2. Übungsblatt „Programmiersprachen“ WS 97/98

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 6. November 1997 **Abgabe:** 13. November 1997

Aufgabe II.1: Operationale Semantik von Scheme-1 ($1+2$ Punkte)

In der Vorlesung wurden nicht alle Transitionsregeln zur Definition der Semantik von Scheme-1 angegeben.

a. Geben Sie die Semantikregel für $\boxed{\texttt{*}}$ an.

b. Beschreiben Sie die Semantik von $\boxed{\texttt{<}}$ mit zwei Regeln (eine für `#t` und eine für `#f`).

Aufgabe II.2: Endrekursion (5 Punkte)

Betrachten Sie folgende Definition der Prozedur `append2` für zwei Argumente.

```
(define append2
  (lambda (l1 l2)
    (if (null? l1)
        l2
        (cons (car l1) (append2 (cdr l1) l2)))))
```

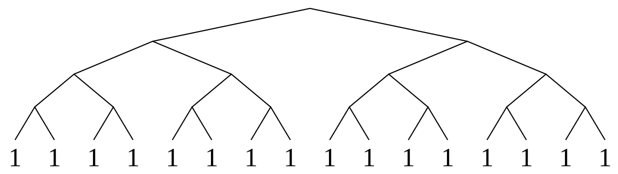
Diese Prozedur ist nicht endrekursiv. Schreiben Sie ein `append2` in Scheme-4, das endrekursiv ist. Dieses darf nur genauso viele cons-Zellen erzeugen, wie die erste Eingabeliste Elemente hat.

Aufgabe II.3: Zustand (4 Punkte)

Schreiben Sie eine Scheme-4-Prozedur `reverse!`, die als Eingabe eine Liste erhält und die Reihenfolge der Elemente in dieser Liste umdreht, ohne dabei neue cons-Zellen zu erzeugen. Rückgabe der Prozedur ist die umgedrehte Liste.

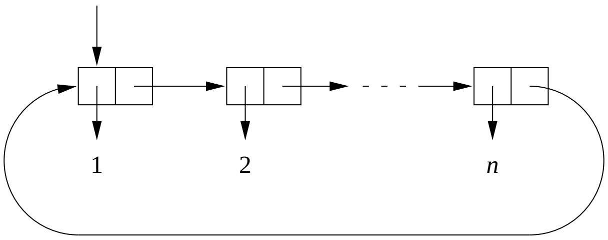
Aufgabe II.4: Bäume (3 Punkte)

Schreiben Sie einen Scheme-3-Ausdruck, der aus einer minimalen Anzahl von cons-Zellen eine Datenstruktur aufbaut, die bezüglich der Operationen `car` und `cdr` nicht von folgendem Baum unterschieden werden kann:



Aufgabe II.5: Zyklische Listen ($4+5$ Punkte)

a. Schreiben Sie eine Scheme-4-Prozedur `make-cycle`, die zu einem gegebenen Integer n eine Datenstruktur aufbaut, wie sie in folgendem Bild dargestellt ist:



Die ersten Komponenten der cons-Zellen sind die Zahlen von 1 bis n, die zweiten Komponenten verketten die Zellen zyklisch. Zurückgeben soll die Prozedur einen Zeiger auf die cons-Zelle mit erster Komponente 1.

Hinweis: Versuchen Sie nicht, SCM zyklische Datenstrukturen ausgeben zu lassen – die Ausgaberoutine gerät dabei in eine Endlosschleife.

b. Implementieren Sie ein Prädikat `is-cyclic-list?`, das überprüft, ob die ihr übergebene Liste zyklisch ist. Insbesondere soll `(is-cyclic-list? (cons 17 (cons 'a (make-cycle 7))))` zu `#t` auswerten. Verwenden Sie hierzu das Prädikat `eq?`, das – auf cons-Zellen angewendet – genau dann `#t` liefert, wenn die beiden Argumente dieselbe Zelle im Speicher bezeichnen.

Hinweis: Dieser Test ist mit linearer Komplexität möglich. Allerdings werden auch ineffizientere Lösungen akzeptiert.

Aufgabe II.6: Quotierung (5 Punkte)

Gegeben sei folgender Scheme-Ausdruck:

```
(quote (define Q
         (lambda (x)
           (* x x))))
```

Schreiben Sie einen Scheme-3-Ausdruck A, der zu demselben Wert wie obiger Ausdruck ausgewertet wird. Der Ausdruck A darf kein `let` oder `letrec` enthalten. Weiterhin darf A nur fünf Verwendungen von `quote` für Symbole enthalten und das verallgemeinerte Quotieren nicht verwenden.

Aufgabe II.7: Selbstapplikation (4 Punkte)

Es ist eine Scheme-3-Prozedur `count-leaves` mit einem Argument zu schreiben, die die Anzahl der Blätter in dem ihr übergebenen Baum zählt und zurückgibt. Beispielsweise hat der Baum `(quote (1 . (2 . ((3 . 4) . 5))))` fünf Blätter.

Verwenden Sie dabei keine Rekursion, sondern Selbstapplikation.