Musterlösung X. Übungsblatt "Informatik I"
WS 95/96

Stephan Bergmann

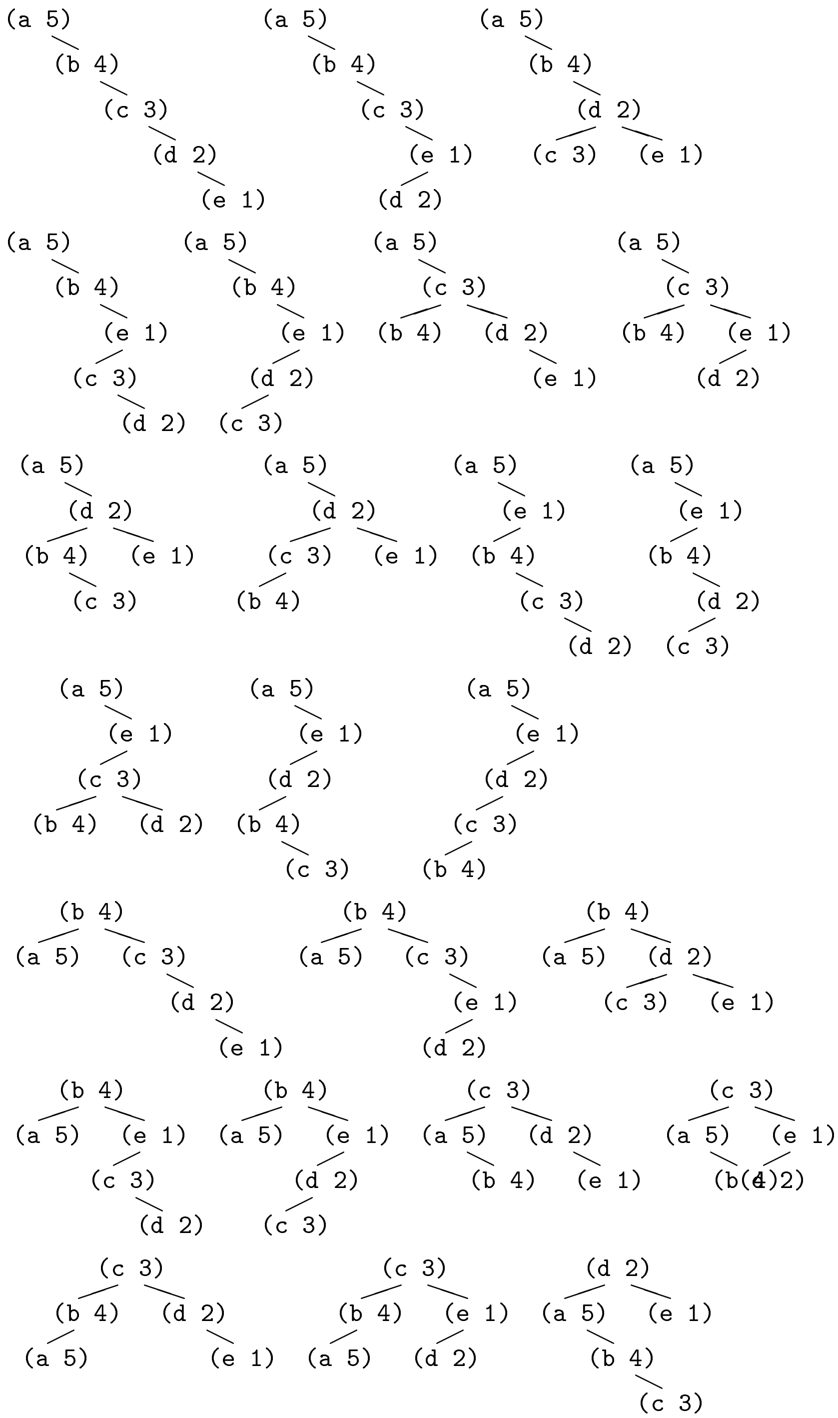
Aufgabe X.1 (Dictionaries)

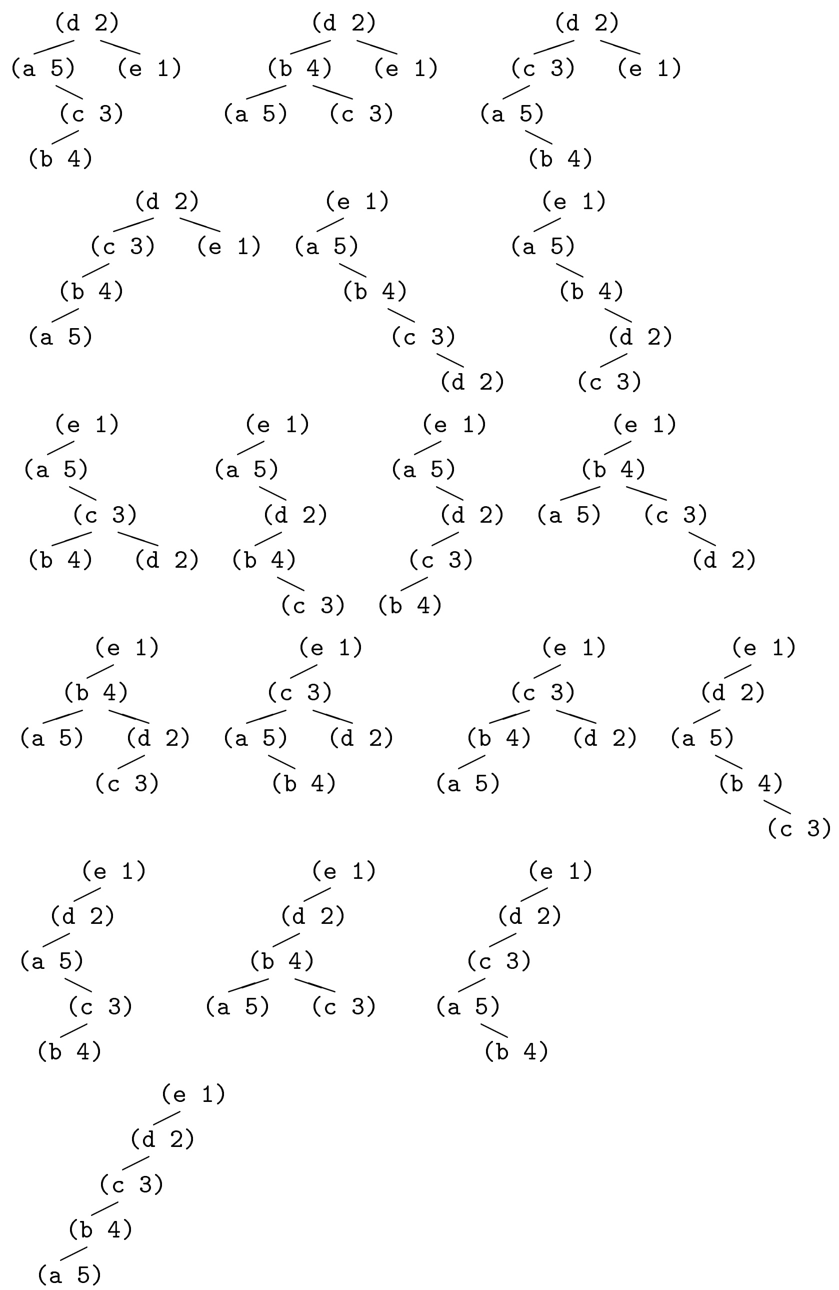
```
(define (symbol<? a b) (string<? (symbol->string a) (symbol->string b)))

(define (dictionary)
  (define d ())
  (define make-entry cons)
  (define get-key car)
  (define get-value cdr)
  (define set-value! set-cdr!)
  (define (put s x)
    (define (put1 dd)
      (cond ((eq? s (get-key (car dd))) (set-value! (car dd) x))
            ((or (null? (cdr dd)) (symbol<? s (get-key (cadr dd))))
             (set-cdr! dd (cons (make-entry s x) (cdr dd))))
            (else (put1 (cdr dd)))))
    (if (or (null? d) (symbol<? s (get-key (car d))))
        (set! d (cons (make-entry s x) d))
        (put1 d)))
  (define (get s)
    (define (get1 dd)
      (cond ((or (null? dd) (symbol<? s (get-key (car dd)))))
            ((eq? s (get-key (car dd))) (get-value (car dd)))
            (else (get1 (cdr dd)))))
    (get1 d))
  (define (remove s)
    (define (remove1 dd)
      (cond ((or (null? (cdr dd)) (symbol<? s (get-key (cadr dd)))))
            ((eq? s (get-key (cadr dd))) (set-cdr! dd (cddr dd)))
            (else (remove1 (cdr dd)))))
    (cond ((or (null? d) (symbol<? s (get-key (car d)))))
          ((eq? s (get-key (car d))) (set! d (cdr d)))
          (else (remove1 d))))
  (define (in? s)
    (define (in1? dd)
      (cond ((or (null? dd) (symbol<? s (get-key (car dd)))) #f)
            ((eq? s (get-key (car dd))) #t)
            (else (in1? (cdr dd)))))
    (in1? d))
  (lambda (message)
    (cond ((eq? message 'put) put)
          ((eq? message 'get) get)
          ((eq? message 'remove) remove)
          ((eq? message 'in?) in?))))
```

Aufgabe X.2 (Binäre Suchbäume)

1. Die möglichen binären Suchbäume (42 Stück):





2. Die Prozedure `fuege-ein`:

```
(define (make-stree key value left right) (list key value left right))
(define (make-empty-stree) (list () () () ()))
(define (is-empty-stree? b) (null? (get-key b)))
(define (change-to-empty-stree b) (set-key! b '()))
(define (change-to-nonempty-stree b s x)
  (set-key! b s)
  (set-value! b x)
```

```
  (set-car! (cddr b) (make-empty-stree))
  (set-car! (cdddr b) (make-empty-stree)))
(define get-key car)
(define get-value cadr)
(define get-left caddr)
(define get-right cadddr)
(define set-key! set-car!)
(define (set-value! b x) (set-car! (cdr b) x))

(define (fuege-ein b s x)
  (cond ((is-empty-stree? b) (change-to-nonempty-stree b s x))
        ((symbol<? s (get-key b)) (fuege-ein (get-left b) s x))
        ((eq? s (get-key b)) (set-value! b x))
        (else (fuege-ein (get-right b) s x))))
```

3. Dictionaries mit binären Suchbäumen:

```
(define (dictionary2)
  (define d (make-empty-stree))
  (define (put s x) (fuege-ein d s x))
  (define (get s)
    (define (get1 dd)
      (cond ((is-empty-stree? dd))
            ((symbol<? s (get-key dd)) (get1 (get-left dd)))
            ((eq? s (get-key dd)) (get-value dd))
            (else (get1 (get-right dd)))))
    (get1 d))
  (define (remove s)
    (define (remove1 dd s)
      (define (extract-leftmost dd)
        (if (is-empty-stree? (get-left dd))
            (let ((key-value-pair (cons (get-key dd) (get-value dd))))
              (remove1 dd (get-key dd))
              key-value-pair)
            (extract-leftmost (get-left dd))))
      (define (extract-rightmost dd)
        (if (is-empty-stree? (get-right dd))
            (let ((key-value-pair (cons (get-key dd) (get-value dd))))
              (remove1 dd (get-key dd))
              key-value-pair)
            (extract-rightmost (get-right dd))))
      (cond ((is-empty-stree? dd))
            ((symbol<? s (get-key dd)) (remove1 (get-left dd) s))
            ((eq? s (get-key dd))
             (if (is-empty-stree? (get-left dd))
                 (if (is-empty-stree? (get-right dd))
                     (change-to-empty-stree dd)
                     (let ((key-value-pair (extract-leftmost (get-right dd))))
                       (set-key! dd (car key-value-pair))
                       (set-value! dd (cdr key-value-pair))))
```

```
                  (let ((key-value-pair (extract-rightmost (get-left dd))))
                    (set-key! dd (car key-value-pair))
                    (set-value! dd (cdr key-value-pair)))))
             (else (remove1 (get-right dd) s))))
     (remove1 d s))
   (define (in? s)
     (define (in1? dd)
       (cond ((is-empty-stree? dd) #f)
             ((symbol<? s (get-key dd)) (in1? (get-left dd)))
             ((eq? s (get-key dd)) #t)
             (else (in1? (get-right dd)))))
     (in1? d))
   (lambda (message)
     (cond ((eq? message 'put) put)
           ((eq? message 'get) get)
           ((eq? message 'remove) remove)
           ((eq? message 'in?) in?))))
```

Aufgabe X.3 (Priority-Queues)

```
(define (prio-queue)
  (define q '((-1 . ())))
  (define make-entry cons)
  (define get-priority car)
  (define get-value cdr)
  (define (q-null?) (eq? (get-priority (car q)) -1))
  (define (dequeue)
    (if (not (q-null?))
        (let ((value (get-value (car q))))
          (set! q (cdr q))
          value)))
  (define (enqueue p x)
    (define (enqueue1 qq)
      (if (<= p (get-priority (car qq))) (enqueue1 (cdr qq))
          (begin (set-cdr! qq (cons (car qq) (cdr qq)))
                 (set-car! qq (make-entry p x)))))
    (enqueue1 q))
  (lambda (message)
    (cond ((eq? message 'null?) q-null?)
          ((eq? message 'dequeue) dequeue)
          ((eq? message 'enqueue) enqueue))))
```

Aufgabe X.4 (Semantikbäume für Scheme 2) Seien U_0 und Z_0 die Umgebung und der Zustand, in denen nur die Namen der primitiven Prozeduren auf ebendiese Prozeduren abgebildet werden.

1. Der erste Ausdruck evaluiert unter U_0 und Z_0 zum Wert 4 und zum Zustand $Z_0[(l_1, l_2)/l_0, \texttt{a}/l_1, \boxed{\texttt{undef}}/l_2]$, wobei die l_i „frische“ Lokationen sind. Der Semantikbaum mit den angewandten Regeln:

- Deklaration
  - Sequentialisierung
    - Zuweisung
      - Applikation
        - Namen
        - Konstanten
        - Namen
        - Primitive Prozeduren
    - Konditional
      - Applikation
        - Namen
        - Namen
        - Applikation
          - Namen
          - Namen
          - Primitive Prozeduren
        - Primitive Prozeduren
      - Konstanten

2. Der zweite Ausdruck kann unter U_0 und Z_0 nicht evaluiert werden und hat somit auch keinen Semantikbaum.

Aufgabe X.5 (Zirkuläre Listen)

```
(define (circular? l)
  (define (circular1? r1 r2)
    (cond ((or (null? (cdr r2)) (null? (cddr r2))) #f)
          ((eq? r1 r2) #t)
          (else (circular1? (cdr r1) (cddr r2)))))
  (and (not (null? l)) (not (null? (cdr l))) (circular1? l (cdr l))))
```