dx(\alpha)), \times(\alpha, Dx(\beta))), \\
Dx(\min(\alpha, \beta)) & \rightarrow \min(Dx(\alpha), Dx(\beta)), \\
Dx(\neg(\alpha)) & \rightarrow \neg(Dx(\alpha)), \\
Dx(\div(\alpha, \beta)) & \rightarrow \\
& \quad \min(\div(Dx(\alpha), \beta), \times(Dx(\alpha), \div(Dx(\alpha), \exp(\beta, 2)))), \\
Dx(\ln(\alpha)) & \rightarrow \div(Dx(\alpha), \alpha), \\
Dx(\exp(\alpha, \beta)) & \rightarrow \\
& \quad \text{plus}(\times(\beta, \times(\exp(\alpha, \neg(\beta, 1)), Dx(\alpha))), \\
& \quad \times(\exp(\alpha, \beta), \times(\ln(\alpha), Dx(\beta)))); \\
\end{align*}
\]

functions: one, zero, x, a, Dx, plus, times, minus, neg, div, exp, two, ln;

polynomials:
\[
\begin{align*}
\mu &= 4, \\
\text{zero} &= 4, \\
\text{one} &= 4, \\
\text{two} &= 4, \\
a &= 4, \\
x &= 4, \\
Dx(\alpha) &= \alpha^2, \\
\text{plus}(\alpha, \beta) &= \alpha + \beta, \\
\times(\alpha, \beta) &= \alpha + \beta, \\
\min(\alpha, \beta) &= \alpha + \beta, \\
\div(\alpha, \beta) &= \alpha + \beta, \\
\exp(\alpha, \beta) &= \alpha + \beta, \\
\ln(\alpha) &= 1 + \alpha, \\
\neg(\alpha) &= 1 + \alpha. \\
\end{align*}
\]

Symbolic Differentiation (long version)
(Knuth, 73), (Dershowitz, 87, p. 79)

A.16 Groups
\[
\begin{align*}
\times(e, x) & \rightarrow x, \\
\times(i(x), x) & \rightarrow e, \\
\times(\times(x, y), z) & \rightarrow \times(x, \times(y, z)), \\
\div(x, y) & \rightarrow \times(x, i(y)); \\
\end{align*}
\]

functions: times, e, i, div;
polynomials:
mu = ν,
times(x, y) = ν + ν x + ν y + ν x y,
div(x, y) = ν + ν x + ν y + ν x y²,
e = ν,
i(x) = ν + ν x².

Groups.
(Huet, 80), (Ben Cherifa & Lescanne, 87, p. 151)

A.17 Taussky Group

times(x, times(y, z)) -> times(times(x, y), z),
times(one, one) -> one,
times(x, i(x)) -> one,
i(times(x, y)) -> times(i(y), i(x)),
g(times(x, y), y) -> f(times(x, y), x),
f(one, y) -> y;

times, one, i, g, f;

polynomials:
mu = 2,
one = 2,
times(x, y) = ν + ν x + ν y + ν x y,
f(x, y) = x + y,
g(x, y) = ν + ν x + ν y² + ν x y²,
i(x) = ν + ν x².

Taussky Group
(Knuth & Bendix, 70)
(Ben Cherifa & Lescanne, 87, p. 155),
(Steinbach 91, Example 8.4)

A.18 Fibonacci Group

comb(a, b) -> c,
comb(b, c) -> d,
a -> comb(d, e),
b -> comb(e, a),
e -> comb(c, d);

functions: a, b, c, d, e, comb;

polynomials:
mu = 0,
a = v,
b = v,
c = v,
d = v,
e = v,
comb(x, y) = x + y.

Fibonacci Group
(Steinbach 91, Example 8.2)

A.19 Negative Coefficient

square(succ(succ(x))) -> f(succ(succ(x))),
f(succ(x)) -> plus(x, square(x)),
square(succ(x)) -> plus(double(x), square(x)),
succ(double(x)) -> plus(x, x);

functions: square, succ, f, plus, double;

polynomials:
mu = 1,
square(x) = x^2,
succ(x) = x + 1,
f(x) = x^2 - x + 1,
plus(x, y) = x + y,
double(x) = 2 x.
References


