LIN 177 Homework 3

Hardy Jones
999397426
Professor Ojeda
Winter 2015

1. `?- findall(X, phone(X), Y).`
   ```
   Y =  [p,b,m,t,d,n,k,g,ŋ,f,v,θ,ð,s,z,ʃ,ʒ,č,ǰ,l,ɹ,j,w,h,i,ɪ,e,æ,u,ʊ,o,a,ə,ʌ].
   ```

2. (b) Yes.

   The left side can be desugared a bit to read:

   `[piro, quechua, german | []]`

   (c) No.

   The left side is a list with 4 elements and the right side is a list with 3 elements.

   (d) Yes.

   It unifies the following variables:

   `Head = french, Head1 = german, Head2 = english`

   (e) No.

   The left side is a list with 2 elements and the right side is a list with 3 elements.

   (f) Yes.

   Clearly the variable `Head` unifies with itself. Then the other unification takes place:

   `[Head1 | Tail] = Tail1`

   (g) No.

   The left side is the empty list, while the right side has at least one element.

   (h) Yes.

   These are the same lists, the left side has more sugar than the right side.

   (i) No.

   These are different lists. The left side is a list with a single element that is the list containing `english`. The right side is a list with a single element `english`

   (j) Yes. with the following instantiations:

   `[english, french] = Head, [spanish] = Tail`

   (k) Yes.

   It has the following instantiations:

   `Item = quechua, Item1 = english, [piro] = Tail`

(l) Yes.
It has the following instantiations
`french = Head, [[quechua, german]] = Tail`

3. For the given definition of "the sublist relation", this predicate holds. However, this predicate does not properly detect that one list is a sublist of the other in the normal sense of the word, as `sublist([1,1,1], [1])` should fail. However, that example passes the predicate.

   The `sublist` fact states that the empty list is a sublist of any list.

   This fact is true, as the empty list is a sublist of itself, and all other lists are made by cons-ing onto another list that is eventually the empty list.

   The `sublist` rule states that for any cons-ed list, if the first element of the list is an element of some list `List` and the rest of the list is a sublist of `List`, then the cons-ed list is a sublist of `List`.

   This rule is true as it ensures each element of the first list is a member in the second list. The recursive call within the rule is always working with a list containing exactly one less element (in particular the head element) so we are guaranteed to check each and every element of the first list.

4. The `member` rule states that an `Element` is a member of a `List` if we can append some `List1` to the front of another list where `Element` is the head, and we get out the `List`.

   There are three cases we can consider (though the first and third are special cases of the second.)

   - In the case where `Element` is the head of `List`, we unify `List1 = []` and we unify `Tail` with the tail of `List`
   - When `Element` is in the middle of `List` at some index i, we unify `List1` with the list consisting of the first $i-1$ elements of `List` and `Tail` with the last $n-i$ elements of `List`–where n is the length of `List`.
   - When `Element` is the last element of `List`, we unify `List1` with the list consisting of all elements of `List` except the last, and we unify `List1` with the empty list.

   From these cases, we can see that the predicate `member` describes "the member relation".