Prolog Summary

November 29, 2006

1 Language

1.1 Values

Every "value" in Prolog is one of the following:

Atom Atoms are words starting with a lowercase letter: `a`, `ross`, `true`, `one2THREE`, `pi`, `[]`

Number Numbers are expressed in the usual decimal form: `1`, `-42`, `2.71`

Structure Structures have a name, not starting with an uppercase letter or a digit, and fields: `point(5,10)`, `tree(0,tree(-5,[],[]),[])`

Variable Variables are words starting with an uppercase letter: `X`, `Length`, `_`

`_` is used to denote a nameless variable, typically meaning "I don't care what this value is."

1.1.1 Lists

Lists can be expressed in many ways in Prolog. Prolog provides a few standard ways and some shorthands for writing them.
`[]` is an atom denoting the empty list.
`[Head | Tail]` denotes the list whose first element is `Head` and following elements are those in the list `Tail`.
`[x,y,z]` is a shorthand for `[x|[y|[z|[]]]]`.
Lists are stored as the structure `.(Head,Tail)`. *Note that '.' is the name of the structure.*

1.1.2 Operators

Operators are shorthands for structures used to make the code more readable. They have precedence, usually following the standard precedences for arithmetic. Parentheses can be used when precedence does not group the operands as desired.

Some common arithmetic operators are, from highest to lowest precedence:

`-` The usual negation

`**`, `^` Denotes X^Y

`*`, `/`, `//`, `mod` The usual multiplication, division, integer division, and integer remainder

`+`, `-` The usual addition and subtraction

For example, `(X+-Y)**Z` is shorthand for `**(+(X,-(Y)),Z)`.
There are more operators described later.

1.2 Goals

Goals are structures or atoms with a deeper meaning, namely pass or fail. For example, `X = Y` is shorthand for the structure `=(X,Y)` and has the meaning that it passes if and only if `X` and `Y` are structurally equivalent. Thus, `2+1 = 1+2` fails since, although `+` is the same, one has a `2` as the first field for `+` while the other has `1`. They may express the same value, but there are other goals that compare values rather than structures.

1.2.1 Value Comparisons

The following operators are used to compare values. In order to compute a value, there cannot be any unassigned variables in the expression. The previously specified arithmetic operators detail how to compute the value, as described above.

`is` Evaluates the term on the right, then unifies with the left

`=:=` The values on each side are equivalent

`=\=` The values on each side are not equivalent

`<` The usual less-than comparator

`=<` The usual less-than-or-equal comparator

`>` The usual greater-than comparator

`>=` The usual greater-than-or-equal comparator

1.2.2 Goal Operators

Goal operators are possibly the most significant operators in Prolog. They combine goals to produce a new goal. Here are the important ones from highest to lowest precedence:

`not`, `\+` Fails if the operand goal passes, otherwise passes (uses a cut, see below)

`,` Passes if and only if both operand goals pass

`;` Passes if either goal passes, first attempting the left operand goal

1.2.3 Special Goals

There are a few goals built in to Prolog that are important to understand:

`true` The goal that always passes

`fail` The goal that always fails

`repeat` Always passes but, when backtracked to, will pass again

`!` The elusive cut essential to mastering and optimizing Prolog. This goal always passes but cannot be backtracked through. Also, if passed and the parent goal passes or fails, prevents reattempting the parent goal

1.3 Rules

A program in Prolog is simply a database of rules. Rules are what define whether a goal passes or not. There can be multiple rules for one goal. In attempting a goal, the rules are applied from top to bottom. If in applying a rule a cut is reached, then no more rules will be attempted. A goal passes if any attempted rule passes. A goal can pass multiple times (when backtracking) if a rule can be applied multiple times or multiple rules can be applied. Rules take two forms:

goal-structure`.` Passes if the goal being attempted fits the goal-structure.

goal-structure `:-` ***goal***`.` Passes if the goal being attempted fits the goal-structure and the following goal also passes. Remember the following goal can and often does include goal operators.

2 Built-In Predicates

Almost every implementation of Prolog comes with built-in predicates. The following are a few of the more powerful ones.

2.1 `bagof(Vars, Goal, Insts)`

`bagof` will be true if element i of `Insts` is the value of `Vars` on answer i of `Goal`.
For example, `element([a,b,c],E).` will first answer with `E = a`, then with `E = b`, then lastly with `E = c`. Thus, `bagof(E, element([a,b,c],E), I)` will answer with (and only with) `I = [a,b,c]`, since then $\texttt{I}_1$ = `a` (value of `E` after first answer of `element([a,b,c],E).`), $\texttt{I}_2$ = `b` (value of `E` after second answer of `element([a,b,c],E).`), and $\texttt{I}_3$ = `c` (value of `E` after third answer of `element([a,b,c],E).`). Since `element([a,b,c],E).` only answers 3 times, `I` only has 3 elements.

`bagof` can have multiple variables in `vars`: `bagof(X/Y, element([[a,b],[a,c],[b,c]],[X,Y]), [a/b,a/c,b/c]).` Say, however, you want only the left side of the pair, so essentially you want to ignore `Y`. Then do `bagof(X, Y^ element([[a,b],[a,c],[b,c]],[X,Y]), [a,a,b]).` The `Y^` tells `bagof` to ignore the `Y`. If you leave out `Y^`, then `bagof` will answer twice. First with `Insts = [a]` and `Y = b`, then with `Insts = [a,b]` and `Y = c`.

2.2 `setof(Vars, Goal, Insts)`

`setof` is essentially `bagof` but `Insts` for `setof` is the sorted set list version of `Insts` for `bagof`, using an unrestrictive literal ordering (not described here) rather than a number or string ordering.

For example:
`setof(E, element([a,c,b,a],E), I).`
answers with `I = [a,b,c]` while
`bagof(E, element([a,c,b,a],E), I).` answers with `I = [a,c,b,a]`.