Tarea 06 - EXPLICIT REFS

Enrique Giottonini, Miguel Navarro

Octubre 26, 2022

Exercise 4.8 [⋆]

Show exactly where in our implementation of the store these operations take linear time rather than constant time.

- En *newref-exp* se utiliza *append* el cual tiene un tiempo de ejecución dependiente de la cantidad de todos los argumentos en la lista (exceptuando el último) por lo que new-ref tiene tiempo de ejecución lineal. Esto siempre sucede.

- En *deref-exp* se utiliza *list-ref*, el cual tiene un tiempo de ejecución lineal por lo que *deref-exp* tiene un tiempo de ejecución lineal. Esto sucede siempre y cuando el *store* no es vacío y la referencia es válida.

- En *setref-exp* se utiliza *list-set* el cual tiene un tiempo de ejecución lineal por lo que *setref-exp* tiene tiempo de ejecución lineal. Esto sucede siempre y cuando el *store* no es vacío y la referencia es válida.

Sintáxis Concreta y Abstracta

Expression ::= ~~proc (Identifier) Expression~~
Expression ::= letproc Identifier (Identifier) Expression in Expression

(letproc-exp name param body exp1)

Semántica

```
(value-of (letproc-exp name param body exp1) env) =
(value-of exp1 ([name = (procedure param body env)] env))
```

Exercise 4.9 [⋆]

Implement the store in constant time by representing it as a Scheme vector. What is lost by using this representation?

Exercise 4.10 [⋆]

Implement the begin expression as specified in exercise 4.4.

Exercise 4.11 [⋆]

Implement list from exercise 4.5.

Exercise 4.12 [⋆⋆⋆]

Our understanding of the store, as expressed in this interpreter, depends on the meaning of effects in Scheme. In particular, it depends on us knowing when these effects take place in a Scheme program. We can avoid this dependency by writing an interpreter that more closely mimics the specification. In this interpreter, value-of would return both a value and a store, just as in the specification. A fragment of this interpreter appears in figure 4.6. We call this a store-passing interpreter. Extend this interpreter to cover all of the language EXPLICIT-REFS. Every procedure that might modify the store returns not just its usual value but also a new store. These are packaged in a data type called answer. Complete this definition of value-of.

```
(newref-exp (exp1)
  (let ((v1 (value-of exp1 env)))
       (ref-val (newref v1))))

(deref-exp (exp1)
  (let ((v1 (value-of exp1 env)))
    (let ((ref1 (expval->ref v1)))
      (deref ref1))))

(setref-exp (exp1 exp2)
  (let ((ref (expval->ref (value-of exp1 env))))
    (let ((val2 (value-of exp2 env)))
      (begin
        (setref! ref val2)
        (num-val 23)))))
```

Exercise 4.13 [⋆⋆⋆]

Extend the interpreter of the preceding exercise to have procedures of multiple arguments.