Homework 1

SNU 4910.102, 2008 가을

due: 9/19(금), 24:00

Exercise 1 "좋은사이"

두개의 정수 리스트를 받아서 각 리스트의 원소들을 차례대로 사이사이에 끼워주는 함수 `zipper`를 작성하세요: 즉, `(zipper '(1 2 3) '(4 5))`는 `(1 4 2 5 3)`을 만들어낸다. 빈 리스트는 무시된다. □

Exercise 2 "좋은사이들"

정수 리스트의 리스트를 받아서 각 리스트의 원소들을 차례대로 사이사이에 끼워주는 함수 `zipperN`을 작성하세요: 즉, `(zipperN '((1 2 3) (4) (9 10 11)))`는 `(1 4 9 2 10 3 11)`을 만들어낸다. 빈 리스트는 무시된다. □

Exercise 3 "k-친수"

일반적으로 k진수($k > 1$)는 다음과 같이 표현한다.

$$d_0 \cdots d_n$$

여기서

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

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 "k친수"를 다음과 같이 정의해보자. 표현은

$$d_0 \cdots d_n$$

여기서

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

그리고 "$d_0 \cdots d_n$"은 크기가

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

인 정수를 표현한다.

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

이러한 2친수 N의 집합을 귀납적으로 정의하면 다음과 같다:

$$\begin{array}{rcl} N & ::= & \texttt{0} \\ & | & \texttt{+} \\ & | & \texttt{-} \\ & | & \texttt{0}N \\ & | & \texttt{+}N \\ & | & \texttt{-}N \end{array}$$

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

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

즉, `0+-`는 Scheme에서

```
(cons 'z (cons 'p (cons 'n nil)))
```

로 표현된다, 왜냐하면

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

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

$$\texttt{crazy2val} : \text{2친수} \to \text{정수}.$$

□

Exercise 4 "2친수 더하기"

두 2친수를 받아서 2친수의 합에 해당하는 2친수를 내어놓는 함수 `crazy2add`를 정의하라

$$\texttt{crazy2add} : \text{2친수} \times \text{2친수} \to \text{2친수}.$$

위의 `crazy2add`는 다음의 성질이 만족되야 한다:

- 당연히, 임의의 2친수 z 과 z'에 대해서

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

- `crazy2add`은 재귀적으로 정의되야 한다.

□

Exercise 5 "나무구조 데이타"

컴퓨터과학(computer science)은 나무를 사랑한다. 깊고 검은 숲이나, 아름드리 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:

$$\texttt{leaf} : \alpha \to \textit{tree}$$
$$\texttt{node} : \textit{tree list} \to \textit{tree}$$

`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 `a`를 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:

$$\texttt{is-empty-tree?} : \textit{tree} \to \textit{bool}$$
$$\texttt{is-leaf?} : \textit{tree} \to \textit{bool}$$
$$\texttt{leaf-val} : \textit{tree} \to \alpha$$
$$\texttt{nth-child} : \textit{tree} \times \textit{nat} \to \textit{tree}$$

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

Exercise 6 "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 뻗어나간 구조는 다시 모빌구조이다.

모빌구조를 만드는 다음 세 가지 함수를 정의하라, 단, 반드시 Exercise 5에서 정의한 함수만을 사용한다:

$$\texttt{model} : \mathit{nat} \to \mathit{mobile}$$
$$\texttt{make-branch} : \mathit{nat} \times \mathit{mobile} \to \mathit{branch}$$
$$\texttt{make-mobile} : \mathit{branch} \times \mathit{branch} \to \mathit{mobile}$$

`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:

$$\texttt{is-balanced?} : \mathit{mobile} \to \mathit{bool}$$
$$\texttt{weight} : \mathit{mobile} \to \mathit{nat}$$

`is-balanced?`는 주어진 모빌이 "균형잡혀있는지"를 판별한다. 하나의 모빌은 양쪽 가지의 토크(길이×무게)가 같을 때 균형이 잡히며, 두 가지의 무게의 합이 그 모빌의 무게가 된다. 한 모빌 구조 내의 모든 하위 모빌 구조가 균형 상태에 있을 때, 그 모빌은 "균형잡혀있다"고 한다. `weight`는 그 모빌구조의 총 무게를 내놓는다. □