605.629: Programming Languages
Assignment 13
Sabbir Ahmed

December 11, 2021

1. . [20 pts, Prolog]

Consider the database in the lecture and write `grandparent` and `grandchild` relationships. Define `grandchild` in terms of `grandparent` and state the pattern about them. Show some examples.

Answer

```
grandparent(X, Z) :-
  parent(X, Y),
  parent(Y, Z).
```

The predicate does not distinguish between the gender of the child and therefore does not utilize the `male(X)` or `female(X)` predicates. The following is a sample output of the predicate:

```
X = fred,
Z = gary
X = fred,
Z = john
X = fred,
Z = bruce
X = inger,
Z = gary
X = inger,
Z = john
```

```
grandchild(X, Z) :-
  grandparent(Z, X).
```

The predicate does not distinguish between the gender of the child like `grandparent(X, Z)` and therefore simply utilizes the predicate in reverse. The following is a sample output of the predicate:

```
X = gary,
Z = fred
X = john,
Z = fred
X = bruce,
Z = fred
X = gary,
Z = inger
```

```
X = john,
Z = inger
```

---

2. [20 pts, Prolog]

Write the `reverse` predicate for lists in Prolog using the `append` predicate. What is the complexity of this reverse predicate?

Answer

The `append` predicate is defined as follows:

```
append([], Y, Y).
append([H | T1], L2, [H | T3]) :- append(T1, L2, T3).
```

The predicate can be used to reverse a list recursively by recursively appending an empty list backwards. The base case of the predicate is as follows:

```
reverse([], []). % empty list
```

The recursive case would get the tail element of the list and append it to the new list and pass it to itself as follows:

```
reverse([H | T], R) :- reverse(T, R2), append(R2, [H], R).
```

The complete predicate:

```
reverse([], []).
reverse([H | T], R) :- reverse(T, R2), append(R2, [H], R).
```

Since the recursive predicate utilizes the recursive `append` predicate, `reverse` has an exponential complexity.

---

3. [20 pts, Prolog]

Write a `length` predicate that computes the length of a list.

Answer

The `length` predicate can be computed recursively. The base case is as follows:

```
length([], 0).
```

The recursive case increments the length variable and returns the tail of the list to itself recursively. The predicate is as follows:

```
length([], 0).
length([_ | L], N) :-
    length(L, M),
    N is M + 1.
```

---

4. [40 pts, Prolog]

Consider the example grammar and given sentence in the lecture. Complete the grammar predicates `predicate`, `determiner`, `verb` and `noun`.

Compute `verb(X, Y)`.
Compute `noun(X, Y)`.

Answer

The example sentence graph:

```
the(1, 2).
professor(2, 3).
discovered(3, 4).
a(4, 5).
group(5, 6).
period(6, 7).
```

The grammar predicates:

```
sentence(K, L) :- subject(K, M), predicate(M, N), period(N, L).
subject(K, L) :- determiner(K, M), noun(M, L).
predicate(K, L) :- verb(K, M), determiner(M, N), noun(N, L).

determiner(K, L) :- a(K, L); the(K, L).
noun(K, L) :- professor(K, L); group(K, L).
verb(K, L) :- discovered(K, L).
```

The following is the output from the predicate: `verb(X, Y)`

```
X = 3,
Y = 4
```

The following is the output from the predicate: `noun(X, Y)`

```
X = 2,
Y = 3
X = 5,
Y = 6
```
