Deerwalk Institute Of Technology

Lab 4: Introduction – Prolog List

(Artificial Intelligence)

Submitted by:
Name: Shiva Tripathi
Roll No: 532
Batch: 2019
Section: A

Submitted to:

Birodh Rijal

1 Introduction - Prolog List

Practical V - Lists Submission Deadline: TBA

So far we have only considered simple items as arguments to our programs. However in Prolog a very common data-structure is the list. Lists themselves have the following syntax. They always start and end with square brackets, and each of the items they contain is separated by a comma. For example: [first,second,third] is a list with three items.
Prolog also has a special facility to split the first part of the list (called the head) away from the rest of the list (known as the tail). We can place a special symbol *j* (pronounced 'bar') in the list to distinguish between the first item in the list and the remaining list. For example, consider the following.
[*first; second; third*] = [*AjB*]
where *A* = *first* and *B* = [*second; third*]
The unification here succeeds. A is bound to the first item in the list, and B to the remaining list.
Now lets consider some comparisons of lists:

ˆ [*a; b; c*] unifies with [*HeadjT ail*] resulting in *Head* = *a* and *T ail* = [*b; c*]
ˆ [*a*] unifies with [*HjT*] resulting in *H* = *a* and *T* = []
ˆ [*a; b; c*] unifies with [*ajT*] resulting in *T* = [*b; c*]
ˆ [*a; b; c*] doesn't unify with [*bjT*]
ˆ [] doesn't unify with [*HjT*]
ˆ [] unifies with []. Two empty lists always match

2 Some Operations on Lists

Type in the following prolog code and try to figure out how they are working.

i. Write list
writelist([]) : -*nl:*
writelist([*HjT*]) : -*write*(*H*)*; nl; writelist*(*T*)*:*

```
?- writelist([ai,daa,cn,crypto,sm]).

ai

daa

cn

crypto

sm

true.
```

ii. Membership

member(*X;* [*XjList*])*:*

member(*X;* [*ElementjList*]) : -*member*(*X; List*)*:*

```
?- member(ai,[ai,daa]).

true.


?- member(python,[ai,daa]).

false.
```

iii. Concatenation

conc([]*; L; L*)*:*

conc([*XjL*1]*; L*2*;* [*XjL*3]) : -*conc*(*L*1*; L*2*; L*3)*:*

```
?- conc([ai,daa],[cn,crypto,sm],Concatenat).

Concatenat = [ai, daa, cn, crypto, sm].
```

iv. Take the n-th element

take(1*;* [*Hj*]*; H*)*:*

take(*N;* [*jT*]*; X*) : -*N*1 is *N* - 1*; take*(*N*1*; T; X*)*:*

```
?- take(1,[ai,daa,cn],ai).

true ;

false.


?- take(2,[ai,daa,cn],ai).

false
```

v. Length of a list

length([]*;* 0)*:*

length([*HjT*]*; N*) : -*length*(*T; M*)*; N* is *M* + 1*:*

```
?- length([ai,daa,cn],3).

true.
```

```
?- length([ai,daa,cn],2).

false.
```

vi. Sum of elements

```
sum([]; 0):
sum([XjL]; Sum) : -sum(L; SL); Sum is X + SL:
```

```
?- sum([2,4,8],14).

true.
```

```
?- sum([2,4,8],64).

false.
```

vii. Reverse of a list

```
reverse([]; X; X):
reverse([XjY ]; Z; W ) : -reverse(Y; [XjZ]; W ):
```

```
?- reverse([a,b,c],Rev).

Rev = [c, b, a].
```

viii. Append

```
append([]; L; L):
append([HjT ]; L; [HjT L]) : -append(T; L; T L):
```

```
?- append([ai,daa],[cn],[ai,daa,cn]).

true.
```

```
?- append([ai,daa],[cn],[ai,daa,cn,python]).

false.
```

3 DFA with input as a list

```
% Code snippet begin
t(0,a,1). t(0,b,2).
t(1,a,1). t(1,b,1).
t(2,a,2). t(2,b,2).
startstate(0). % 0 is a starting state
finalstate(1). % 1 is a final state
% Code snippet end
```

In the code, the predicate t(0,a,1) denotes a transition from state ’0’ to state ’1’ on input ’a’. The start state has the label ’0’ and the end state (final state) is labelled ’1’. Implement a predicate checkinput(Start,Input) that checks if a word (here, input) given as a list (e.g. [a,b,b,a,b]) is accepted by the DFA starting from a start state (here State).

```
?- checkinput(1,[a]).

true .
```

```
?- checkinput(0,[a]).

true ;

false.
```

4 Using Structures

Going back to the family tree lab, we will focus on a better way to represent structured data. We first define a family/3 predicate to store three components of a family: father, mother and children. For example:

```
family(
person(homer,simpson,date(7,may,1960),works(inspector,6000)),
person(marge,simpson,date(7,may,1965),housewife),
[ person(bart,simpson,date(7,may,1967),student),
person(lisa,simpson,date(7,may,1965),student) ].
```

Using the family predicate, implement the following relation as rules:

A. husband(X) : true if X is someone’s husband

```
?-husband(homer).
```

```
 true
```

B. wife(X) : true if X is someone's wife

```
?-wife(merge).
 true
```

C. child(X) : true if X is someone's child

```
?-child(bart),
 true
```

D. exists(Person) : true if the person is in the database

```
?-exists(bart).
 true


?-exists(lisa).
 true
```