Principles of Programming Languages, 2013.02.13

Notes:
- Total available time: 2h.
- You may use any written material you need.
- You cannot use computers, phones or laptops during the exam.

Exercise 1, Haskell (5+6 pts)

CBCB Inc. produces its goods (bikes and other stuff) by assembling various parts coming from several suppliers. CBCB is connected to its suppliers through a number of supply brokers, and each of them is specialized in dealing one type of item (e.g. wheels, components, brakes...). At a fixed scheduled times (say, once every three months) each broker sends to CBCB's server a message containing the current offers from its suppliers (e.g. of wheels). Such message is a sequence of offers of analogous items. CBCB's server enqueues such sequence in a sequence of sequences; when this is complete, the servers calls the procedure *allPossibleBikes* to return all the possible combinations of bikes that could be built.

Example:
supply broker 1 sends (ultra-wheel-1, WheelyWheel);
supply broker 2 sends ("very nice frame", "another frame", "frame000");
supply broker 3 sends (3444,712,9938,115403).

In this case allPossibleBikes should return the sequence:
(ultra-wheel-1 "very nice frame" 3444) (WheelyWheel "very nice frame" 3444) (ultra-wheel-1 "another frame" 3444) (WheelyWheel "another frame" 3444) (ultra-wheel-1 "frame000" 3444) (WheelyWheel "frame000" 3444) (ultra-wheel-1 "very nice frame" 712) (WheelyWheel "very nice frame" 712) (ultra-wheel-1 "another frame" 712) (WheelyWheel "another frame" 712) (ultra-wheel-1 "frame000" 712) (WheelyWheel "frame000" 712) (ultra-wheel-1 "very nice frame" 9938) (WheelyWheel "very nice frame" 9938) (ultra-wheel-1 "another frame" 9938) (WheelyWheel "another frame" 9938) (ultra-wheel-1 "frame000" 9938) (WheelyWheel "frame000" 9938)(ultra-wheel-1 "very nice frame" 115403) (WheelyWheel "very nice frame" 115403) (ultra-wheel-1 "another frame" 115403) (WheelyWheel "another frame" 115403) (ultra-wheel-1 "frame000" 115403) (WheelyWheel "frame000" 115403)

1) Define a suitable data structure for CBCB's orders in Haskell, knowing that the sequence can contain any number of elements, and that each supply broker message is a nonempty sequence of any number of offers. For simplicity, assume that each offer may either be represented as a natural number, or a string.
2) Define a purely functional version of *allPossibleBikes* in Haskell.

Exercise 2, Scheme (5+6 pts)

1) Define a portion of a set library for Scheme, optimized for lookup (procedure *in?*) and insertion (procedure *put!*) of elements. Set elements may be numbers or symbols.
2) Define an *intersection* procedure for this library that can have any number of sets as arguments (at least one).

Exercise 3, Prolog (5+5 pts)

CBCB has been notified that some of its suppliers had problems with their servers, so they could include in the same sequence two or more copies of the same item.

1. Define a procedure to get only unique bikes in the sequence obtained by *allPossibleBikes* of Ex 1.
2. CBCB decided to filter the offer sequences from the supply brokers, to check which of them have problems. Write a procedure that, given an input sequence, returns only the repeated items in it.

Solutions

Ex 1.1

```
data Item = Nm Int | St String deriving Show
type Order = [Item]
-- example:
bikes = [[St "ultra-wheel-1", St "WheelyWheel"], [St "very nice frame", St "another frame", St "frame000"],
         [Nm 3444, Nm 712, Nm 9938, Nm 115403]]
```

Ex 1.2

```
allPossibleBikes ms = foldr k [[]] ms
  where k m m' = [(x:xs) | x <- m, xs <- m']
```

Ex 2.1

```
(library (sets)
  (export
      make-set
      in?
      put!
      remove!
      intersect)
  (import  (rnrs)(rnrs hashtables))

    (define make-set make-eqv-hashtable)

    (define (in? set x)
      (hashtable-ref set x #f)) ;; or hashtable-contains?

    (define (put! set x)
      (hashtable-set! set x #t))

    (define remove! hashtable-delete!)
```

Ex 2.2

```
    (define (isect x y)
      (let ((res (hashtable-copy x #t)))
        (vector-for-each (lambda (e)
                           (remove! res e))
                           (hashtable-keys y))
        res))

    (define (intersect set . sets)
      (if (null? sets)
          set
          (apply intersect (cons (isect set (car sets))
                           (cdr sets)))))
```

Ex 3.1

code for removing duplicates in a list: (there is also *list_to_set* in the library)

```
remdupl([],[]).
remdupl([X|Xs],[X|Ys]) :- not(member(X,Xs)), remdupl(Xs,Ys).
remdupl([X|Xs],Out) :- member(X,Xs), remdupl(Xs,Out).
```

Ex 3.2 (it's like the previous one, just with the two last clauses swapped)

```
onlydup([],[]).
onlydup([Y|Xs],[Y|Ys]) :- member(Y,Xs), onlydup(Xs,Ys).
onlydup([X|Xs],Ys) :- not(member(X,Xs)), onlydup(Xs,Ys).
```