Department of Computer Science and Engineering
University of California at San Diego

CSE 230
Winter 1998

# Problem Set 7

DUE TUESDAY FEBRUARY 24 IN CLASS.

1. Provide in ML an abstract type for scalar physical quantities such as mass or angular momentum which have magnitudes and dimensions in SI units. Implement a pure abstract type (one where values are not updatable) and use the ML module construct, not the ML `abstype` construct, which is obsolete. Discuss briefly the design decisions you make.

   Note that there are several different so-called metric systems, but only one international standard metric system. This system of units is called SI, which stands for *Système International* in French. For information on SI see `http://www.kumc.edu/ASP/si_units.html/`

2. A Thcheme meta-interpreter is an interpreter for Thcheme written in Thcheme. Dr. Lisp says "Thcheme themanticth ith tho eathy," because he believes that the meanings of programs written in Thcheme can be defined by a Thcheme meta-interpreter. Unfortunately, Dr. Lisp is wrong, because most Thcheme meta-interpreters are ambiguous.

   Consider the Thcheme meta-interpreter below. Assume that the meaning of all predefined functions such as `car` and of all syntactic forms such as `(if` $\mathcal{C}$ $\mathcal{X}$ $\mathcal{Y}$`)` is well-defined, including whether they are strict and/or lenient. Explain why the meta-interpreter nevertheless does not specify the semantics of Thcheme unambiguously.

```
(define eval (exp envt fundefs)
    (if (number? exp) exp
    (if (symbol? exp) (assoc exp envt)
    (if (= (car exp) 'quote) (cadr exp)
    (if (= (car exp) 'if)
        (if (null? (eval (cadr exp) envt fundefs))
            (eval (cadddr exp) envt fundefs)
            (eval (caddr exp) envt fundefs))
    (if (userfun? (car exp) fundefs)
        (apply-userfun
            (assoc (car exp) fundefs)
            (evallist (cdr exp) envt fundefs)
            fundefs)
    (if (= (length exp) 2)
        (apply-unary-op (car exp) (eval (cadr exp) envt fundefs))
        (apply-binary-op
            (car exp)
            (eval (cadr exp) envt fundefs)
            (eval (caddr exp) envt fundefs))))))))
```

The code above is taken from *Programming Languages: An Interpreter-Based Approach* by Samuel N. Kamin (Addison-Wesley, 1990), p. 49. The three arguments of `eval` are an expression to be evaluated, an environment `envt` of variable bindings, and a separate environment `fundefs` of closures for user-defined functions. Environments are represented as association lists, that is as lists of name–value pairs. Auxiliary functions are defined below; their details are not important for this question. The notation `(cadar alist)` means `(car (cdr (car alist)))` and other combinations of `car` and `cdr` are defined similarly.

[The following notes on scoping are due to Nicholas Mitchell.] While using this `eval` function, the `fundefs` list of closures is never changed. This means that all functions are top-level globals, as in C, so static and dynamic scoping are indistinguishable for identifiers denoting functions: there is never any change in the bindings of these identifiers while using `eval`.

Other variables have to be parameters since there is no syntax for local declarations. Looking at `apply-userfun` we see a further restriction: closures in this language in fact do not contain any saved environment. The `car` of `fundef` is a list of formal parameter names, the `cadr` is a body expression, and there is nothing else in `fundef`. So, these user-defined functions cannot refer to any identifiers except their own formal parameters. A function's own formal parameters behave in the same way under dynamic scoping and under static scoping, so again the scoping discipline is unidentifiable but irrelevant for this language.

If Thcheme had full static scoping then `fundef` would have a third component and the `mkassoc*` call would read

```
(mkassoc* (car fundef) args (caddr fundef))
```

This would ensure that the new bindings between formal parameters and actual arguments were *pre*-pended to the saved environment. Prepending ensures that in case of a name conflict, `assoc` finds the newly created binding first, which gives the effect of inner declarations overriding outer declarations.

If Thcheme had full dynamic scoping then `apply-userfun` would take a fourth argument `envt` and the `mkassoc*` call would read `(mkassoc* (car fundef) args envt)`.

```
(define userfun? (f fundefs) (assoc f fundefs))

(define apply-userfun (fundef args fundefs)
    (eval (cadr fundef)
          (mkassoc* (car fundef) args '())
          fundefs))

(define evallist (el rho fundefs)
    (if (null? el) '()
        (cons (eval (car el) rho fundefs)
              (evallist (cdr el) rho fundefs))))

(define apply-binary-op (f x y)
      (if (= f 'cons) (cons x y)
```

```
      (if (= f '+) (+ x y)
      (if (= f '-) (- x y)
      (if (= f '*) (* x y)
      (if (= f '/) (/ x y)
      (if (= f '<) (< x y)
      (if (= f '>) (> x y)
      (if (= f '=) (= x y) 'error))))))))

(define apply-unary-op (f x)
      (if (= f 'car) (car x)
      (if (= f 'cdr) (cdr x)
      (if (= f 'number?) (number? x)
      (if (= f 'list?) (list? x)
      (if (= f 'symbol?) (symbol? x)
      (if (= f 'null?) (null? x) 'error)))))))

(define assoc (x alist)
     (if (null? alist) '()
         (if (= x (caar alist)) (cadar alist)
             (assoc x (cdr alist)))))

(define mkassoc* (keys values alist)
     (if (null? keys) alist
         (mkassoc* (cdr keys) (cdr values)
                   (mkassoc (car keys) (car values) alist))))

(define mkassoc (x y alist)
     (if (null? alist) (list1 (list2 x y))
         (if (= x (caar alist)) (cons (list2 x y) (cdr alist))
             (cons (car alist) (mkassoc x y (cdr alist))))))
```