Problem Set 2
COMP301 Fall 2019
10.10.2019 17:30 - 18:45

Problem 1[1]: Implement the following procedures for handling lambda-calculus expressions.

- Constructors:
  `var-exp`: $Var \rightarrow LcExp$
  `lambda-exp`: $Var \times LcExp \rightarrow LcExp$
  `app-exp`: $LcExp \times LcExp \rightarrow LcExp$
- Predicates:
  `var-exp?`: $LcExp \rightarrow Bool$
  `lambda-exp?`: $LcExp \rightarrow Bool$
  `app-exp?`: $LcExp \rightarrow Bool$
- Extractors:
  `var-exp->var`: $LcExp \rightarrow Var$
  `lambda-exp->bound-var`: $LcExp \rightarrow Var$
  `lambda-exp->body`: $LcExp \rightarrow LcExp$
  `app-exp->rator`: $LcExp \rightarrow LcExp$
  `app-exp->rand`: $LcExp \rightarrow LcExp$

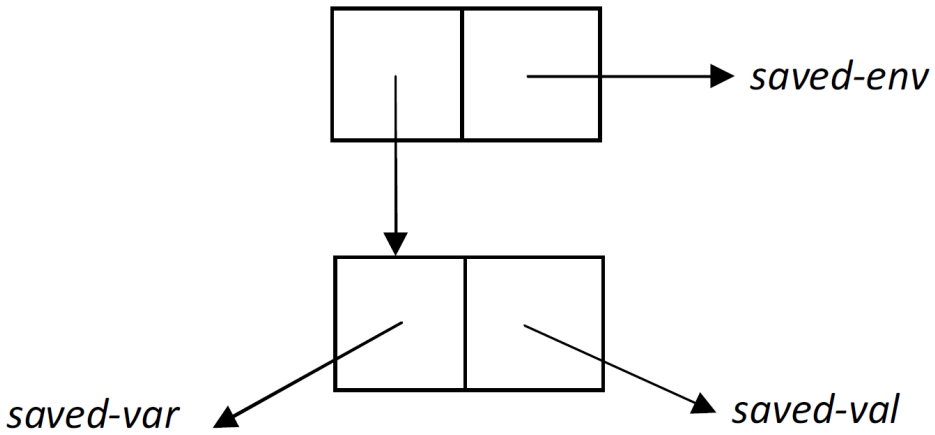
Problem 2[2]: Implement the four required operations (`predecessor, successor, addition, multiplication`) and `is-zero?` for bigits. Then use your implementation to calculate the factorial of 10. How does the execution time vary as this argument changes? How does the execution time vary as the base changes? Explain why. (EOPL p.34 has information regarding Bigit.) As an example: in base 10, `(factorial '(4))` would return the list `(4 2)` because $4! = 24 = 4 * 10^0 + 2 * 10^1$.

Problem 3[3]: Consider the data type of *stacks* of values, with an interface consisting of the procedures: `empty-stack`, `push`, `pop`, `top`, and `empty-stack?`. Write a specification for these operations in the style of the example above (see Problem 1). Which operations are constructors and which are observers?

---

[1] EOPL p.42-43
[2] EOPL p.34 Exercise 2.1
[3] EOPL p.37 Exercise 2.4

Problem 4[4]: We can use any data structure for representing environments, if we can distinguish empty environments from non-empty ones, and if we can can extract the pieces of a non-empty environment. Implement environments using a representation in which the empty environment is represented as the empty list, and in which extend-env builds an environment that looks like:



This is called an *a-list* or *association-list* representation.

Problem 5[5]: Add to the environment interface (from Problem 4) an observer called `has-binding?` that takes an environment *env* and a variable *s* and tests to see if *s* has an associated value in *env*. Implement it using the *a-list* representation.

[4]EOPL p.39 Exercise 2.5
[5]EOPL p.39 Exercise 2.9