Homework 1

SNU 4190.310, Fall 2013

Kwangkeun Yi

Due: 9/14, 24:00

Exercise 1 "씨그마"

우리가 중고등 수학시간에 숱하게 썼던 다음의 "씨그마"를 OCaml로 정의하세요:

$$\Sigma_{n=a}^{b} f(n)$$

씨그마의 타입은

```
sigma : int * int * (int -> int) -> int.
```

즉, `sigma(a,b,f)`로 표현하면 $\Sigma_{n=a}^{b} f(n)$과 같도록. □

Exercise 2 "참거짓"

Propositional Logic 식들(`formula`)을 다음과 같이 정의했습니다:

```
type formula = TRUE
             | FALSE
             | NOT of formula
             | ANDALSO of formula * formula
             | ORELSE of formula * formula
             | IMPLY of formula * formula
             | LESS of expr * expr
and  expr = NUM of int
          | PLUS of expr * expr
          | MINUS of expr * expr
```

주어진 formula를 받아서 참값을 만들어내는 함수 eval

$$\texttt{eval} : \texttt{formula} \rightarrow \texttt{bool}$$

를 정의하세요. □

Exercise 3 "자연수"

자연수 nat 는 다음과 같이 정의될 수 있다:

```
type nat = ZERO | SUCC of nat
```

두 자연수를 받아서 그 합/곱에 해당하는 자연수를 만드는 두 함수

```
natadd : nat * nat -> nat
natmul : nat * nat -> nat
```

를 정의하세요. □

Exercise 4 "CheckMetroMap"

아래 metro 타입을 생각하자:

```
type metro = STATION of name
           | AREA of name * metro
           | CONNECT of metro * metro
and name = string
```

아래 checkMetro 함수를 정의하라:

```
checkMetro: metro -> bool
```

checkMetro는 주어진 metro 가 제대로 생겼는지를 확인해 준다. "metro가 제대로 생겼다"는 것은(iff) 메트로 역 이름(id in STATION(id))들이 항상 자기 이름의 지역(m in AREA(id, m))에서만 나타나는 경우를 뜻한다.

예를들어, 제대로 생긴 metro 들은:

- AREA("a", STATION "a")
- AREA("a", AREA("a", STATION "a"))

- `AREA("a", AREA("b", CONNECT(STATION "a", STATION "b")))`
- `AREA("a", CONNECT(STATION "a", AREA("b", STATION "a")))`

그렇지 못한 것들의 예들은:

- `AREA("a", STATION "b")`
- `AREA("a", CONNECT(STATION "a", AREA("b", STATION "c")))`
- `AREA("a", AREA("b", CONNECT(STATION "a", STATION "c")))`

□

Exercise 5 "왼쪽편에 쏠려있는 힙"

"우선큐"(priority queue)라는 자료구조의 핵심은, 원소의 순서가 자료구조에 들어가는 순서와 무관하게 원래의 고유 순서를 가지고 있는 것입니다. 큐와 스택과는 다르지요. 큐는 원소들이 들어오는 순서대로 나가게되고, 스택은 그 역순으로 나가게 되는 구조이지만, 우선큐는 원소의 고유 순서가 있어서, 그 순서대로 나가게 됩니다.

따라서, 우선큐는 현재 있는 원소들 중에서 제일가는 원소를 알아보는 데에 특화되있어야 합니다. 힙(heap)이 대표적인 것이지요. 그중에서도 왼쪽으로 쏠린 힙(leftist heap, 왼쏠힙)이라는 것을 구현해 봅시다.

- 왼쏠힙: 힙은 힙인데 모든 왼쪽 노드의 급수가 오른쪽 형제 노드의 급수보다 크거나 같다.
- 노드의 급수: 그 노드에서 오른쪽으로만 타고 내려가서 끝날 때 까지 내려선 횟수, 즉 오른편 척추의 길이.
- 힙: 이진 나무 구조로서 모든 갈래길 길목의 값이 갈라진 후의 모든 노드들의 값보다 작거나 같다.

왼쏠힙은 다음의 타입으로 정의됩니다:

```
type heap = EMPTY | NODE of rank * value * heap * heap
and rank = int
and value = int
```

넣고, 빼고, 하는 등의 함수는 다음으로 정의됩니다:

```
exception EmptyHeap
let rank = function EMPTY -> 0
                  | NODE(r,_,_,_) -> r
```

```
let insert = function (x,h) -> merge(h, NODE(0,x,EMPTY,EMPTY))
let findMin = function EMPTY -> raise EmptyHeap
                     | NODE(_,x,_,_) -> x
let deleteMin = function EMPTY -> raise EmptyHeap
                       | NODE(_,x,lh,rh) -> merge(lh,rh)
```

나머지 함수 `merge`

```
merge: heap * heap -> heap
```

를 정의하세요. 이 때, 왼쏠힙의 장점을 살려서 여러분이 정의한 `merge`는 $O(\log n)$으로 끝나도록 해야 합니다 (n은 힙의 노드 수). (참고사실: 왼쏠힙에서 오른쪽 척추에 붙어있는 노드수는 많아야 $\lfloor \log(n+1) \rfloor$입니다.) 정의할 때 다음의 함수를 이용하시기를:

```
let shake = function (x,lh,rh) ->
       if (rank lh) >= (rank rh)
        then NODE(rank rh + 1, x, lh, rh)
        else NODE(rank lh + 1, x, rh, lh)n
```

□