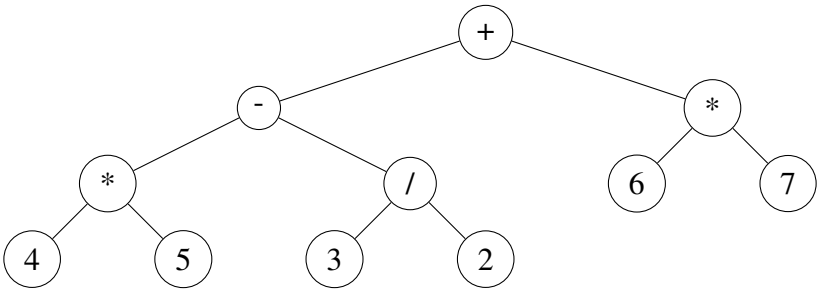
Prof. Dr. Christoph Bockisch
Steffen Dick
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Übungen zur Vorlesung
Deklarative Programmierung: Sommersemester 2018

Nr. 7, Abgabe bis 18.6.2018 11:59 Uhr

Aufgabe 7.1: Arithmetische Buxbäume — 4 Punkte



Der Operatorbaum arithmetischer Ausdrücke (vgl. Abb. oben) kann in DrRacket durch S-Expressions dargestellt werden. Der Ausdruck zur Repräsentation des Beispielbaums ist wie folgt:

```
'(((4 * 5) - (3 / 2)) + (6 * 7))
```

a) Implementieren Sie eine Funktion `(evalTree t)`, die den übergebenen Baum (in Form einer S-Expression) zu einer Zahl auswertet. Beschränken sie sich dabei auf die Operationen +,-,*,/.

b) Erweitern Sie das Programm so, dass die Blätter nicht nur natürliche Zahlen, sondern auch Konstanten (z.B. repräsentiert durch Symbole) enthalten können. Bei der Auswertung muss nun neben dem eigentlichen Baum eine Zuordnung von Konstantennamen zu natürlichen Zahlen übergeben werden. Immer wenn Sie bei der Auswertung auf eine Konstante stoßen, muss deren Wert in der Zuordnung nachgeschlagen und zurück gegeben werden. Kann keine Zuordnung gefunden werden, soll ein Fehler ausgegeben werden.

c) (Teil-)Ausdrücke, in denen keine Konstante vorkommt, können ohne Konstantenzuordnung ausgewertet werden. Implementieren Sie eine Methode (simplify t), die den übergebenen Ausdruck soweit wie möglich vereinfacht.
Beispiel: `(simplify '(((4 + (3 * 7)) - (a / (3 + 4))) + 6)` → `'((25 -(a / 7)) + 6)`

Hinweis: Für die Zuordnung von Variablennamen zu natürlichen Zahlen ist es hilfreich eine Struktur zu definieren, welche genau eine Variable zuordnet und der `eval`-Funktion eine Liste solcher Strukturen zu übergeben.
Bsp: `(evalTree t list((make-const 'x 5) (make-const 'y 7)))`

Aufgabe 7.2: Equivalent Exchange | 4 Punkte

```
; [X] (list-of X) (list-of X) -> (list-of X)
; concatenates two lists
(define (append x y)
        (cond [(empty? x) y]
              [(cons? x) (cons (first x)
                               (append (rest x) y))]))

; [X] (list-of X) -> (list-of X)
; reverse the order of elements in a list
(define (reverse x)
        (cond [(empty? x) empty]
              [(cons? x) (append (reverse (rest x))
                                 (cons (first x) empty))]))
```

Betrachten Sie die Implementierungen von append und reverse und zeigen Sie, dass gilt: `(reverse (append l1 l2))` $\equiv$ `(append (reverse l2) (reverse l1))`, für alle Listen l1 und l2.

Hinweis: Verwenden Sie die Aussagen zu `append` von Übungszettel 5 ohne Beweis und geben Sie zu jedem Schritt die Verwendeten Regeln an!

Aufgabe 7.3: Mergesort | 4 Punkte

Mergesort ist ein rekursiver Algorithmus zum Sortieren einer Liste (`https://de.wikipedia.org/wiki/Mergesort`). Die zu sortierende Liste wird dabei zunächst in zwei (gleich lange) Teile zerlegt (split). Diese Teile werden sortiert (rekursiver Aufruf) und anschließend wieder zu einer sortierten Liste verschmolzen (merge). Listen der Länge 1 bzw. 0 sind stets sortiert und bilden den Basisfall der Rekursion. In dieser Aufgabe sollen sie den Mergesort-Algorithmus implementieren:

a) Schreiben Sie eine Funktion `(split l)`, die die übergebene Listen in zwei (etwa) gleich lange Listen zerlegt und diese als Liste von Listen zurück liefert.

b) Schreiben Sie eine Funktion: `(mergeHelp l1 l2)`, die zwei sortierte Teillisten zu einer sortierten Gesamtliste zusammenführt.

c) Implementieren Sie nun die Funktion `(mergesort l)`, die die übergebene Liste wie beschrieben sortiert.

Hinweis: Sie können bei der Implementierung auf die eingebaute Funktion `(length l)` zurückgreifen, die die Länge einer Liste liefert.

Aufgabe 7.4: Institut für Seeverkehrswirtschaft und Logistik + | 4 Punkte

Werten Sie die folgenden Ausdrücke schrittweise und unter Angabe der verwendeten Regeln der ISL+ aus:

a)
```
(define mysq (lambda (x) (* x x)))
(mysq 8)
```

b)
```
(define x 5)
(define f (lambda (y) (local [(define x 3)]
        (+ x y))))
(f 3)
```

c)
```
(define addX (lambda (x)
        (local [(define y (lambda (y) (+ x y)))]
                y)))
((addX 5) 3)
```