4.2 Type Checking

- Recall type of **`if`** statement:

`(type-of-expression` *<<test-exp>> tenv*`)` = **`bool`**

`(type-of-expression` *<<true-exp>> tenv*`)` = *x*

`(type-of-expression` *<<false-exp>> tenv*`)` = *x*

`(type-of-expression`

`if` *<<test-exp* **`then`** *true-exp* **`else`** *false-exp>> tenv*`)` = *x*

4.2 Type Checking

- Will want to check the type of all expressions.
- Some expressions (**`proc`**, **`let`**/**`letrec`**) will need to declare types.
- Others (**`if`**, application) will know about types from sub-expressions.
- Language of this chapter is functional, not OO.

- Recall grammar for type declarations:

  <type-exp> ::= **int**

  int-type-exp ()

  <type-exp> ::= **bool**

  bool-type-exp () *e.g.*, **(int*int) -> bool**

  <type-exp> ::= **(**{<type-exp>}*(*) **->** <type-exp>**)**

  proc-type-exp (arg-texps result-texp)

- Now can write grammar for typed language:

Grammar for Typed Language

<expression> ::= **true**

true-exp

<expression> ::= **false**

false-exp

<expression> ::= **proc (**{<type-exp> <identifier>}*(,)**)**
<expression>

proc-exp (arg-texps ids body)

Grammar for Typed Language

<expression> ::= **letrec**

{<type-exp> <identifier>

({ <type-exp> <identifier>}*(,)**)**
= <expression>}*

in <expression>

```
letrec-exp
    (result-texps proc-names
     arg-texpss idss bodies
     letrec-body)
```

Grammar for Typed Language

- Examples:

```
(1) proc (int x) add1(x)


(2) letrec
        int fact (int x) =
            if zero?(x) then 1
            else *(x, (fact sub1(x)))
        in (fact 3)
```

- Q: Why don't we need to say **`int proc`** in (1) ?

Data Structures for Typed Language

- Need a data structure for types.
- Recall that types are either primitives (**`int`**, **`bool`**) or procedures/functions *over* types: **`int->bool`**

```
(define-datatype type type?
  (atomic-type
    (name symbol?)) ; int, bool, ...
  (proc-type
    (arg-types (list-of type?))
    (result-type type?)))
```

- Now can define as many primitive types as we like:

```
(define int-type (atomic-type 'int))
(define bool-type (atomic-type 'bool))
```

- Checking type equality can be done using **`equal?`**

```
(define check-equal-type!
  (lambda (t1 t2)
    (if (not (equal? t1 t2))
      (eopl:error ...))))
```

- Q: why `!` in **`check-equal-type!`**
- Now can write **`type-of-expression`** :

```
(define type-of-expression
  (lambda (exp tenv)
    (cases expression exp
      (lit-exp (number) int-type)
      (true-exp () bool-type)
      (false-exp () bool-type)
      (var-exp (id) (apply-tenv tenv id))
```