

Homework 1

SNU 4910.102A Spring 2006

Kwangkeun Yi

due: 4/4 24:00

Exercise 1 [k -진수 연산법]

일반적으로 k 진수($k > 1$)는 다음과 같이 표현한다.

$$d_0 \cdots d_n$$

여기서

$$\forall d_i \in \{0, \dots, k-1\}.$$

그리고 " $d_0 \cdots d_n$ "은 크기가

$$d_0 \times k^0 + \cdots + d_n \times k^n$$

인 정수를 표현한다.

이것을 살짝 확장해서 " k 진수"를 다음과 같이 정의해보자. 표현은

$$d_0 \cdots d_n$$

여기서

$$\forall d_i \in \{1-k, \dots, 0\} \cup \{0, \dots, k-1\}.$$

그리고 " $d_0 \cdots d_n$ "은 크기가

$$d_0 \times k^0 + \cdots + d_n \times k^n$$

인 정수를 표현한다.

예를 들어, 2진수의 경우를 생각하자. 베이스가 $\{-1, 0, 1\}$ 이 되겠다. 0이 0을, +가 1을 -가 -1을 표현한다고 하면, +는 1을, +0+는 5를, +-는 -1을, +-0-는 -9인 정수를 표현한다.

이러한 2진수 N 의 집합을 귀납적으로 정의하면 다음과 같다:

$$\begin{array}{l}
 N ::= 0 \\
 \quad | + \\
 \quad | - \\
 \quad | 0N \\
 \quad | +N \\
 \quad | -N
 \end{array}$$

그리고, Scheme에서는 2진수 N 을 다음과 같은 방법 \underline{N} 에 의해 Scheme의 리스트로 표현할 수 있다:

$$\begin{array}{l}
 \underline{0} = (\text{cons } 'z \text{ nil}) \\
 \underline{+} = (\text{cons } 'p \text{ nil}) \\
 \underline{-} = (\text{cons } 'n \text{ nil}) \\
 \underline{0N} = (\text{cons } 'z \underline{N}) \\
 \underline{+N} = (\text{cons } 'p \underline{N}) \\
 \underline{-N} = (\text{cons } 'n \underline{N})
 \end{array}$$

즉, $0+-$ 는 Scheme에서

$$(\text{cons } 'z (\text{cons } 'p (\text{cons } 'n \text{ nil})))$$

로 표현된다, 왜냐하면

$$\begin{array}{l}
 \underline{0+-} = (\text{cons } 'z \underline{+-}) \\
 = (\text{cons } 'z (\text{cons } 'p \underline{-})) \\
 = (\text{cons } 'z (\text{cons } 'p (\text{cons } 'n \text{ nil}))).
 \end{array}$$

자 이제, 위와 같이 표현되는 2진수를 받아서 그것의 값을 계산하는 함수 `crazy2val`을 정의하라

$$\text{crazy2val} : 2\text{진수} \rightarrow \text{정수}.$$

□

Exercise 2 두 2진수를 받아서 2진수의 합에 해당하는 2진수를 내어놓는 함수 `crazy2add`를 정의하라

$$\text{crazy2add} : 2\text{진수} \times 2\text{진수} \rightarrow 2\text{진수}.$$

위의 `crazy2add`는 다음의 성질이 만족되어야 한다:

- 당연히, 임의의 2진수 z 과 z' 에 대해서

$$(\text{crazy2val } (\text{crazy2add } z \ z')) = (\text{crazy2val } z) + (\text{crazy2val } z').$$

- `crazy2add`은 재귀적으로 정의되어야 한다.

□

Exercise 3 두 2진수를 받아서 2진수의 곱에 해당하는 2진수를 내어놓는 함수 `crazy2mul`를 정의하라

$$\text{crazy2mul} : 2\text{진수} \times 2\text{진수} \rightarrow 2\text{진수}.$$

위의 `crazy2mul`은 다음의 성질이 만족되어야 한다:

- 당연히, 임의의 2진수 z 과 z' 에 대해서

$$(\text{crazy2val } (\text{crazy2mul } z \ z')) = (\text{crazy2val } z) \times (\text{crazy2val } z').$$

- `crazy2mul`은 재귀적으로 정의되어야 한다.

□

Exercise 4 “나무구조 데이터”

전산학(computer science)은 나무를 사랑한다. 깊고 검은 숲이나, 아름답리 나무나, 왕성하게 뻗은 가지들을 우리는 늘 다루게 된다.

나무구조(혹은 가지구조, tree)는 다음과 같이 정의된다:

- 기본 나무구조: 잎새 하나는 나무구조이다.
- 나무구조들을 품고있는 나무구조: 하나의 꼭지에서 한 개 이상의 나무구조들이 하나하나 가지로 뻗어나간 구조는 다시 나무구조이다.

위의 두 가지 조건이 나무구조를 만드는 두가지 방법을 결정한다: 기본적인 나무를 만드는 방법(base case)과 만든 나무들을 가지고 새로운 나무를 만드는 방법(inductive case).

나무구조를 만드는 다음의 두 함수를 정의하라:

$$\text{leaf} : \alpha \rightarrow \text{tree}$$

$$\text{node} : \text{tree list} \rightarrow \text{tree}$$

`leaf`는 임의의 타입(“ α ”로 표현하기로 하자)의 값을 가지는 잎새 나무를 만든다. 즉, (`leaf 1`)하면 정수 1을 가지는 잎새 나무가, (`leaf 'a`)하면 심볼

a를 가지는 잎새나무가, (`leaf` '(1 2))하면 리스트 (1 2)를 가지는 잎새나무가 되겠다. `node`는 나무들의 리스트를 받아서 그들을 차례로 매달고 있는 새로운 나무를 만든다. `node`가 빈 리스트를 받으면 빈 나무를 만든다.

나무를 사용하는 다음 세가지 함수들도 정의하라:

```
is-empty-tree? : tree → bool
is-leaf? : tree → bool
leaf-val : tree → α
nth-child : tree × nat → tree
```

`is-empty-tree?`는 주어진 나무가 빈 나무인지를 판별한다. `is-leaf?`는 주어진 나무가 잎새 나무인지 아닌지를 판별한다. `leaf-val`은 잎새가 가지고 있는 데이터를 꺼낸다. `nth-child`는 나무와 자연수 $n \geq 0$ 을 받아서 그 나무의 n 번째 가지의 나무(n -th subtree)를 내놓는다. □

Exercise 5 “모빌 무게 재기”

천장에 매달려 균형을 잡은 채 은은히 흔들리고 있는 모빌을 떠올려보자. 일반적인 두갈래 모빌은 다음과 같이 정의된다:

- 모형 하나는 모빌구조이다.
- 모빌들을 품고있는 모빌: 하나의 균형점에서 왼쪽/오른쪽의 모빌구조들이 뺄어나간 구조는 다시 모빌구조이다.

모빌구조를 만드는 다음 세 가지 함수를 정의하라, 단, 반드시 Exercise 4에서 정의한 함수만을 사용한다:

```
model : nat → mobile
make-branch : nat × mobile → branch
make-mobile : branch × branch → mobile
```

`model`은 입력한 자연수값만큼의 무게를 갖는 모형 모빌을 만든다. `make-branch`는 그 길이와 끝에 매달린 모빌을 받아 하나의 가지를 이룬다. `make-mobile`은 왼쪽/오른쪽의 가지를 받아 하나의 모빌을 이룬다.

모빌을 사용하는 다음 두 가지 함수들도 정의하라:

```
is-balanced? : mobile → bool
weight : mobile → nat
```

`is-balanced?`는 주어진 모빌이 '균형잡혀있는지'를 판별한다. 하나의 모빌은 양쪽 가지의 토크(길이×무게)가 같을 때 균형이 잡히며, 두 가지의 무게의 합

이 그 모빌의 무게가 된다. 한 모빌 구조 내의 모든 하위 모빌 구조가 균형상태에 있을 때, 그 모빌은 '균형잡혀있다'고 한다. *weight*는 그 모빌구조의 총 무게를 내놓는다. □

Exercise 6 “논리회로는 나무구조”

부울 회로(Boolean circuit)라는 것은 다음과 같이 귀납적으로 정의된다. 모든 부울 회로는 하나 이상의 입력과 하나의 출력을 가지고 있다.

- 기본: 0을 가진 전기줄은 부울 회로이다.
- 기본: 1을 가진 전기줄은 부울 회로이다.
- 귀납: 부울 회로 하나를 가지고 또다른 회로를 만드는 방법 *not*이 있다.
- 귀납: 부울 회로 두개를 가지고 또다른 회로를 만드는 방법 *and*가 있다.
- 귀납: 부울 회로 두개를 가지고 또다른 회로를 만드는 방법 *or*가 있다.

부울 회로를 만드는 위의 다섯가지 방법들을 정의하라:

zero : *circuit*
one : *circuit*
not : *circuit* → *circuit*
and : *circuit* × *circuit* → *circuit*
or : *circuit* × *circuit* → *circuit*

위의 함수들을 정의할 때는 반드시 Exercise 4에서 정의한 함수만을 사용한다. 즉, “요약의 경계”(abstraction barrier)를 깨지 않도록 한다. 그리고, 부울 회로가 어떻게 왜 나무구조가 되는지를 음미하자.

그리고 부울 회로를 사용하는 아래의 여섯가지 함수들도 정의하라:

is-zero? : *circuit* → *bool*
is-one? : *circuit* → *bool*
is-not? : *circuit* → *bool*
is-and? : *circuit* → *bool*
is-or? : *circuit* → *bool*
sub-circuit : *circuit* × *nat* → *circuit*

*sub-circuit*은 회로와 자연수 $n \geq 0$ 을 받아서 그 회로의 n 번째 가지의 회로(n -th *sub-circuit*)를 내놓는다. □

Exercise 7 “논리 회로의 계산”

Exercise6의 함수들로 만들어진 부울 회로의 최종 출력값(회로의 의미)를 계산하는 함수

$$\text{output} : \text{circuit} \rightarrow \{0, 1\}$$

를 정의하라.

부울 회로의 최종 출력값은 그 회로가 어떻게 만들어 졌냐에 따라 다음과 같이 재귀적으로 정의된다. **zero**의 출력값은 0. **one**의 출력값은 1. (**not** B)은 회로 B 의 출력값이 0이면 1, 1이면 0. (**and** B_1 B_2)은 회로 B_1 과 B_2 둘의 출력값이 모두 1일 때만 1, 아니면 0. (**or** B_1 B_2)은 회로 B_1 과 B_2 둘의 출력값이 모두 0일 때만 0, 아니면 1. □