

Program 2

Practice in functional programming.

If you are used to strongly typed languages, Scheme’s untyped λ -lists might seem unnerving. But it doesn’t have to be that way. We can do some more useful runtime type checking in a way that makes our code a little clearer. For example, instead of something like:

```
(define cube (lambda (x) (* x x x)))
```

we can put together a “`tlambda`” for *typed* λ -lists, allowing us to write code like this:

```
(define cube (tlambda ((number? x)) (* x x x)))
```

That is, a `tlambda`-list can have both untyped parameters (like λ) and typed parameters. A typed parameter is of the form:

```
( [list-of] predicate var )
```

where *predicate* evaluates to a predicate that must be true for *var* to be a meaningful parameter. This predicate may be produced by functionals (like `and~`, `not~` etc.) applied to *other* predicates! Since many Scheme procedures operate on lists of a given type, we’ve also introduced a `list-of` keyword as a way to declare lists with a given element type.

For this assignment, you will write a procedure that translates “`tlambda`” into ordinary Scheme code. For example, the `tlambda` code for `cube`, above, could be mechanically translated into ordinary Scheme as:

```
(define cube
  (lambda (x)
    (if (number? x)
        (* x x x)
        "parameter type mismatch")))
```

Here are some of the procedures in the *Scheme Warm-up Exercises* handout, re-written with `tlambda`:

```
(define sin5
  (tlambda ((real? x)
            ...))
(define all-equal
  (tlambda ((list? s)
            ...))
(define fac
  (tlambda ((integer? n)
            ...))
(define gen-consec
  (tlambda ((and~ positive? integer?) n)
            ...))
(define average
  (tlambda ((list-of number? s)
            ...))
(define multi-iter
  (tlambda ((procedure? f) ((and~ positive? integer?) n)
            ...))
```

The slickest way to do this uses Scheme’s ability to define macros, but that is beyond the scope of our little two-week crash course. Instead, we will do it with a function. You will write a function `ttrans` that (a) translates function definitions from `tlambda` form into `lambda` form, and (b) executes the resulting definition.

So if you wanted to define `sin5`, as above, you would do this way: [note the quote! why?]

```
(ttrans
 '(define sin5
  (tlambda ((real? x)
            ...)))
```

Your `ttrans` may assume the definition is syntactically well-formed. The `tlambda` should handle any number (0 or more) typed or untyped parameters. A more specific error message than “parameter type mismatch” is strongly recommended, but not required. Comment judiciously, and follow Scheme conventions. Please restrict yourself to our “basic Scheme vocabulary”; ask in class if you feel you need something more. Remember to work incrementally: pick an easy special case first, and gradually make your `ttrans` more powerful.

Submit your work in the file `tlambda.scm`. This is an individual assignment. **Due** Thursday September 26.

The screenshot shows the DrScheme IDE window titled "p2scratch.scm - DrScheme". The menu bar includes "File", "Edit", "Windows", "Show", "Language", "Scheme", and "Help". The toolbar contains buttons for "p2scratch.scm", "(define ...)", "Save", "Check Syntax", "Step", "Execute", and "Break".

```
(map ttrans '(  
  
  (define cube  
    (tlambda ((number? x))  
      (* x x x)))  
  
  (define first  
    (tlambda ((pair? s))  
      (car s)))  
  
  ; ...  
'))
```

The execution results show:

```
> (cube 4)  
64  
> (cube '(a b c))  
(#<procedure:cube> : parameter (a b c) type mismatch)  
> (first '(a b c))  
a  
> (first 13098)  
(#<procedure:first> : parameter 13098 type mismatch)  
>
```

At the bottom of the window, there is a status bar showing the time "23:72", the state "Unlocked", and "not running".

Note: Of course, Scheme catches type mismatches at runtime already, but only in the lowest-level built-in functions. With `tlambda`, we enable mismatches to be caught right where a precondition is most meaningful, not buried in the call stack. And, of course, it also allows programmers to read the preconditions directly in the code.