Chapter 16

Logic Programming

Input and Output

- The build-in predicate **`read`** is used for reading terms from the current input.
- The goal **`read(X)`** will cause the next term, `T`, to be read, and this term will be matched with `X`.
  - If `X` is a variable then `X` will be instantiated to `T`.
  - If matching does not succeed the the goal fails.

Input and Output

  - This predicate is deterministic, so in the case of failure there will be no backtracking to input another term.

- The build-in predicate **`write`** is used for writing terms to the current output.
  - This predicate 'knows' to display any term no matter how complicated it may be.

Constructing and Decomposing Atoms

- An atom can be converted to a sequence of characters using the build-in predicate **`name`**.
  - This predicate relates atoms and their ASCII codes.
  - `name(zx232,[122,120,50,51,50]).`
- Two typical uses:
  1. Given an atom, break it down into single characters.
  2. Given a list of characters, combine them into an atom

Testing the Type of Terms

- Sometimes it is useful to know what is the type of some value.
- Example: if we want to add the values of two variables `X` and `Y` by: **`Z is X + Y.`**
  - Before this goal is executed, `X` and `Y` have to be instantiated to integers.
- The build-in predicate `integer(X)` is true if `X` is an integer or if it is a variable whose value is an integer.
  - `X` must 'currently stand for' an integer.

Testing the Type of Terms

- **`var(X)`** succeeds if `X` is currently an uninstantiated variable.
- **`nonvar(X)`** succeeds if `X` is a term other than a variable, or `X` is an already instantiated variable.
- **`atom(X)`** is true if `X` currently stands for an atom.
- **`atomic(X)`** is true if `X` currently stands for an integer or an atom.

Testing the Type of Terms

- **`compound(X)`** succeeds if `X` is a compound term (a structure, including lists but not []).
- **`number(X)`** succeeds if `X` is a number (integer of floating-point).
- **`float(X)`** succeeds if `X` is a floating-point number.

Constructing and Decomposing Terms

- There are three build-in predicates for decomposing terms and constructing new terms.
  - **`Term=..L`** is true if `L` is a list that contains the principal functor of `Term`, followed by its arguments.
  - **`functor(Term,F,N)`** is true if `F` is the principal functor of `Term` and `N` is the arity of F.
  - **`arg(N,Term,A)`** is true if `A` is the `Nth` argument in `Term`, assuming that arguments are numbered from left to right starting with 1.

Finding all Solutions to a Query

- Prolog can generate, by backtracking, all the objects, one by one, that satisfy some goal.
  - Each time a new solution is generated, the previous one disappears and is not accessible any more.
  - Sometime we would prefer to have all generated objects available together.

Finding all Solutions to a Query

- **`findall(T,G,L)`** find each solution to `G`; instantiates variables to `T` to the values that they have in that solution; and adds that instantiation of `T` to `L`.
- **`bagof(T,G,L)`** like `findall` except for its treatment of the free variables of `G` (those that do not occur in `T`).

Finding all Solutions to a Query

  - Whereas `findall` would try all possible values of all variables, `bagof` will pick the first set for the free variables that succeeds, and use only that set of values when finding the solution in `L`.
  - If you ask for an alternative solution to `bagof`, you will get the results of trying another set of values for the free variables.
- **`setof(T,G,L)`** like `bagof` but the elements of `L` are sorted into alphabetical order and duplicates are removed.