

Homework 4
SNU 4190.310, 2025 봄
이 광근
Due: 4/18(금) 24:00

이번 숙제의 목적은:

- 수업시간에 살펴본, 상식적인 명령형 언어의 정확한 정의를 이해하고 그 실행기를 구현해 보기.
- 그 언어로 프로그램 해보면서 아쉬운 점에 눈뜨기.
- 앞으로 프로그래밍 언어 구현에서 넘어야 할 산이, 상식만 가지고는 넘기 어렵다는 것을 겪어보기. 마지막 두 문제입니다.

Exercise 1 (40pts) “K- 실행기”

수업시간에 정의한 명령형 언어 K^{-1} 를 생각하자. 이번 숙제는 K- 프로그램을 의미정의대로 실행시키는 함수(interpreter)를 작성하는 것이다.

아래의 KMINUS 꼴을 가지는 모듈 K를 정의하라.

```
module type KMINUS =  
sig  
  exception Error of string  
  type id = string  
  type exp = NUM of int | TRUE | FALSE | UNIT  
            | VAR of id  
            | ADD of exp * exp  
            | SUB of exp * exp
```

¹숙제를 위한 문법과 의미의 정확한 정의는 TA페이지 참고.

```

| MUL of exp * exp
| DIV of exp * exp
| EQUAL of exp * exp
| LESS of exp * exp
| NOT of exp
| SEQ of exp * exp (* sequence *)
| IF of exp * exp * exp (* if-then-else *)
| WHILE of exp * exp (* while loop *)
| LETV of id * exp * exp (* variable binding *)
| ASSIGN of id * exp (* assign to variable *)
| READ of id
| WRITE of exp
| LETF of id * id list * exp * exp (* NEW: procedure binding *)
| CALLV of id * exp list (* NEW: call by value *)
| CALLR of id * id list (* NEW: call by reference *)
| RECORD of (id * exp) list (* NEW: record construction *)
| FIELD of exp * id (* NEW: access record field *)
| ASSIGNF of exp * id * exp (* NEW: assign to record field *)

type program = exp
type memory
type env
type value
val emptyMemory: memory
val emptyEnv: env
val run: memory * env * program -> value
end

```

K- 프로그램이 어떻게 `exp`들로 표현될지는 쉽게 추측할 수 있을 것입니다. `exp`으로 표현된 K- 프로그램이 S 라고 하면,

`K.run (K.emptyMemory, K.emptyEnv, S)`

는 프로그램 S 를 실행시키게 되는데, 성공적으로 끝나면 최후의 값을 내어주게 됩니다. 이때 프로그램은 실행중에 I/O를 하면서 프로그램이 하는 일을 바깥세상에 드러내게 됩니다. 실행중에 타입이 맞지 않는 프로그램이면 `Error`라는 예외상황을 발생시키고 프로그램 실행이 중단되어야 합니다. “Error”란 (if and only if) 정의된

의미 규칙으로는 그 프로그램의 의미가 정의될 수 없는 경우입니다. 입출력은 정수만 가능합니다. 출력은 정수를 화면에 뿌리고 “newline”을 프린트합니다. □

Exercise 2 (20pts) “K- 프로그래밍: 구조물 데이터”

다음은 K- 로 작성하고, 위에서 구현한 실행기 K.run로 실행시켜 제대로 실행되는 지를 확인한다.

두갈래 나무구조(binary tree)를 만들고 쓸 수 있는 아래의 함수들을 정의하라:

```
leaf:    int → tree           (* a leaf tree *)
makeLtree: int × tree → tree  (* a tree with only a left subtree *)
makeRtree: int × tree → tree  (* a tree with only a right subtree *)
makeTree: int × tree × tree → tree (* a tree with both subtrees *)

isEmpty: tree → bool         (* see if empty tree *)
rTree:   tree → tree         (* right subtree *)
lTree:   tree → tree         (* left subtree *)
nodeValue: tree → int       (* node value *)
dft:     tree → unit         (* print node values in depth-first order *)
bft:     tree → unit         (* print node values in breath-first order *)
```

위의 함수들 만을 이용해서 나무 구조를 만들고 dft와 bft를 돌려서 제대로 된 순서로 출력되는 지를 확인하라.

참고로, 만든 실행기에 메모리 소모량을 측정하는 장치를 달고, 프로그램을 돌렸을 때 얼마만큼의 메모리를 소모하는지를 재보자. 메모리 소모를 가능하면 아끼는 프로시저를 구현하도록 해보자. □

Exercise 3 (20pts) “탐사 준비”

탐사해야 할 지역의 지도를 보고 탐사를 성공리에 마치기 위해 필요한 최소의 준비물을 알아내는 프로그램을 작성해 보자.

탐사는 지도에 나타난 길을 따라 이동하면서 길에 놓인 보물상자를 열고 보물을 모아가는 것이고, 모든 보물이 모아지면 그 탐사는 성공한 것이다. 준비물은 모든 보물상자를 열 수 있는 열쇠들이다.

보물상자와 열쇠:

- 보물상자에는 고유의 알파벳 이름이 표시되어 있다.
- 이름없이 “*”라고 찍혀있는 보물상자도 있다.
- 같은 이름의 보물 상자는 같은 열쇠로 열린다.
- 하나의 열쇠는 외갈래 혹은 두갈래로 갈라진 가지구조(tree)이다.

- 열쇠는 반복해서 사용할 수 있다.

보물상자와 열쇠를 OCaml 타입으로 정의하면,

```
type treasure = StarBox | NameBox of string
type key = Bar | Node of key * key
```

탐사지도:

- 시작 지점은 하나이다.
- 길들은 모두 외길이거나 두 갈래로 나뉘어 진다.
- 보물상자들은 모두 막다른 골목의 끝에 있다.
- 갔던 길을 되돌아 오지 않고 왔던 곳으로 다시 오는 방법은 없다(tree).
- 길목에 세워진 안내판에는 앞으로 만날 보물상자의 알파벳 이름이 쓰여져 있다.
- 모든 안내판의 이름은 모두 다르다.

탐사지도를 OCaml 타입으로 정의하면,

```
type map = End of treasure
         | Branch of map * map
         | Guide of string * map
```

보물상자마다 필요한 열쇠의 모양은 보물상자의 위치가 전체 탐사지도에서 어디냐에 따라 결정되는데, 지도에서 각 지역이 암시하는 열쇠의 모양은 다음의 조건으로 결정된다:

현재위치(지도) e	위치의 뜻	열쇠모양의 조건
$*$	$*$ 보물상자	- (Bar)
x	x 라는 이름의 보물상자	현재 위치에서 x 를 열어줄 열쇠 모양
$\boxed{x}e_1$	안내판 \boxed{x} 이 앞에있는 지도 e_1	e_1 안에서 만날 보물상자 x 의 열쇠가 α 이고 e_1 의 시작점이 암시하는 열쇠모양을 β 라고 하면, 현재 위치가 암시하는 열쇠모양은 (α, β) (왼쪽가지 α , 오른쪽가지 β).
$e_1 e_2$	e_1 과 e_2 로 갈라지는 갈림길	e_1 의 시작점이 암시하는 열쇠모양은 (α, β) 이어야 하고 e_2 의 시작점이 암시하는 열쇠모양은 α 이어야 한다. 이때, 현재 위치가 암시하는 열쇠모양은 β .

예를들어, 각 지도를 성공적으로 탐험할 최소의(열쇠들 크기의 합을 기준으로) 열쇠꾸러미는 다음과 같다:

1. 지도 x 에는 $\{-\}$.
2. 지도 $\boxed{x}x$ 에는 $\{-\}$.
3. 지도 $(\boxed{x}x)|*$ 에는 $\{-\}$.
4. 지도 $(\boxed{x}(x|x))|*$ 를 성공적으로 탐험하는 것은 불가능.
5. 지도 $(\boxed{x}x)|(\boxed{y}y)|*$ 에는 $\{-\}$.
6. 지도 $(\boxed{x}x)|(\boxed{y}y)$ 에는 $\{-, (-, -)\}$.
7. 지도 $x|*$ 에는 $\{-, (-, -)\}$.

다음의 타입에 맞도록, 위와같은 일을 하는 `getReady` 함수

`getReady: map → key list`

를 정의하기 바랍니다.

□

Exercise 4 (20pts) “돼지선물”

엄마돼지의 고민은 새끼돼지들이 모두 행복해 할 수 있도록 가장 저렴하게 봄 선물을 준비하는 것이다. 새끼돼지들은 시샘이 많다. 매년 이맘때쯤이면 그녀는

새끼돼지들에게 선물을 한 꾸러미씩 나눠주는데, 받고나면 새끼돼지들끼리 다른 새끼돼지가 받은 선물을 시샘하면서 서로 조르고 울고. 그래서 다시 정리해서 주면 또 만족스럽지 않아서 조르고 울고.

엄마돼지는 그 고민을 다음과 같이 풀기로 했다. 선물 쇼핑을 나가기전에 새끼돼지들에게 받을 선물의 후보들을 알려주고 각자는 그중의 부분집합을 선물로 받을 것이라고 선언한다. 그러곤 새끼돼지들에게 각자가 만족할(싸우지않을) 조건을 얘기하라고 한다. 그녀는 **가능하다면 가장 적은 비용으로** 이러한 조건들을 모두 만족시키도록 선물꾸러미들을 준비하려고 한다. 새끼돼지마다 받는 선물꾸러미는 “집합”이다. 즉, 한 선물 꾸러미에는 같은 선물이 두개이상 포함되지는 않는다.

새끼돼지들의 조건들은 이런식이다: “나는 최소한 갤럭시폰과 동생 영희가 받은 선물만큼은 받아야 해요.” “나는 최소한 철수오빠와 숙희언니의 선물들에 공통된 것들 하고, 영숙이 선물중에서 노트북 말고는 가져야 해요” 등등. 예를 들어 A, B, C 세명의 돼지들이 있다면, 조건에 따라 받는 선물은 다음과 같다:

- 샘플 많은 돼지들은 아무것도 못받는다. A: “최소한 B 만큼”, B: “최소한 A 만큼”, C: “최소한 B 만큼.”
- 까다로운 돼지들도 아무것도 못받는다. A: “최소한 B 만큼에서 iPad 말고”, B: “최소한 A 만큼에서 Galaxy 말고”, C: “최소한 B 만큼에서 러닝화 말고.”
- 탐욕스런 돼지들도 아무것도 못받는다. A: “최소한 B와 C만큼”, B: “최소한 A와 C만큼”, C: “최소한 A와 B만큼.”
- 샘플 없는 돼지들은 원하는 것만 받는다. A: “최소한 iPad”, B: “최소한 Galaxy”, C: “최소한 러닝화.”

돼지의 조건은 다음과 같은 꼴로 표현된다고 하자:

X 는 최소한 ($cond_1$ 그리고 \dots 그리고 $cond_k$)을 받아야해요.

수학식으로는

$$X \supseteq cond_1 \cup \dots \cup cond_k.$$

1. 예를들어, 새끼돼지 세 마리의 조건이 다음과 같을때

$$A \supseteq \{1, 2\} \cup (B \cap C)$$

$$B \supseteq C \cap \{2, 3\}$$

$$C \supseteq \{1\} \cup (A \setminus \{3\})$$

최소의 선물꾸러미들은 A 에게 $\{1, 2\}$, B 에게 $\{2\}$, C 에게 $\{1, 2\}$ 이다.

2. 조건이 다음과 같을때는

$$A \supseteq \{1, 2\} \setminus B$$

$$B \supseteq A \setminus C$$

$$C \supseteq \{2\} \cap A$$

최소의 선물꾸러미는? 혹시 이 경우 최소의 선물꾸러미가 없는건 아닐까?

다섯 새끼돼지들의 조건(require)을 받아서 최소의 선물쇼핑 리스트가 있다면 찾아주는 shoppingList를 작성하라:

```
shoppingList: require list → (id * gift list) list
```

OCaml 타입들은 다음과 같다:

```
type require = id * (cond list)
and cond = Items of gift list      (* gifts *)
        | Same of id              (* same gifts as for id *)
        | Common of cond * cond    (* common gifts *)
        | Except of cond * cond    (* exclude gifts *)
and gift = int                    (* gift id *)
and id = A | B | C | D | E       (* pig names *)
```

□