Structure Inspection and Meta-logical Predicates in Prolog

Michael McThrow

San Jose State University
Computer Science Department
CS 152 – Programming Paradigms

November 2, 2020

Table of Contents

Table of Contents

Type Predicates

A type predicate is a unary predicate that is able to determine the type of a term.

Type Predicates

A type predicate is a unary predicate that is able to determine the type of a term.

Type predicates in Prolog have an equivalent in Scheme (e.g., `number?`, `pair?`, `symbol?`, etc).

integer

`integer` determines whether a parameter is an integer.

integer

`integer` determines whether a parameter is an integer.

Examples:

- `integer(5)?` ⇒ `true.`
- `integer(-3)?` ⇒ `true.`
- `integer(x)?` ⇒ `false.`

`atom`

`atom` determines whether the parameter is an atom (i.e., a lowercase symbol in Scheme terminology).

atom

`atom` determines whether the parameter is an atom (i.e., a lowercase symbol in Scheme terminology).

Examples:

- `atom(bulbasaur)?` $\Rightarrow$ `true.`
- `atom(5)?` $\Rightarrow$ `false.`

compound

compound determines whether the parameter has the form of a relation with one or more arguments.

compound

compound determines whether the parameter has the form of a relation with one or more arguments.

- `compound(evolution(bulbasaur,ivysaur))?` $\Rightarrow$ `true.`
- `compound(s(0))?` $\Rightarrow$ `true.`
- `compound(bulbasaur)?` $\Rightarrow$ `false.`

number

`number` determines whether the parameter is a number. The number can be floating-point as well as an integer.

number

`number` determines whether the parameter is a number. The number can be floating-point as well as an integer.

- `number(5)?` ⇒ `true.`
- `number(5.1)?` ⇒ `true.`
- `number(-5.1)?` ⇒ `true.`
- `number(x)?` ⇒ `false.`

atomic

`atomic` determines whether the parameter is either an atom or a number.

atomic

`atomic` determines whether the parameter is either an atom or a number.

- `atomic(5)?` $\Rightarrow$ `true.`
- `atomic(-3.14159)?` $\Rightarrow$ `true.`
- `atomic(x)?` $\Rightarrow$ `true.`
- `atomic(s(0))?` $\Rightarrow$ `false.`

Time for a demo in SWI-Prolog.

What about for variables? Are there any built-in type predicates that detect whether a term is a variable?

Yes, and we will explore them later during this lecture.

Table of Contents

Recall that a compound term is one that has the form of a relation with one or more arguments.

Recall that a compound term is one that has the form of a relation with one or more arguments.

Examples:

- `evolution(bulbasaur,ivysaur).`
- `plus(s(0),s(s(0)),s(s(s(0))))`.
- `s(0).`

Suppose we want to obtain information about a compound term, such as its arguments or its arity (i.e., number of arguments).

Suppose we want to obtain information about a compound term, such as its arguments or its arity (i.e., number of arguments). We can obtain this information by using two built-in relations: `functor/3` and `arg/3`.

functor

`functor(Term,Name,Arity)` is a relation that accepts a compound term `Term` and checks to see if the term's name matches with `Name` and if the arity of the term is equal to `Arity`.

functor

`functor(Term,Name,Arity)` is a relation that accepts a compound term `Term` and checks to see if the term's name matches with `Name` and if the arity of the term is equal to `Arity`.

Example:

- `functor(evolution(eevee,jolteon),evolution,2)?` $\Rightarrow$ `true.`

arg

`arg(N,Term,Arg)` checks the compound term `Term` to see if its `N`th argument is equal to `Arg`. Note that `N` starts at 1.

arg

`arg(N,Term,Arg)` checks the compound term `Term` to see if its `N`th argument is equal to `Arg`. Note that `N` starts at 1.

Example:

- `arg(1,evolution(eevee,jolteon),eevee)?` ⇒ `true.`
- `arg(2,evolution(eevee,jolteon),eevee)?` ⇒ `false.`

Time for another SWI-Prolog demo.

Table of Contents

Meta-logical predicates are predicates that "sit above" the logic programming system. They are used for exercising control over the execution of logic programs.

So, back to our earlier question? Are there any built-in type predicates that detect whether a term is a variable?

So, back to our earlier question? Are there any built-in type predicates that detect whether a term is a variable?

Yes.

var and nonvar

- `var(Term)` checks if `Term` is a variable.
- `nonvar(Term)` checks if `Term` is *not* a variable.

Example

Here is a new version of `plus(X,Y,Z)` that uses `nonvar`:

```
plus(X,Y,Z) :- nonvar(X),nonvar(Y),Z is X+Y.
plus(X,Y,Z) :- nonvar(X),nonvar(Z),Y is Z-X.
plus(X,Y,Z) :- nonvar(Y),nonvar(Z),X is Z-Y.
```

Example

Here is a new version of `plus(X,Y,Z)` that uses `nonvar`:

```
plus(X,Y,Z) :- nonvar(X),nonvar(Y),Z is X+Y.
plus(X,Y,Z) :- nonvar(X),nonvar(Z),Y is Z-X.
plus(X,Y,Z) :- nonvar(Y),nonvar(Z),X is Z-Y.
```

Compared to the last `plus` example from previous lectures, this has restored some query functionality; we can now run queries such as `plus(X,6,10)?`. Note that performing queries with two variables is still not supported in this above definition, but we have the tools to extend the definition to support such queries (and this will be one of your lab exercises).

Example

Here is a new version of `plus(X,Y,Z)` that uses `nonvar`:

```
plus(X,Y,Z) :- nonvar(X),nonvar(Y),Z is X+Y.
plus(X,Y,Z) :- nonvar(X),nonvar(Z),Y is Z-X.
plus(X,Y,Z) :- nonvar(Y),nonvar(Z),X is Z-Y.
```

The uses of `nonvar` that are placed at the initial parts of the bodies of the above clauses are examples of *meta-logical tests*. Meta-logical tests decide which clause in a procedure should be used [Sterling and Shapiro].

The == predicate checks to see if `X` and `Y` are identical. \== checks to see if `X` and `Y` are not identical.

`freeze` and `melt`

The relation `freeze(Term,Frozen)` makes a copy of `Frozen` and makes it ground (i.e., treats it as if it had no variables). The relation `melt(Frozen,Thawed).` "unfreezes" `Frozen` and makes it un-ground. Note that the textbook describes a `melt_new` relation, but it is not defined in SWI-Prolog.

Logical Disjunction in Prolog

- We can perform logical conjunction (i.e, AND) by using commas in Prolog rules.
- For logical disjunction, we use the semicolon:
  - Example: `(X ; Y).` means `X` OR `Y`.