Introduction to CL I (684.01) OSU Linguistics
Detmar Meurers Winter 2007

Exercise sheet 2

(Submit as a plain text email message to dm@ling.osu.edu before class on Thursday, Jan. 18)

Provide Prolog definitions for the following relations. Define them using the minimal means necessary—in particular, there is no need to make use of other relations defined in class or predefined predicates.
Thoroughly test your predicates before handing them in!

1. `next_to_last/2`: a two place relation which takes a list as first argument and returns the next to last element of that list (if there is one) as second argument; i.e.,
   `last(+List,−Next-to-Last-List-element)`

   Example queries:

   ```
   ?- next_to_last([a,b,c,d]),X). ⇒ X=c
   ?- next_to_last([a,b,c],X). ⇒ X=b
   ?- next_to_last([],X). ⇒ no
   ```

2. `wrap_with_f/2`: a two place relation which takes a list and returns the same list with the functor f wrapped around every element; i.e.,
   `wrap_with_f(+List,−List-With-f-Wrapped-Elements)`

   Example queries:

   ```
   ?- wrap_with_f([a,b,c,d],X). ⇒ [f(a),f(b),f(c),f(d)]
   ?- wrap_with_f([],X). ⇒ X=[]
   ```

3. `delete_b/2`: a two place relation which takes a list and deletes one occurrence of `b` (if there is one); i.e., `delete_b(+List,−List-with-one-b-less)`

   Example queries:

   ```
   ?- delete_b([b,e,b,d],X). ⇒ X=[e,b,d]; X=[b,e,d]
   ?- delete_b([e,b,c,b,g,h],X). ⇒ X=[e,c,b,g,h]; X=[e,b,c,g,h]
   ?- delete_b([e,g,b],X). ⇒ X=[e,g]
   ?- delete_b([e,g,b,b],X). ⇒ X=[e,g,b]; X=[e,g,b]
   ?- delete_b([e,c],X). ⇒ no
   ```

4. Define the relation `delete_one_b` which is just like `delete_b` except that it removes only the first occurrence of a `b`.

   ```
   ?- delete_one_b([b,e,b,d],X). ⇒ X=[e,b,d]
   ?- delete_one_b([e,b,c,b,g,h],X). ⇒ X=[e,c,b,g,h]
   ?- delete_one_b([e,g,b],X). ⇒ X=[e,g]
   ?- delete_one_b([e,g,b,b],X). ⇒ X=[e,g,b]
   ?- delete_one_b([e,c],X). ⇒ no
   ```

5. `in_list/2`: a two place relation which succeeds if the first list is a sublist (with sublist being reflexive) of the second; i.e., `in_list(+`*`Sublist`*`,+`*`List`*`)`

   Example queries:

   ```
   ?- in_list([b,c],[a,b,c,d]).  ⇒ yes
   ?- in_list([b,c],[b,c]).  ⇒ yes
   ?- in_list([b,c],[a,b,b,c,d]).  ⇒ yes
   ?- in_list([a,b],[a,b,c,d]).  ⇒yes
   ?- in_list([a,b,c],[a,b,c,d]).⇒ yes
   ?- in_list([a],[a,b,c,d]).  ⇒ yes
   ?- in_list([],[a,b,c,d]).  ⇒ no
   ?- in_list([a,c],[a,b,c,d]).  ⇒ no
   ?- in_list([b,d],[a,b,c,d]).  ⇒ no
   ```

6. `last_added_first/2`: a two place relation which takes a list and returns the same list with the last element of the input list added to the beginning of the result; i.e., `last_added_first(+`*`List`*`,−`*`List-With-Last-Added-First`*`)`

   Example queries:

   ```
   ?- last_added_first([a,b,c,d],X). ⇒ [d,a,b,c,d]
   ?- last_added_first([a,b,c],X). ⇒ [c,a,b,c]
   ?- last_added_first([],X). ⇒ no
   ```

7. `same_number/1`: a one place relation which takes a list as argument and succeeds if the list contains the same number of a as b elements, with all a's coming first (i.e., the language $a^n b^n$, with each character being encoded as an element of a list).

   Example queries:

   ```
   ?- same_number([a,a,b,b]).  ⇒ Yes
   ?- same_number([a,a,a,b,b,b]).  ⇒ Yes
   ?- same_number([a,a,b,b,b]).  ⇒ No
   ?- same_number([a,a,a,b]).  ⇒ No
   ?- same_number([aaaa,bbbb]).  ⇒ No
   ?- same_number([]).  ⇒ No
   ```

8. `same_number_mixed/1`: a one place relation which takes a list as argument and succeeds if the list contains the same number of a as b elements, coming in any order and including the empty list.

   Example queries:

   ```
   ?- same_number_mixed([b,a]).  ⇒ Yes
   ?- same_number_mixed([b,a,a]).  ⇒ No
   ?- same_number_mixed([a,a,b,b]).  ⇒ Yes
   ?- same_number_mixed([a,b,a,a,b,b]).  ⇒ Yes
   ?- same_number_mixed([a,b,a,a,b,b,a]).  ⇒ No
   ?- same_number_mixed([]).  ⇒ Yes
   ```

9. `mix/2`: a two place relation which takes a list as its first argument and returns as second argument each list that consists of all and only the elements of the input list in any order of occurrence; i.e., `mix(+List,−Mixed-list)`

   Example queries:

   ```
   ?- mix([a,b,c],X). ⇒ X = [a,b,c] ; X = [b,a,c] ; X = [b,c,a] ; X = [a,c,b]
   ; X = [c,a,b] ; X = [c,b,a]
   ```

   Hint: in defining `mix` it is useful to define an auxiliary relation `insert` which inserts a single element into an input list at any arbitrary position of the list and returns this newly constructed list.

Introduction to CL I (684.01) OSU Linguistics
Detmar Meurers Winter 2007

Exercise sheet 2a

(Submit as a plain text email message to dm@ling.osu.edu before class on Thursday, Jan. 18)

Provide Prolog definitions for the following relations. Define them using the minimal means necessary—in particular, there is no need to make use of other relations defined in class or predefined predicates.
Thoroughly test your predicates before handing them in!

1. `last/2`: a two place relation which takes a list as first argument and returns the last element of that list (if there is one) as second argument; i.e., `last(+`*`List`*`,−`*`Last-List-element`*`)`

   Example queries:

   - `?- last([a,b,c,d]),X).` ⇒ `X=d`
   - `?- last([a,b,c],X).` ⇒ `X=c`
   - `?- last([],X).` ⇒ `no`

2. `interleave/3`: a three place relation interleaving the elements from the first with the elements of the second list (of same length).

   Example queries:

   `?- interleave([a,b,c],[1,2,3],X).` ⇒ `X=[a,1,b,2,c,3]`

   `?- interleave([],[],X).` ⇒ `X=[]`

3. `firstLastSwap/2`: a two place relation which takes a list and returns the same list with one difference: the first list element and the last list element are exchanged; i.e., `firstLastSwap(+`*`List`*`,−`*`List-With-First-Last-Swapped`*`)`

   Example queries:

   - `?- firstLastSwap([a,b,c,d],X).` ⇒ `X=[d,b,c,a]`
   - `?- firstLastSwap([a,b,c],X).` ⇒ `X=[c,b,a]`
   - `?- firstLastSwap([a,c],X).` ⇒ `X=[c,a]`
   - `?- firstLastSwap([],X).` ⇒ `no`

4. `next_element/3`: a three place relation which takes a shorter list and a longer list as arguments, checks whether the first list is contained in the second, and returns the element immediately following the match (backtracking if there are multiple possible matches).

   Example queries:

   - `?- next_element([b],[k,a,b,c,d,b,a,e,f],X).` ⇒ `X=a; X=d`
   - `?- next_element([a,b,c],[k,a,b,c,d,e,f],X).` ⇒ `X=d`
   - `?- next_element([a,b,c],[k,a,b,c,d],X).` ⇒ `X=d`
   - `?- next_element([a,b,c],[k,a,b,c,d,e,a,b,c,f,g],X).` ⇒ `X=d; X=f`
   - `?- next_element([a,b,c],[b,a,c,b,d],X).` ⇒ `no`
   - `?- next_element([a,b,c],[a,b,c],X).` ⇒ `no`
   - `?- next_element([a,b,c],[],X).` ⇒ `no`
   - `?- next_element([],Y,X).` ⇒ `no`

5. `preceding_element/3`: a three place relation which takes a shorter list and a longer list as arguments, checks whether the first list is contained in the second, and returns the element immediately preceding the match (backtracking if there are multiple possible matches)..

   Example queries:

   - `?- preceding_element([a,b,c,d],[k,a,b,c,d,e,f],X).` ⇒ `X=k`
   - `?- preceding_element([a,b,c,d],[d,a,q,a,b,c,d,e,f],X).` ⇒ `X=q`
   - `?- preceding_element([a,b,c],[b,a,c,b,d],X).` ⇒ `no`
   - `?- preceding_element([a,b,c],[a,b,c],X).` ⇒ `no`
   - `?- preceding_element([a,b,c],[],X).` ⇒ `no`
   - `?- preceding_element([],Y,X).` ⇒ `no`