VIII. Übungsblatt "Informatik I" WS 95/96

Christian Schulte, Prof. Gert Smolka

Ausgabe: 8. Dezember 1995

Abgabe: 15. Dezember 1995 vor der Vorlesung (Programme per E-Mail)

Aufgabe VIII.1 (Eliminierung von Let-Ausdrücken (8))
Erweiterte arithmetische Ausdrücke (siehe letztes Übungsblatt) können in einfache arithmetische Ausdrücke transformiert werden, wenn jeder Ausdruck `(let (`v A_1`)` A_2`)` durch einen Ausdruck ersetzt wird, den man durch Ersetzen jedes Auftretens der Variable v durch die Definition A_1 im Rumpf A_2 erhält. Zum Beispiel kann `(let (x (+ y z)) (+ 5 (* x z)))` zu `(+ 5 (* (+ y z) z))` transformiert werden.
Dabei muß jedoch das sogenannte *Kapern von Variablen* vermieden werden. Dazu betrachten wir den folgenden Ausdruck: `(let (x (* y 4)) (let (y 2) (* y x)))`, der die freie Variable `y` enthält. Eliminieren des äußeren Let ergibt `(let (y 2) (* y (* y 4)))`. Dieser Ausdruck ist nicht äquivalent zu dem vorherigen, da `y` nicht mehr frei auftritt. Wir sagen, daß die Variable `y` durch das innere Let *gekapert* wird: Aus einem freien Auftreten einer Variablen wird ein gebundenes.

1. Definieren Sie Konstruktoren für Summen und Produkte: `erz-summe` und `erz-produkt`.
2. Schreiben Sie eine Scheme Prozedur `(elim-let a)`, die zu einem erweiterten Ausdruck `a` einen äquivalenten Ausdruck ohne Let-Ausdrücke berechnet, wobei das Kapern von Variablen vermieden wird.
3. Schätzen Sie die Größe eines erweiterten Ausdrucks `a` im Verhältnis zur Größe des Ausdrucks ab, der durch das Elminieren von Let ensteht.

   Hinweis: Betrachten Sie als Beispiel den erweiterten Ausdruck:

   ```
   (let (x (let (y (* z z)) (* y y))) (* x x)).
   ```

Aufgabe VIII.2 (Generisches Sortieren durch Einfügen (3)) In der Vorlesung haben Sie gelernt, wie man Listen durch "Einfügen" sortiert. Üblicherweise braucht man aber auch Prozeduren, die Listen von Zahlen ihrer Größe nach sortieren, Listen von Listen der Länge nach, arithmetische Ausdrücke der Anzahl ihrer freien Variablen nach, usw. Eine Idee ist, daß man eine generische Prozedur `(sortiere xs o)` zum Sortieren von Listen schreibt, wobei die zweistellige Prozedur `o` die Ordnung für das Sortieren der Liste angibt.
So soll `(sortiere '(1 7 2) <)` zu `(1 2 7)` evaluieren, `(sortiere '(1 7 2) >)` zu `(7 2 1)` evaluieren, und
`(sortiere '((a 7) () (1)) (lambda (x y) (< (length x) (length y))))`
zu `(() (1) (a 7))` evaluieren.
Schreiben Sie eine entsprechende Prozedur `(sortiere xs o)`.

Aufgabe VIII.3 (Mischen sortierter Listen (3)) Gegeben seien zwei sortierte Listen (bezüglich einer Scheme Prozedur `o` wie in der vorherigen Aufgabe). Man kann eine sortierte Liste mit den Elementen beider Listen erhalten, indem man mit der leeren Liste beginnt und dieser Liste jeweils das kleinere (bezüglich der Ordnung `o`) der beiden ersten Elemente der Liste hinzufügt.

1. Schreiben Sie eine Prozedur `(mische xs ys o)`, die aus den bezüglich `o` sortierten Listen `xs` und `ys` eine sortierte Liste nach dem obigen Schema berechnet. Zum Beispiel soll `(mische '(1 3 6) '(2 2 4 5) <)` zu `(1 2 2 3 4 5 6)` evaluieren.

2. Geben Sie den Zeitaufwand der Prozedur `mische` in der O-Notation an.

3. Argumentieren Sie, warum unsere Beschreibung der durch die Prozedur `mische` implementierte Funktion nicht eindeutig ist.

Aufgabe VIII.4 (Sortieren durch Mischen (3)) Eine Idee, wie man Listen sortieren kann, ist die folgende:

- Die leere Liste ist sortiert.
- Eine Liste mit genau einem Element ist sortiert.
- Eine Liste mit mehr als einem Element wird in zwei annähernd gleich große Teillisten zerlegt. Diese werden getrennt sortiert und mit der Prozedur `mische` aus der vorherigen Aufgabe zu einer sortierten Liste zusammengesetzt.

1. Schreiben Sie eine Prozedur `(sortiere-mische xs o)`, die die Liste `xs` bezüglich der Ordnung `o` nach der oben angegebenen Methode sortiert.

2. Geben Sie den Zeitaufwand der Prozedur `sortiere-mische` in der O-Notation an.

Aufgabe VIII.5 (Huffman-Bäume (8))
In der Vorlesung haben Sie gelernt, wie man mit dem Huffman-Verfahren einen optimalen Codebaum zu einer gewichteten Symbolliste berechnet. Mehr Details zu diesem Thema finden Sie im Abschnitt 2.2.6 im Buch.
Anders als Sie es in der Vorlesung gesehen haben, werden Paare (z.B. gewichtete Symbole) in Scheme politisch korrekt als zweielementige Listen dargestellt (und *nicht* als Paare). Die Verwendung politisch inkorrekter Paare (z.B. `((1 . a) (4 . c))`) wird durch den Interpretierer mit kryptischer Ausgabe (z.B. `(1 a 4 . c)`) bestraft.

1. Schreiben Sie Konstruktoren und Selektoren für Codebäume und gewichtete Bäume in politisch korrekter Darstellung.

2. Schreiben Sie eine Prozedur `(huffman-baum paare)`, die zu einer Liste von gewichteten Symbolen `paare` einen optimalen Codebaum nach dem Huffman-Verfahren berechnet.

3. Definieren Sie eine Prozedur `(codiere symbole code)`, die eine Symbolliste `symbole` gemäß eines Codebaumes `code` in eine Bitliste codiert. Implementieren Sie dazu eine Prozedur `(symbol-codes code)`, die aus dem Codebaum `code` eine Liste von (politisch korrekten) Paaren aus Symbol und Bitliste berechnet. So soll `(symbol-codes '(a (b d)))` zu `((a (0)) (c (1 0)) (d (1 1)))` evaluieren.

4. Definieren Sie eine Prozedur `(decodiere bits code)`, die eine Bitliste `bits` gemäß eines Codebaumes `code` in eine Symbolliste decodiert.

5. Bevor man die obigen Verfahren überhaupt benutzen kann, muß ein gewichtetes Alphabet bekannt sein. Dieses gewichtete Alphabet kann man aus einer Beispielsymbolliste berechnen, indem man die Vorkommen jedes Symbols in der Liste zählt. Schreiben Sie eine Prozedur `(gewichte symbole)`, die zu einer Symbolliste `symbole` eine Liste gewichteter Symbole berechnet. So soll `(gewichte '(a b c b a a))` zu `((a 3) (b 2) (c 1))` evaluieren.

6. Geben Sie Gewichte für die Symbole `a`, `b`, `c`, `d` und `e`, so daß man mit dem Huffman-Verfahren den folgenden Codebaum berechnen kann: `((a b) ((c d) e))`.