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`.

```
> (start)
```

Then the system prints out the following start message.

```
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 **H**elp) the current menu of possible commands will be printed.

```
>> 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`.

```
>> L
Enter a filename:
```

Accompanying the system there is a directory `examples` of example files. One of the easiest example in these files is `Nested_Function_Symbols`.

```
Enter a filename:  Nested_Function_Symbols

>>
```

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 "->", 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.

```
f(f(x)) -> g(g(x)),
g(g(f(x))) -> f(g(g(x)));


functions:  f, g.


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

Nested Function Symbols
(Steinbach 91, Example 8.1)
```

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

2.2 Proving Termination

By entering `H` again you will notice that the menu of possible command has changed.

```
>> H

Termination proof with
[1] polynomials of degree 1
[S] simple-mixed polynomials
[V] provided values
[G] given polynomials
```

```
[L] Load new trs (current one is
Nested_Function_Symbols)
[M] Change method for generating polynomial orderings
[Q] Quit
[H] Help

>>
```

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, `mu` 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.

```
>> 1


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

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
mu 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
[L] Load new trs (current one is
Nested_Function_Symbols)
[M] 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.

```
>> A

Begin of the Termination Proof

Run time:  0.02 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
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 (`X1^2` stands for x_1^2). 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.

```
>> S


Approach:
mu is mapped to the polynomial V0
f(X1) is mapped to the polynomial V1 + V2 X1 + V3 X1^2
g(X1) is mapped to the polynomial V4 + V5 X1 + V6 X1^2


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 μ 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(X1) is mapped to the polynomial 1 + 2 X1
f(X1) is mapped to the polynomial 2 + 2 X1
mu 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.

```
>> G

Enter a polynomial for mu :  2
Enter a polynomial for f(X1) :  V X1 + X1^3
Enter a polynomial for g(X1) :  V + X1 + X1^3

Approach:
mu is mapped to the polynomial 2
f(X1) is mapped to the polynomial V0 X1 + X1^3
g(X1) is mapped to the polynomial V1 + X1 + X1^3
```

```
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 "$f(x_1, \ldots, x_n) =$ *polynomial*". Here f must be a function symbol, the names $x_1, \ldots, x_n$ 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(x, 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.

```
=> H

[T] Use Testing
[C] Use modified Collins algorithm
[H] Help
[B] Back

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.

```
=> T
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.

```
=> C
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.

File	Approach	Method	Time
`Nested_Function_Symbols` [Ste91, Example 8.1]	`1`	`TESTING` (2 1 0), 20 sec.	0.03 sec.
`Nested_Function_Symbols` [Ste91, Example 8.1]	`S`	`TESTING` (2 1 0), 20 sec.	0.9 sec.
`Nested_Function_Symbols` [Ste91, Example 8.1]	`1`	`COLLINS` 40 sec., 1/3, ¬neg., rat.	19.8 sec.
`Flatten`	`1`	`TESTING` (2 1 0), 20 sec.	0.04 sec.
`Flatten`	`S`	`TESTING` (2 1 0), 20 sec.	0.8 sec.
`Flatten`	`V` ($\mu = 2$, nil $= 2$)	`COLLINS` 40 sec., 1/3, ¬neg., rat.	20.4 sec.
`Stack` [DJ90, p. 253]	`1`	`TESTING` (2 1 0), 20 sec.	0.1 sec.
`Boolean_Ring_1` [HD83], [Der87, p. 102]	`S`	`TESTING` (2 1 0), 20 sec.	0.3 sec.
`Boolean_Ring_2` [Hsi82], [BL87, p. 152]	`S`	`TESTING` (2 1 0), 20 sec.	0.4 sec.
`Neutral_Elements_Distributivity` [Pau84], [Ste91, Example 8.13]	`S`	`TESTING` (2 1 0), 20 sec.	0.4 sec.
`Plus`	`S`	`TESTING` (2 1 0), 20 sec.	0.6 sec.
`Endomorphism_Associativity` [Bel84], [BL87]	`V` ($\mu = 2$, map$(x) = v + vx$)	`TESTING` (2 1 0), 20 sec.	0.1 sec.
`Endomorphism_Associativity` [Bel84], [BL87]	`S`	`TESTING` (2 1 0), 20 sec.	0.8 sec.
`Endomorphism_Associativity` [Bel84], [BL87]	`V` ($\mu = 2$)	`COLLINS` 40 sec., 1/3, ¬neg., rat.	35.5 sec.
`Distributivity_Associativity` [Der87, p. 78]	`S`	`TESTING` (2 1 0), 20 sec.	1.9 sec.
`Distributivity_Associativity` [Der87, p. 78]	`V` ($\mu = 2$, plus$(x, y) = v + vx + vy$)	`COLLINS` 40 sec., 1/3, ¬neg., ¬rat.	25.1 sec.

File	Approach	Method	Time
`Binomial_Coefficients` [Ste91, Example 8.8], [Ste92, Example 13]	`S`	`TESTING` `(2 1 0)`, 20 sec.	1.6 sec.
`Running_Example` [Mid89], [Ste91, Example 6.1]	`F`	`TESTING` `(2 1 0)`, 20 sec.	0.2 sec.
`Boolean_Ring_3` [Ste91, Example 8.5]	`F`	`TESTING` `(2 1 0)`, 20 sec.	2.6 sec.
`Reverse` [Ste91, Example 8.6]	`F`	`TESTING` `(2 1 0)`, 20 sec.	1.7 sec.
`Symbolic_Differentiation` [Knu73], [Der87, p. 79]	`F`	`TESTING` `(2 1 0)`, 20 sec.	3.4 sec.
`Symbolic_Differentiation` [Knu73], [Der87, p. 79]	`F`	`TESTING` `(1 0)`, 20 sec.	0.6 sec.
`Symbolic_Differentiation_long` [Knu73], [Der87, p. 79]	`F`	`TESTING` `(2 1 0)`, 20 sec.	0.2 sec.
`Groups` [Hue80], [BL87, p. 151]	`F`	`TESTING` `(2 1 0)`, 20 sec.	0.9 sec.
`Taussky_Group` [KB70], [BL87, p. 155], [Ste91, Example 8.4]	`F`	`TESTING` `(2 1 0)`, 20 sec.	5.5 sec.
`Fibonacci_Group` [Ste91, Example 8.2]	`V` $(\mu = 0,$ $\mathsf{comb}(x, y) = vx + vy)$	`TESTING` `(4 2 1 0)`, 20 sec.	4.8 sec.
`Fibonacci_Group` [Ste91, Example 8.2]	`F`	`TESTING` `(4 2 1 0)`, 20 sec.	0.3 sec.
`Fibonacci_Group` [Ste91, Example 8.2]	`F`	`TESTING` `(4 3 2 1 0)`, 20 sec.	0.8 sec.
`Fibonacci_Group` [Ste91, Example 8.2]	`F`	`COLLINS` 40 sec., 1/3, ¬neg., rat.	8.1 sec.
`Negative_Coefficient`	`F`	`TESTING` `(2 1 0)`, 20 sec.	0.01 sec.
`Negative_Coefficient`	`V` $(\mathsf{plus}(x, y) = x + y)$	`TESTING` `(2 1 0)`, 20 sec.	7.3 sec.

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

```
f(f(x)) -> g(g(x)),
g(g(f(x))) -> f(g(g(x)));

functions:  f, g.

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

Nested Function Symbols
(Steinbach 91, Example 8.1)
```

A.2 Flatten

```
flatten(nil) -> nil,
flatten(cons(nil, y)) -> cons(nil, flatten(y)),
flatten(cons(cons(u, v), w)) -> flatten(cons(u, cons(v, w)));

functions:  flatten, cons, nil.

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

Flatten
```

A.3 Stack

```
top(push(x, y)) -> x,
pop(push(x, y)) -> y,
alternate(empty, z) -> z,
alternate(push(x, y), z) -> push(x, alternate(z, y));

functions:  top, push, pop, alternate, 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(x, F) -> x,
xor(x, neg(x)) -> F,
and(x, 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.

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

Plus
```

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.

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

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

A.9 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.

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

Binomial Coefficients
(Steinbach 91, Example 8.8,
Steinbach92, Example 13)
```

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:
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.
```

A.12 Boolean_Ring_3

```
impl(x, y) -> xor(and(x, y), xor(x, true)),
or(x, y) -> xor(and(x, y), xor(x, y)),
equiv(x, y) -> xor(x, xor(y, true)),
neg(x) -> xor(x, true);

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

polynomials:
mu = 1,
true = 1,
impl(x, y) = v + v x + v y,
equiv(x, y) = v + v x + v y,
or(x, y) = v + v x + v y,
neg(x) = v + v x,
and(x, y) = v + v x + v y,
xor(x, y) = v + v x + v y.

======================================================================
Boolean_Ring_3
(Steinbach 91, Example 8.5)
```

A.13 Reverse

```
append(nil, y) -> y,
append(add(x, y), z) -> add(x, append(y, z)),
append(append(x, y), z) -> append(x, append(y, z)),
reverse(nil) -> nil,
reverse(add(x, y)) -> append(reverse(y), add(x, nil)),
reviter(nil, y) -> y,
reviter(add(x, y), z) -> reviter(y, add(x, z)),
append(reverse(x), y) -> reviter(x, y),
reverse(x) -> reviter(x, nil);

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

polynomials:
mu = 2,
nil = 2,
reverse(x) = v + v x^2,
add(x, y) = v + v x + v y + v x y,
reviter(x, y) = v x + v x y,
```

```
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)),
```

```
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)),
Dx(div(alpha, beta)) ->
minus(div(Dx(alpha), beta), times(alpha, div(Dx(beta), exp(beta, two)))),
Dx(ln(alpha)) -> div(Dx(alpha), alpha),
Dx(exp(alpha, beta)) ->
       plus(times(beta, times(exp(alpha, minus(beta, one)), Dx(alpha))),
                  times(exp(alpha, beta), times(ln(alpha), Dx(beta))));

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

polynomials:
mu = 4,
zero = 4,
one = 4,
two = 4,
a = 4,
x = 4,
Dx(alpha) = alpha^2,
plus(alpha, beta) = alpha + beta,
times(alpha, beta) = alpha + beta,
minus(alpha, beta) = alpha + beta,
div(alpha, beta) = alpha + beta,
exp(alpha, beta) = alpha + beta,
ln(alpha) = 1 + alpha,
neg(alpha) = 1 + alpha.

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

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

A.16 Groups

```
times(e, x) -> x,
times(i(x), x) -> e,
times (times(x, y), z) -> times(x, times(y, z)),
div(x, y) -> times(x, i(y));

functions:  times, e, i, div;
```

```
polynomials:
mu = v,
times(x, y) = v + v x + v y + v x y,
div(x, y) = v + v x + v y + v x y^2,
e = v,
i(x) = v + v x^2.

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

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;

functions:  times, one, i, g, f;

polynomials:
mu = 2,
one = 2,
times(x, y) = v + v x + v y + v x y,
f(x, y) = x + y,
g(x, y) = v + v x + v y^2 + v x y^2,
i(x) = v + v x^2.

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

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

[ACM84] D. S. Arnon, G. E. Collins & S. McCallum. Cylindrical Algebraic Decomposition I. *SIAM Journal of Computing*, 13(4): 865-877, 1984.

[Bel84] F. Bellegarde. Rewriting Systems on FP Expressions that reduce the Number of Sequences they yield. *Symposium on LISP and Functional Programming*, ACM, Austin, TX, 1984.

[BL87] A. Ben Cherifa & P. Lescanne. Termination of Rewriting Systems by Polynomial Interpretations and its Implementation. *Science of Computer Programming*, 9(2):137-159, 1987.

[Col75] G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proc. 2nd GI Conf. on Automata Theory and Formal Languages*, Kaiserslautern, Germany, 1975.

[Der87] N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1, 2):69-115, 1987.

[DJ90] N. Dershowitz & J.-P. Jouannaud. Rewrite Systems. *Handbook of Theoretical Comp. Science*, J. van Leuwen, Ed., vol. B, ch. 6, pp. 243-320, Elsevier, 1990.

[Gie95a] J. Giesl. Generating Polynomial Orderings for Termination Proofs. In *Proc. 6th Int. Conf. Rewriting Techniques and Applications*, Kaiserslautern, Germany, 1995.

[Gie95b] J. Giesl. Generating Polynomial Orderings for Termination Proofs (Extended Version). Technical Report IBN 95/23, Technische Hochschule Darmstadt, Germany. Available by anonymous ftp from `kirmes.inferenzsysteme.informatik.th-darmstadt.de` under `pub/termination`.

[Hue80] G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Journal of the ACM* 27(4):797-821, 1980.

[Hsi82] J. Hsiang. Topics in Automated Theorem Proving and Program Generation, PhD Thesis, University of Illinois, 1982.

[HD83] J. Hsiang & N. Dershowitz. Rewrite Methods for Clausal and Non-Clausal Theorem Proving. In *Proc. 10th EATCS Int. Colloquium on Automata, Languages and Programming*, Barcelona, Spain, 1983.

[KB70] D. E. Knuth & P. B. Bendix. Simple Word Problems in Universal Algebras. *Computational Problems in Abstract Algebra*, J. Leech, ed., Pergamon Press, pp. 263-297, 1970.

[Knu73] D. E. Knuth. Fundamental Algorithms. In *The Art of Computer Programming*, vol. 1, 2nd edn. Reading, MA. Addison-Wesley. 1973.

[Mid89] A. Middeldorp. A Sufficient Condition for the Termination of the Direct Sum of Term Rewriting Systems. In *Proc. 4th Annual Symposium on Logic in Computer Science*, Pacific Grove, CA, 1989.

[Pau84] E. Paul. Proof by Induction in Equational Theories with Relations between Constructors. In *Proc. 9th Colloquium on Trees in Algebra and Programming*, Bordeaux, France, 1984.

[Ste91] J. Steinbach. Termination Proofs of Rewriting Systems — Heuristics for Generating Polynomial Orderings. SEKI-Report SR-91-14, Univ. Kaiserslautern, Germany, 1991.

[Ste92] J. Steinbach. Proving Polynomials Positive. In *Proc. 12th Conf. on Foundations of Software Technology and Theoretical Computer Science*, New Delhi, India, 1992.