Scheme

New procedures

1. \((\text{cons } a \ b)\) - Makes a cons-cell (pair) from a and b
2. \((\text{car } c)\) - extracts the value of the first part of the pair
3. \((\text{cdr } c)\) - extracts the value of the second part of the pair
4. \((\text{cadr } x)\) - shortcuts. \((\text{cadr } x)\) is the same as \((\text{car } (\text{cdr } x))\)
5. \((\text{list } a \ b \ c \ ... )\) - builds a list of the arguments to the procedure
6. \((\text{define nil } '()\) - the special object '(), called the empty list, denotes the end of a list. We often write this as nil instead of '().
7. \((\text{null? } a)\) - returns #t if a is the empty list (nil or '()), and #f otherwise.

Problems

1. Draw box-and-pointer diagrams for the values of the following expressions. Also give the printed representation.

(a) \((\text{cons } 1 \ 2)\)

(b) \((\text{cons } 1 \ (\text{cons } 3 \ (\text{cons } 5 \ '())))\)

(c) \((\text{cons } (\text{cons } (\text{cons } 3 \ 2) \ (\text{cons } 1 \ 0)) \ '())\)
(d) (cons 0 (list 1 2))
(0 1 2)

(e) (list (cons 1 2) (list 4 5) 3)
((1 . 2) (4 5) 3)

2. Write expressions whose values will print out like the following.

(a) (1 2 3)
    (list 1 2 3) or (cons 1 (cons 2 (cons 3 '())))
(b) (1 2 . 3)
    (cons 1 (cons 2 3))
(c) ((1 2) (3 4) (5 6))
    (list (list 2 3) (list 3 4) (list 5 6))

3. Create a data abstraction for points in a plane. It should have a constructor, (make-point x y), which returns a point, and two selectors (point-x pt) and (point-y pt), which return the x and y coordinates.

(define (make-point x y)
  (list x y))
(define (point-x pt)
  (car pt))
(define (point-y pt)
  (cadr pt))

4. Now, extend the point abstraction to handle line segments, with a constructor (make-line-segment pt1 pt2), and selectors line-segment-start and line-segment-end.

(define (make-line-segment pt1 pt2)
  (cons pt1 pt2))
(define (line-segment-start pt)
  (car pt))
(define (line-segment-end pt)
  (cadr pt))
5. Write a procedure (\texttt{intersection \textit{seg1 \textit{seg2}}}) that returns a point where two line segments intersect if they do, and returns \#f if they do not intersect. Be sure to honor the abstractions defined.

\begin{verbatim}
(define (intersection \textit{s1 \textit{s2}})
  (let ((\textit{x1} (point-x (line-segment-start \textit{s1})))
         (\textit{x2} (point-x (line-segment-end \textit{s1})))
         (\textit{x3} (point-x (line-segment-start \textit{s2})))
         (\textit{x4} (point-x (line-segment-end \textit{s2})))
         (\textit{y1} (point-y (line-segment-start \textit{s1})))
         (\textit{y2} (point-y (line-segment-end \textit{s1})))
         (\textit{y3} (point-y (line-segment-start \textit{s2})))
         (\textit{y4} (point-y (line-segment-end \textit{s2}))))
    (let ((\textit{n1} (- (* (- \textit{x4} \textit{x3}) (- \textit{y1} \textit{y3}))
                         (* (- \textit{y4} \textit{y3}) (- \textit{x1} \textit{x3}))))
          (\textit{n2} (- (* (- \textit{x2} \textit{x1}) (- \textit{y1} \textit{y3}))
                         (* (- \textit{y2} \textit{y1}) (- \textit{x1} \textit{x3}))))
          (\textit{d} (- (* (- \textit{y4} \textit{y3}) (- \textit{x2} \textit{x1}))
                         (* (- \textit{x4} \textit{x3}) (- \textit{y2} \textit{y1}))))
    (if (zero? \textit{d}) #f
        (let ((\textit{u1} (/ \textit{n1} \textit{d}))
              (\textit{u2} (/ \textit{n2} \textit{d})))
          (if (or (< \textit{u1} 0) (< \textit{u2} 0)
                     (> \textit{u1} 1) (> \textit{u2} 1))
              #f
              (make-point (+ \textit{x1} (* \textit{u1} (- \textit{x2} \textit{x1})))
                          (+ \textit{y1} (* \textit{u1} (- \textit{y2} \textit{y1}))))))))
\end{verbatim}