Prolog Types

The term is Prolog’s basic data structure.

Everything is expressed in the form of a term. This includes programs and data.

Prolog’s term is similar to Scheme’s S-Expression.

Prolog has four basic types of terms in Prolog:
1. **variables** start with an uppercase letter;
2. **compound terms** are lists, strings, and structures;
3. **atoms** start with a lower-case letter (they are similar to Scheme’s symbols);
4. **numbers**.

Prolog Types...

Prolog Numbers

Most Prolog implementations support infinite precision integers. **This is not true of GNU Prolog!**

The built-in operator `is` evaluates arithmetic expressions:

```
| ?- X is 6*7.  
  X = 42
| ?- X is 6.0*7.0.  
  X = 42.0
| ?- X is 60000000000000*70000000000000.  
  X = 1
```
Prolog Arithmetic Expressions

An infix expression is just shorthand for a structure:

| ?- X = +(1, *(2, 3)).
X = 1+2*3
| ?- X = 1+2*3.
X = 1+2*3
| ?- X is +(1, *(2, 3)).
X = 7
| ?- X is 1+2*3.
X = 7

\[X = 1\ast 2\] means “make the variable X and 1*2 the same”. It looks like an assignment, but it’s what we call unification. More about that later.

Prolog Atoms

Atoms are similar to Scheme’s symbols.

Atoms start with a lower-case letter and can contain letters, digits, and underscore (\_).

| ?- X = hello.
X = hello
| ?- X = hE_l_l_o99.
X = hE_l_l_o99

Prolog Variables

Variables start out uninstantiated, i.e. without a value.

Uninstantiated variables are written _number:

| ?- write(X).
_16

Once a Prolog variable has been instantiated (given a value), it will keep that value.

| ?- X=sally.
X = sally
| ?- X=sally, X=lisa.
no

Prolog Variables...

When a program backtracks over a variable instantiation, the variable again becomes uninstantiated.

| ?- (X=sally; X=lisa), write(X), nl.
sally
X = sally ? ;

lisa
X = lisa
A Prolog program consists of a database of **facts** and **rules**:

- `likes(lisa,chocolate).`  
- `likes(lisa,X) :- tastes_like_chocolate(X).`
- `:- is read if.  
- `:- is just an operator, like other Prolog operators. The following are equivalent:

  - `likes(lisa,X) :- boy(X),tastes_like_choc(X).`

  - `(likes(lisa,X),
    (boy(X),tastes_like_chok(X))).`

Prolog facts/rules can be **overloaded**, wrt their **arity**.
You can have a both a rule `foo()` and a rule `foo(X)`:

- `?- [user].
  foo.
  foo(hello).
  foo(bar,world).
  foo(X,Y,Z) :-
    Z is X + Y.
  <ctrl-D>`

- `?- foo.
  yes
  |- ?- foo(X).
    X = hello
  |- ?- foo(X,Y).
    X = bar
    Y = world
  |- ?- foo(1,2,Z).
    Z = 3`

**Standard predicates**

- `read(X)` and `write(X)` read and write Prolog terms.  
- `nl` prints a newline character.
- `write` can write arbitrary Prolog terms:

  - `?- write(hello(world)),nl.
    hello(world) uncaught exception: error`

  - `Note that read(X) requires the input to be syntactically correct and to end with a period:`
  
  - `?- read(X).
    foo).
    uncaught exception: error`
Unification/Matching

- The operator tries to make its left and right-hand sides the same.
- This is called unification or matching.
- If Prolog can’t make $x$ and $y$ the same in $x = y$, matching will fail.

```prolog
| ?- X=lisa, Y=sally, X = Y.
  no
| ?- X=lisa, Y=lisa, Z = X, Z = Y.
  X = lisa
  Y = lisa
  Z = lisa
```

- We will talk about this much more later.

Backtracking

- Prolog will try every possible way to satisfy a query.
- Prolog explores the search space by using backtracking, which means undoing previous computations, and exploring a different search path.

```prolog
| ?- [user].
girl(sally).
girl(lisa).
pretty(lisa).
blonde(sally).
| ?- girl(X),pretty(X).
  X = lisa
| ?- girl(X),pretty(X),blonde(X).
  no
| ?- (X=lisa; X=sally), pretty(X).
  X = lisa
```

- We will talk about this much more later.

Māori Family Relationships

John Foster (in He Whakamaarama – A New Course in Māori) writes:

Relationship is very important to the Māori. Social seniority is claimed by those able to trace their whakapapa or genealogy in the most direct way to illustrious ancestors. Rights to shares in land and entitlement to speak on the marae may also depend on relationship. Because of this, there are special words to indicate elder or younger relations, or senior or younger branches of a family.

- Māori is the indigenous language spoken in New Zealand. It is a polynesian language, and closely related to the language spoken in Hawaii.
### Māori Terms of Address

<table>
<thead>
<tr>
<th>Māori</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>au</td>
<td>I</td>
</tr>
<tr>
<td>tipuna, tupuna</td>
<td>grandfather, grandmother, grandparent, ancestor</td>
</tr>
<tr>
<td>tipuna</td>
<td>grandparents</td>
</tr>
<tr>
<td>matua taane</td>
<td>father</td>
</tr>
<tr>
<td>maatua</td>
<td>parents</td>
</tr>
<tr>
<td>paapaa</td>
<td>father</td>
</tr>
<tr>
<td>whaea, maamaa</td>
<td>mother</td>
</tr>
<tr>
<td>whaea kee</td>
<td>aunt</td>
</tr>
<tr>
<td>kuia</td>
<td>grandmother, old lady</td>
</tr>
<tr>
<td>tuakana</td>
<td>older brother of a man, older sister of a woman</td>
</tr>
<tr>
<td>teina</td>
<td>younger brother of a man, younger sister of a woman</td>
</tr>
</tbody>
</table>

*Literally: "What you follow at night"

### The Whanau

A program to translate between English and Māori must take into account the differences in terms of address between the two languages.

Write a Prolog predicate `calls(X, Y, Z)` which, given a database of family relationships, returns all the words that X can use to address or talk about Y.

```prolog
?- calls(aanaru, hata, Z).
Z = tuakana ;
Z = maataamua ;
no

?- calls(aanaru, rapeta, Z).
Z = teina ;
no
```
The Whanau...  

Whanau is Māori for family.

Below is a table showing an extended Māori family.

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Father</th>
<th>Mother</th>
<th>Spouse</th>
<th>Born</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoone</td>
<td>male</td>
<td>unknown</td>
<td>unknown</td>
<td>Rita</td>
<td>1910</td>
</tr>
<tr>
<td>Rita</td>
<td>female</td>
<td>unknown</td>
<td>unknown</td>
<td>Hone</td>
<td>1915</td>
</tr>
<tr>
<td>Ranginui</td>
<td>male</td>
<td>unknown</td>
<td>unknown</td>
<td>Reremoana</td>
<td>1915</td>
</tr>
<tr>
<td>Reremoana</td>
<td>female</td>
<td>unknown</td>
<td>unknown</td>
<td>Ranginui</td>
<td>1916</td>
</tr>
<tr>
<td>Rewi</td>
<td>male</td>
<td>Hoone</td>
<td>Rita</td>
<td>Reremoana</td>
<td>1935</td>
</tr>
<tr>
<td>Rahia</td>
<td>female</td>
<td>Ranginui</td>
<td>Reremoana</td>
<td>Rewi</td>
<td>1940</td>
</tr>
<tr>
<td>Hata</td>
<td>male</td>
<td>Rewi</td>
<td>Rahia</td>
<td>none</td>
<td>1957</td>
</tr>
<tr>
<td>Kiri</td>
<td>female</td>
<td>Rewi</td>
<td>Rahia</td>
<td>none</td>
<td>1959</td>
</tr>
</tbody>
</table>

---

The Whanau Program — Database Facts

We start by encoding the family as facts in the Prolog database.

% person(name, sex, father, mother, spouse, birth-year).

person(hoone, male, unkn1, unkn5, rita, 1910).
person(rita, female, unkn2, unkn6, hoone, 1915).
person(ranginui, male, unkn3, unkn7, reremoana, 1915).
person(reremoana, female, unkn4, unkn8, ranginui, 1916).
person(rewi, male, hoone, rita, reremoana, 1935).
person(rahia, female, ranginui, reremoana, rita, 1916).
person(hata, male, rewii, rahia, none, 1957).
person(kiri, female, rewii, rahia, none, 1959).

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Father</th>
<th>Mother</th>
<th>Spouse</th>
<th>Born</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hiniera</td>
<td>female</td>
<td>Rewi</td>
<td>Rahia</td>
<td>Pita</td>
<td>1960</td>
</tr>
<tr>
<td>Aanaru</td>
<td>male</td>
<td>Rewi</td>
<td>Rahia</td>
<td>none</td>
<td>1962</td>
</tr>
<tr>
<td>Rapeta</td>
<td>male</td>
<td>Rewi</td>
<td>Rahia</td>
<td>none</td>
<td>1964</td>
</tr>
<tr>
<td>Mere</td>
<td>female</td>
<td>Rewi</td>
<td>Rahia</td>
<td>none</td>
<td>1965</td>
</tr>
<tr>
<td>Pita</td>
<td>male</td>
<td>unknown</td>
<td>unknown</td>
<td>Hiniera</td>
<td>1960</td>
</tr>
<tr>
<td>Moeraa</td>
<td>female</td>
<td>Pita</td>
<td>Hiniera</td>
<td>none</td>
<td>1986</td>
</tr>
<tr>
<td>Huia</td>
<td>female</td>
<td>Pita</td>
<td>Hiniera</td>
<td>none</td>
<td>1987</td>
</tr>
<tr>
<td>Irihaapeti</td>
<td>female</td>
<td>Pita</td>
<td>Hiniera</td>
<td>none</td>
<td>1988</td>
</tr>
</tbody>
</table>
We introduce some auxiliary predicates to extract information from the database.

% Auxiliary predicates

gender(X, G) :- person(X, G, _, _, _, _).
othergender(male, female).
othergender(female, male).
female(X) :- gender(X, female).
male(X) :- gender(X, male).

We next write some predicates that computes common family relationships.

% Is Y the <operator> of X?
wife(X, Y) :- person(X, male, _, _, Y, _).
husband(X, Y) :- person(X, female, _, _, Y, _).
spouse(X, Y) :- wife(X, Y).
spouse(X, Y) :- husband(X, Y).

% Is Y a sibling of X of the gender G?
sibling(X, Y, G) :- <left as an exercise>.

% Is Y one of X’s older/sibling of gender G?

% Is Y a sibling of X of his/her opposite gender?
siblingofothersex(X, Y) :- <left as an exercise>.

% Is Y an ancestor of X of gender G?
ancestor(X,Y,G) :- <left as an exercise>.

% Is Y an older relative of X of gender G?
olderrelative(X,Y,G) :-
    ancestor(X, Y, G).
    ancestor(X, Z, _),
    sibling(Y, Z, G).

% Is Y one of X’s older/sibling of either gender?

% Is Y one of X’s older/younger siblings of either gender?

Some of the following are left as an exercise:

% Is X older than Y?
older(X,Y) :-
    person(X, _, _, _, _, Xyear),
    person(Y, _, _, _, _, Yyear),
    Yyear > Xyear.

% Is Y an ancestor of X of gender G?
ancestor(X,Y,G) :- <left as an exercise>.

% Is Y an older relative of X of gender G?
olderrelative(X,Y,G) :-
    ancestor(X, Y, G).
    ancestor(X, Z, _),
    sibling(Y, Z, G).

% Is Y a sibling of X of his/her opposite gender?
siblingofothersex(X, Y) :- <left as an exercise>.
We can now finally write the predicate \texttt{calls(X,Y,T)} which computes all the ways \texttt{T} in which \texttt{X} can address \texttt{Y}.

\begin{verbatim}
% Me.
calls(X, X, au).

% Parents.
calls(X,Y,paapaa) :- person(X, _,Y, _, _, _).
calls(X,Y,maamaa) :- person(X, _, _,Y, _, _).

% Oldest/youngest sibling of same sex.
calls(X, Y, tuakana) :-
    gender(X, G),
    eldestsibling(X, Y, G).
calls(X, Y, teina) :-
    gender(X, G),
    youngestsibling(X, Y, G).
\end{verbatim}

\begin{verbatim}
% Siblings of other sex.
calls(X, Y, tungaane) :- <left as an exercise>.
calls(X, Y, tuahine) :- <left as an exercise>.
calls(X, Y, tipuna) :- <left as an exercise>.

% Sons and daughters.
calls(X, Y, tama) :- <left as an exercise>.
calls(X, Y, tamahine) :- <left as an exercise>.

% Oldest/youngest child.
calls(X, Y, maataamua) :- <left as an exercise>.
calls(X, Y, pootiki) :- <left as an exercise>.

% Child-in-law.
calls(X, Y, hunaonga) :- <left as an exercise>.

% Grandchild.
calls(X, Y, mokopuna) :- <left as an exercise>.
\end{verbatim}

Read Clocksin-Mellish, Chapter 2.