Term Ordering

Prover9's term ordering procedures and options are simpler than Otter's, but somewhat less flexible. We recommend that those who use Otter's "ad hoc" ordering try Prover9's KBO ordering.

Prover9 has available several methods for comparing terms. (Although atomic formulas, literals, and clauses are not, strictly speaking, terms, the term orderings we write about here apply to those objects as well.)

The term orderings are partial (and sometimes total on ground terms), and they are used in two ways.

- To orient equalities (positive and negative). If one side of the equality is greater than the other, the greater side is placed on the left-hand side, and the equality is marked as oriented.
- To decide which literals in clauses are admissible for application of inference rules. Several of the resolution and paramodulation rules require that some of the literals be maximal in their clause.

For many problems, a good term ordering can determine the difference between success and failure. The default settings work well in many cases, but many difficult problems require adjustments to the term ordering.

The primary choice (via parameter **`order`**) is type of ordering: LPO, RPO, or KBO. Each of those types uses a symbol precedence (see the `function_order` and `predicate_order` commands), and KBO also uses a symbol weighting function (see the `list(kbo_weights)` command). In addition, the options **`eq_defs`** and **`inverse_order`** cause changes to the term ordering.

See [Dershowitz-termination] for a survey on term ordering.

The Primary Choice

The *symbol precedence* is is a total order on function and predicate symbols (including constants). The symbol weighting function maps symbols to nonnegative integers.

- LPO (Lexicographic Path Ordering). The term ordering is determined entirely by the symbol precedence. It is total on ground terms.
- RPO (Recursive Path Ordering). The term ordering is determined entirely by the symbol precedence. It is not necessarily total on ground terms, because the order of subterms is not considered.
- KBO (Knuth-Bendix Ordering). This ordering uses a weighting function on symbols as well as the symbol precedence. The weighting function is used first, and the symbol

precedence breaks ties. It is total on ground terms.

KBO is perhaps the most natural of the three, because it it based on weights of symbols, but it is more cumbersome to specify because it is determined both by the symbol weights and by the symbol precedence. However, if one of the two terms being compared has more occurrences of a variable, it cannot be smaller. For example, the distributivity equation cannot be oriented so that it distributes (expands) terms.

LPO is perhaps the most powerful of the three, because it can usually orient more equations. However, it allows rewrite rules that expand terms in explosive ways, for example (this is from a real problem),

`(x * y) * z` rewrites to `E * (x * (E * (x * (E * (y * (E * (x * (E * (x * (E * z))))))))))`

RPO is perhaps the least useful of the three, because it is not necessarily total on ground terms. That is, not all ground equations can be oriented. Also, see the sections on demodulation options and on inference rules.

The reasonable choice is usually between LPO (the default) and KBO. For many problems, either one is good. The main reason LPO is the the default is that it is a bit faster than KBO.

Here is the primary option.

```
assign(order, string).  % default string=lpo, range [lpo,rpo,kbo]
```

This option is used to select the primary term ordering to be used for orienting equalities and for determining maximal literals in clauses. The choices are `lpo` (Lexicographic Path Ordering), `rpo` (Recursive Path Ordering), and `kbo` (Knuth-Bendix Ordering).

Termination of Demodulation

If each member of a set of demodulators (rewrite rules) is oriented with respect to the current ordering (LPO, RPO, or KBO), then demodulation (term rewriting) is guaranteed to terminate (in theory) on all terms, regardless of the the order in which the demodulators are applied or the order in which the subject terms are demodulated. However, there are sets of demodulators that are intractable in practice.

The Default Term Ordering

The default symbol precedence (for LPO, RPO, and KBO) is given by the following rules (in order).

- function symbols < equality symbol < non-equality predicate symbols;
- if the term ordering is KBO, and if there is exactly one unary function in the problem, that function is greater than all other functions;
- function symbols: arity-0 < arity-2 < arity-1 < arity-3 < arity-4 ... (note the position of arity-1);
- non-equality predicate symbols: lower arity < higher arity;
- non-Skolem symbols < Skolem symbols;
- for Skolem symbols, the lower index is the lesser;
- for non-Skolem symbols, more occurrences < fewer occurrences;

- the lexical ASCII ordering (UNIX strcmp() function).

The specific symbol precedence for a problem is given in the output file in the section `PROCESS INPUT`.

The default symbol-weighting function for KBO is given by the following rules.

- Variables have weight 1;
- if there is exactly one unary function in the problem, it has weight 0;
- all other symbols have weight 1.

Adjustments to the Term Ordering

The Symbol Precedence

The `function_order` and `predicate_order` commands can be used to assign a symbol precedence. (The `lex` command is synonym for `function_order`.) They contain lists of symbols ordered by increasing precedence. For example,

```
predicate_order([=, <=, P, Q]).          % = < P < Q
function_order([a, b, c, +, *, h, g]).   % a < b < c < + < * < h < g
```

There are two separate commands, because presecate symbols are always greater than function symbols.

If there are function or predicate symbols in the problem that do not appear in the corresponding command, a warning is issued, and Prover9 will complete the precedence inserting the missing symbols at the beginning of the precedence using its default rules. In these cases, the user should check that Prover9 has constructed a reasonable precedence.

Note that Skolem symbols cannot appear in a function_order command, because Skolem symbols do not exist at the time the function_order command is written. If there is a function_order command, and if Skolem symbols are generated, each one will be inserted, in effect, into the function_order command at a position just before the first symbol of higher arity. This method gives a symbol precedence similar to the default in many cases.

Otter's lex command has a syntax that shows the arities of the symbols; Prover9's function_order and predicate_order commands list only the symbols. The arities are not necessary for Prover9, because a string cannot represent two symbols with different arities.

The KBO Weights

If the term ordering is KBO, `assign(`**order**`, kbo)`, the user can change the default symbol-weighting function. For example,

```
list(kbo_weights).
  a = 3.
  b = 2.
  * = 5.
  j = 22.
end_of_list.
```

(This has no relationship to the term-weighting function for selecting the given clause and discarding inferred clauses.)

If any symbols are absent from the list, they retain their default KBO weights of 1. The symbol weights must be greater than 0, with the exception that there may be one unary symbol of weight 0. (The definition of KBO allows for one unary symbol of weight 0 which must also be greatest in the precedence. This special case allows an such as `g(f(x,y)) = f(g(y),g(x))` to be oriented as shown and used as a rewrite rule.)

Term Ordering Options

```
set(inverse_order).     % default set
clear(inverse_order).
```

If this flag is set, if there is no function_order command (which defines the function symbol precedence), and if the term ordering is LPO or RPO, then Prover9 will attempt to adjust the default symbol precedence if there are any input equations that specify an inverse operation. For example, if `f(x,g(x)) = c` is input, `g` will be placed after `f` in the precedence. This allows an equation such as `g(f(x,y)) = f(g(y),g(x))` to be oriented as shown for demodulation and paramodulation. If this flag is set, the `PROCESS INPUT` section of the output file shows how the flag changes the symbol precedence.

```
assign(eq_defs, string).  % default string=unfold, range [unfold,fold,pas
```

If *string*=unfold, and if the input contains an equational definition, say `j(x,y) = f(f(x,x),f(y,y))`, the defined symbol `j` will be eliminated from the problem before the search starts. This procedure works by adjusting the symbol precedence so that the defining equation becomes a demodulator. If there is more than one equational definition, cycles are avoided by choosing a cycle-free subset of the definitions. If the primary term ordering is KBO, this option may admit demodulators that do not satisfy the KBO ordering, because a variable may have more variables on the right-hand side. However, this exception is safe (does not cause non-termination).

If *string*=fold, and if the input contains an equational definition, say `j(x,y) = f(f(x,x),f(y,y))`, the term ordering will be adjusted so that equation is flipped and becomes a demodulator which introduces the defined symbol whenever possible during the search.

If *string*=pass, nothing special happens. In this case, functions may still be unfolded or folded if the term ordering and symbol precedence happen to arrange the demodulators to do so.

Next Section: More Prep