Prolog: The typeless language?

"Like Lisp, Prolog is a declaration-free, typeless language."
[Chapter 6, page 125, *The Art of Prolog*]

Prolog: The typeless language?

"Like Lisp, Prolog is a declaration-free, typeless language."
[Chapter 6, page 125, *The Art of Prolog*]

Consider the following built-in predicates:

var(X) succeeds if X is currently an uninstantiated variable
nonvar(X) succeeds if X is not a variable, or X is an already instantiated variable
atom(X) is true if X currently stands for an atom
integer(X) is true if X currently stands for an integer
float(X) is true if X currently stands for a real number
number(X) is true if X currently stands for a number
atomic(X) is true if X currently stands for a number or an atom
compound(X) is true if X currently stands for a compound term (a structure)

Prolog: The typeless language?

If Prolog is "typeless", what's with the predicates that test for type?

Consider the following built-in predicates:

var(X) succeeds if X is currently an uninstantiated variable
nonvar(X) succeeds if X is not a variable, or X is an already instantiated variable
atom(X) is true if X currently stands for an atom
integer(X) is true if X currently stands for an integer
float(X) is true if X currently stands for a real number
number(X) is true if X currently stands for a number
atomic(X) is true if X currently stands for a number or an atom
compound(X) is true if X currently stands for a compound term (a structure)

Prolog: The typeless language?

"Like Lisp, Prolog is a declaration-free, typeless language." [Chapter 6, page 125, *The Art of Prolog*]

If you can really talk about data types in the context of a programming language which doesn't have variables in the traditional sense, then Prolog, like Lisp, is dynamically typed. This means type checking is done at execution time, so the programmer doesn't have to declare data types up front. This may lead to the illusion of the language being "typeless", but rest assured that these languages keep track of type.

Prolog: The typeless language?

"Like Lisp, Prolog is a declaration-free, typeless language." [Chapter 6, page 125, *The Art of Prolog*]

But again, since Prolog doesn't really have variables, some folks argue that Prolog is not a typed language. If you wanted to make that argument, however, you wouldn't preface it with "Like Lisp..." because Lisp is both strongly typed (according to one of the many definitions of "strongly typed") and dynamically typed.

Another built-in predicate

When Prolog solves your query, it looks for the first instance of the query that it can derive from the program. If there's another instance of the query that's implied by the program, Prolog will tell you about it if you ask (by typing ";").

But what if you're interested in collecting at once all the instances of the query that can be derived from the program? Given that Prolog loses all information about the first instance once it backtracks to find a second instance, is this even possible?

Another built-in predicate

Well of course it is! If there was a way to tell Prolog in advance that you want all the instances that satisfy a query, Prolog could use something like `assert` to keep track of every solution as it finds it, no?

One of the ways you tell Prolog to collect all the solutions is through the use of the `findall` predicate.

Another built-in predicate

Well of course it is! If there was a way to tell Prolog in advance that you want all the instances that satisfy a query, Prolog could use something like `assert` to keep track of every solution as it finds it, no?

One of the ways you tell Prolog to collect all the solutions is through the use of the `findall` predicate.

```
?- findall(X, Q, L).
```

will produce the list L of all the objects X that satisfy the query Q. For example...

Another built-in predicate

In the context of the Flintstone Family Tree, findall works like this:

To find all the kids who have Bamm-Bamm as their father:

```
?- findall(X,father(bamm-bamm,X),L).
L = [roxy, chip]
```

Another built-in predicate

In the context of the Flintstone Family Tree, findall works like this:

To find all the kids who have a father:

```
?- findall(Y,father(X,Y),L).
L = [pebbles, bamm-bamm, roxy, chip]
```

Another built-in predicate

In the context of the Flintstone Family Tree, findall works like this:

To find all the fathers of any kids:

```
?- findall(X,father(X,Y),L).
L = [fred, barney, bamm-bamm, bamm-bamm]
```

Note that Bamm-Bamm shows up twice. Why? He is the father of two children.

Another built-in predicate

In the context of the Flintstone Family Tree, findall works like this:

To find all the fathers of any kids:

```
?- findall(X,father(X,Y),L).
L = [fred, barney, bamm-bamm, bamm-bamm]
```

Note that Bamm-Bamm shows up twice. Why? He is the father of two children.

`bagof` and `setof` are related predicates. Read Chapter 16.1 for more about `findall`. Read the SWI-Prolog manual (4.29) for more about `bagof` and `setof`.