Logic and Functional Programming

Labwork 10

April 20, 2021

1 SWI-Prolog: useful commands

A program is read via the `File->Consult...` menu option of SWI-Prolog.

- `halt` – to exit the interpreter.
- To abort a long-running computation:
  - Mac OS X: press `Cmd+C` followed by `a`
  - Windows: press Ctrl+C ...
- `listing.` – displays all clauses from the knowledge base.
- `listing(`p`).` – displays the clauses from the the knowledge base which define predicate p. Similarly, `listing(`$[p_1, \ldots, p_n]$`).` displays the clauses from the the knowledge base which define the predicate $p_1, \ldots, p_n$.
- After we obtained an answer to a query, we can press either
  `.` to stop searching other answers, or
  `;` to resume the search of another answer.
- `trace.` to turn on the interactive tracing of every step of the computation.

2 Comparison and arithmetic operators

In SWI-Prolog, the comparison operators for numeric expressions are implemented as predefined predicates. Their names are `<, >, =, =<, >=`.

Note that we write `A =< B` instead of `A<=B` to check if $\mathtt{A} \leq \mathtt{B}$ when `A, B` are numbers. For example:

```
?- 1 > 2.          ?- 4.5=<5.5.
false.             true.
```

SWI-Prolog has the following built-in arithmetic operators:

E1 + E2	addition
E1 − E2	subtraction
E1 * E2	division
E1 / E2	division
E1 // E	integer division
E1 `div` E2	quotient of integer division
E1 `rem` E2	remainder of integer division
E1 ** E2	raising to a power
E1 /\ E2	bitwise AND
E1 \/E2	bitwise OR
E1 ^ E2	bitwise XOR
E1 << E2	shift of bits to the left
E1 >>E2	shift of bits to the right

Important remark! By default, Racket does not evaluate arithmetic expressions. For example:

```
? X=1+2.    % instantiates  X with the unevaluated expression
X = 1+2.
? 1+2=3+4. % the unevaluated expressions look different, are not unifiable.
false.
```

We can enforce the evaluation of arithmetic expressions in two ways:

1. With the predefined operator `is`.

   `X is` E.

   This query succeeds in the following two cases:

   (a) If `X` is uninstantiated. In this case, `X` gets instantiated with the numeric value of E.

   (b) If `X` has a numeric value which coincides with that of E.

   For example:

   ```
   ?- X is 4512 // 100.      ?- 45 = 4512 // 100.
   X = 45.                   true.
   ```

2. With the boolean operators

E1 =:= E2	checks if the numeric values of E1 and E2 are the same.
E1 =\= E2	checks if the numeric values of E1 and E2 are different.

For example:

```
?- 2*3 =:= 5+1.     ?- 7-1 =\= 1+2.
true.               true.
```

3 Warmup exercises

1. Consider the following logic program:

```
% thief(X) expresses the fact that X is thief
thief(bob).
% likes(X,Y) expresses the fact that X likes Y
likes(mary,candies).
likes(mary,wine).
likes(bob,X) :- likes(X,wine).
% may_steal(X,Y) expresses the fact that X may steal Y
may_steal(X,Y) :- thief(X), likes(X,Y).
```

   (a) Write a query for the question "What may Bob steal?". Without running Prolog, indicate all answers to this query that can be deduced from the given program.

   (b) Use Prolog to verify if your answers were correct.

2. Assume the following relations have already been defined in a program:

   - `father(X,Y)` to indicate that `X` is the father of `Y`
   - `mother(X,Y)` to indicate that `X` is the mother of `Y`
   - `man(X)` to indicate that `X` is a man
   - `woman(X)` to indicate that `X` is a woman

   Extend this program with definitions of the following relations:

   (a) `parent(X,Y)` to indicate that `X` is a parent of `Y`

   (b) `isFather(X)` to indicate that `X` is a father

   (c) `isMother(X)` to indicate that `X` is a mother

   (d) `sister(X,Y)` to indicate that `Y` is the sister of `X`

   (e) `grandpa(X,Y)` to indicate that `X` is the grandpa of `Y`

3. Consider the problem of finding all elements which appear in two given lists, by defining a predicate `member_both(X,L1,L2)` to hold if `X` is both an element of list `L1` and list `L2`.

4. Consider the problem of defining the relation `neighbor(X,Y)` for the fact that `X` is neighbor of `Y`. This relation is assumed to be symmetric: if `X` is neighbor of `Y`, then `Y` is neighbor of `X`.

   (a) How would you encode the following knowledge base: "Alan is neighbor of Bob. Bob is neighbor of Caleb. Caleb is neighbor of Dan and Dick. Dan is neighbor of Erin."

   (b) Write a query for the question "Who are the neighbors of Dan?" What answers will you get?

5. Define by induction on the structure of list `L1` the predicate `app(L1,L2,L)` which holds if `L` is the result of appending lists `L1` and `L2`.
   (a) What is the meaning of the query `app(L1,L2,[1,2,3,4])`? What answers will you get?
   (b) What is the meaning of the query `app(L,_,[1,2,3,4])`? What answers will you get?
   (c) Use `app` to define the predicate `sublist(S,L)` to hold if `S` is a sublist of list `L`. For example, `[1,2]` and `[2,3]` are sublists of `[1,2,3,4,5]`, but `[2,4]` is not sublist of `[1,2,3,4,5]`.

6. Consider the problem of arranging three 1's, three 2's, ..., three 9's in sequence so that for all $1 \leq i \leq 9$ there are exactly i numbers between successive occurrences of i. Use Prolog to define the relation `niceList(L)` for lists which have this property.

   SUGGESTION. Note that `L` is such a nice list if it has 27 elements and the following property: for all $1 \leq i \leq i$, `L` contains a sublist of the form

   $$[i, \underbrace{\_, \ldots, \_}_{i \text{ times}}, i, \underbrace{\_, \ldots, \_}_{i \text{ times}}, i]$$

   Use the definition of `sublist(S,L)` from the previous exercise to express this property.

7. Consider the program defined by

```
part([X],[],[],[]).
part(X,[H|T],[H|L],R) :- H<X,part(X,T,L,R).
part(X,[H|T],L,[H|R]) :- H>=X,part(X,T,L,R).
```

   (a) Use SWI-Prolog to compute the answers to the queries

```
?-part(4,[1,7,3,5],L,R).
?-part(6,[10,1,3,7,5,9,20],L,R).
```

   (b) When `X` is a number. `Lst` a list of numbers, and `L`, `R` two uninstantiated variables, what will be the answer to the query

```
?- part(X,Lst,L,R).
```

4 Unification: exercises

Check which of the following terms have an mgu. For those which are unifiable, indicate an mgu:

1. `f(X,Y,Z)` and `f(a,Z,h(a))`
2. `f(g(X),g(c),Y)` and `f(g(g(Y)),X,a)`
3. `f(h(b),X,X,Y)` and `f(h(b),g(Y),g(g(Z)),g(a))`

Suggestion: use the Martelli-Montanari algorithm.