Assignment 1

Principles of Programming Languages
II UWr, 2017/18

due October 18, 2017
(but your solutions will be accepted also on October 25)

Problem 1 (2 pts). Solve Exercise 1.21 from EoPL.

Problem 2 (2 pts). Solve Exercise 1.26 from EoPL.

Problem 3 (2 pts). Solve Exercise 1.28 from EoPL.

Problem 4 (2 pts). Solve Exercise 1.30 from EoPL.

Problem 5 (3 pts). Solve Exercise 1.34 from EoPL.

Problem 6 (3 pts). Define a function `exp` such that `((exp k) n)` returns the number representing `n` to the power `k`, but in such a way that `(exp k)` generates code that does not refer to `k`. So, for instance, the following definition is not what you are looking for:

```
(define (exp k)
  (lambda (n)
    (if (= k 0)
        1
        (* n ((exp (- k 1)) n)))))
```

Problem 7 (optional, 2 pts). Fill in the missing fragments in the following code, so that the resulting definition is an implementation of the factorial function.

```
(define fact
  (let ((f (lambda (g)
             (lambda (n)
               (if (= n 0)
                   1
                   (* n (□ (- n 1))))))))
    □))
```

You should use neither explicit recursion nor mutable state.

Problem 8 (optional, 3 pts). Implement a function `fix` such that

```
(define fact
  (fix fact-maker))
```

defines the factorial function, where

```
(define fact-maker
  (lambda (h)
    (lambda (n)
      (if (= n 0)
          1
          (* n (h (- n 1)))))))
```

You should use neither explicit recursion nor mutable state. A solution to this problem can be derived from the solution to Problem 7.

Problem 9 (optional, 2 pts). Prove that it is impossible to define in Racket a function `halts-for-nil` such that for all functions `f`, `(halts-for-nil f)` returns `#t` if `f` halts for `'()`, and returns `#f` otherwise. To this end, first define a one-argument function `loops-forever` that does not terminate for any argument. Then, define a function `contradiction` such that `(halts-for-nil contradiction)` yields `#t` if and only if `(contradiction '())` loops forever.