Homework 4*
SNU 4190.210, Fall 2011
Kwangkeun Yi

due: 10/08(Mon), 24:00

The objectives of this homework :

- Experience a problem which you don't know if it's a correct problem or not.
- To get a motivation of knowing there's lots more to learn.

Have a confidence that a program you make outputs a correct result.

Exercise 1 "Don't match"

String is a list of integers between 0 and 9. i.e. 0000, 1102201, 998011199, etc. "String s matches code c." means that string s is a part of string set which is expressed by code c. Code c is defined as following:

$$c \rightarrow 0|1|\cdots|9|c{\cdot}\,c|c \shortmid c|c?|c*$$

String set $[\![c]\!]$ expressed by code c is defined as following. An empty string is ecpressed as ϵ.

$$\begin{array}{rcl} [\![0]\!] & = & \{0\} \\ & \vdots & \\ [\![9]\!] & = & \{9\} \\ [\![c_1 \cdot c_2]\!] & = & \{s_1 s_2 \mid s_q \in [\![c_1]\!], s_2 \in [\![c_2]\!]\} \\ [\![c_1 \shortmid c_2]\!] & = & [\![c_1]\!] \cup [\![c_2]\!] \\ [\![c?]\!] & = & \{\epsilon\} \cup [\![c]\!] \\ [\![c*]\!] & = & \{\epsilon\} \cup [\![c]\!] \cup [\![c \cdot c]\!] \cup [\![c \cdot c \cdot c]\!] \cup \cdots \end{array}$$

Define the following functions which hide inner content of the code:

*translated by Youngseok Lee

`atom` : int → code	`is-atom?` : code → bool
`dot` : code × code → code	`is-dot?` : code → bool
`bar` : code × code → code	`is-bar?` : code → bool
`optional` : code → code	`is-optional?` : code → bool
`star` : code → code	`is-star?` : code → bool
`de-atom` : code → int	`de-star` : code → code
`de-dot-0` : code → code	`de-bar-0` : code → code
`de-dot-1` : code → code	`de-bar-1` : code → code
`de-optional` : code → code	

What they do:

$$\begin{aligned} \texttt{(de-atom (atom } n\texttt{))} &= n \\ \texttt{(de-dot-}i\texttt{ (dot } c_0\ c_1\texttt{))} &= c_i \\ \texttt{(de-bar-}i\texttt{ (bar } c_0\ c_1\texttt{))} &= c_i \\ \texttt{(de-star (star } c\texttt{))} &= c \\ \texttt{(de-optional (optional } c\texttt{))} &= c \end{aligned}$$

Using above funtions, define the following function which determines whether a string s matches code c.

`smatch` : string × code → bool

For example, `(smatch` 11 1·0`*`·1`)` returns true, `(smatch` 11 (10)`*`·1`)` returns false. String is implemented by a list of integers. i.e. 1001 as `'(1 0 0 1)`. □

Exercise 2 "Delightful anxiety"

My concern is to get a present for my nephews so that thay are all happy, with the least cost. They are all jealous. Every year at this time of a year, I give them a present package, and when they receive it, they compare it with each other and cry over other's presents.
I decided to solve this problem this way. I tell my nephews the list of presents they are going to receive this year, and tell them they're going to get a subset of them. Then I tell them the condition they'll be satisfied in (not fighting over the present). Then I prepare the package of presents to suit all the conditions with the least cost. Their conditions are like this: "I want to get a present same as Eric and a fountain pen at least", "I want the common present between Vince and Sloan, and present of Ari, excluding the CD.", etc. For example, if there's three nephews A,B,and C, the present they'll get according to their conditions are as following:

- Jealous nephews don't get any presents. A: "At least as much as B", B: "At least as much as A", C: "At least as much as B".
- Cranky nephews don't get any presents either. A: "At least as much as B but a fountain pen", B: "At least as much as A but CD", C:"At least as much as B but USB".
- Greedy nephews don't get any presents either. A: "At least as much as B and C", B: "At least as much as A and C", C: "At least as much as A and B".

- Unjealous nephews get what they want. A: "At least a fountain pen", B: "At least a CD", C: "At least a USB".

Let's say that each nephew's conditions are expressed as following:

"At least, I have to receive ($cond_1$ and $\cdots$ and $cond_k$)."

Now define a `shoppingList` which creates a minimum shopping list by receiving each nephew's conditions:

`shoppingList` : `(id * cond) list` $\rightarrow$ `(id * gift list) list`

The result is a present list for each nephew. For example, let's say the conditions for each nephew is as following

A : At least ($\{1, 2\}$ and common(B, C)).
B : At least common(C, $\{2, 3\}$).
C : At least ($\{1\}$ and (A except $\{3\}$)).

The minumum package of presents are $\{1, 2\}$ for A, $\{2\}$ for B, and $\{1, 2\}$ for C, thus ouput of `shoppingList` should be

```
((A . (1 2)) (B . (2)) (C . (1 2)))
```

The present package for each nephew is a "set". In other words, one package doesn't contain two or more of the same present.

For implementation, the functions for making and using the condition for the present package are as following. Functions which need to be made:

```
mustItems: gift list -> cond           (* Items that I must have *)
mustBeTheSame: id -> cond              (* Presents for another nephew *)
mustHaveExceptFor: cond * gift list -> cond (* Condition but some presents *)
mustHaveCommon: cond * cond -> cond         (* Common presents of two conditions *)
mustAnd: cond * cond -> cond                (* Both of two conditions *)
```

Functions which is used:

```
isItems: cond -> bool                  isSame: cond -> bool
isExcept: cond -> bool                 isCommon: cond -> bool
isAnd: cond -> bool
whichItems: cond -> gift list          whoTheSame: cond -> id
condExcept: cond -> cond               itemsExcept: cond -> gift list
condCommon: cond -> cond * cond        condAnd: cond -> cond * cond
```

From the above, `gift` is implemented as an integer, and the nephew's name `id` as a symbol in Scheme. For example, nephew A's conditions(`cond`) will be made as following.

```
(mustAnd (mustItems '(1 2))
         (mustHaveCommon (mustBeTheSame 'B)
                         (mustBeTheSame 'C)))
```

□