Midterm Results

- Average: 87
- Median: 88

```
1
0 9 9 9 9 9 9 9 9 9 9 8 8 8 8 8 8 8 8 8 8 7 7 7 7 7 7 7 7 7 7 6 6 6 6 6 6 6 ...
0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 ...
************* ************* *************  ** **         * *              * ***
 *****  *** *  * * **     *  * *   **  **   * *
  *     ***                  * *   *    *
                             *     *
                             *
```

- Most of the variance was from question 6, and there will be another one like it on the final

Quiz

- Question #1: What is the value of the following expression?

  +(1,1)

- Wrong answer: **0**
- Wrong answer: **42**
- Answer: **2**

Quiz

- Question #2: What is the value of the following expression?

  + **proc** 8

- Wrong answer: **error**
- Answer: Trick question! + **proc** 8 is not an expression

Language Grammar for Quiz

```
<expr>  ::=  <num>
        ::=  <bool>
        ::=  <id>
        ::=  <prim> ( { <expr> }*(,) )
        ::=  proc (<id>*(,)) <expr>
        ::=  (<expr> <expr>*)
        ::=  if <expr> then <expr> else <expr>
<prim>  ::=  +  |  -  |  *  |  add1  |  sub1
<bool>  ::=  true  |  false
```

Quiz

- Question #3: Is the following an expression?

  add1(1, 7)

- Wrong answer: **No**

- Answer: **Yes** (according to our grammar)

Quiz

- Question #4: What is the value of the following expression?

  add1(1, 7)

- Answer: **2** (according to our interpreter)

- But no *real* language would accept **add1**(1, 7)

- Let's agree to call **add1**(1, 7) an ***ill-formed expression*** because **add1** should be used with only one argument

- Let's agree to never evaluate ill-formed expressions

Quiz

- Question #5: What is the value of the following expression?

  add1(1, 7)

- Answer: **None** - the expression is ill-formed

Quiz

- Question #6: Is the following a well-formed expression?

  +(**proc**(**x**)**x**, 5)

- Answer: **Yes**

Quiz

- Question #7: What is the value of the following expression?

  +(**proc**(**x**)**x**, 5)

- Answer: **None -** it produces an error:

  +: expects type <number> as 1st
  argument, given: (closure ((cbv-var x))
  (var-exp x) (empty-env-record)); other
  arguments were: 5

- Let's agree that a **proc** expression cannot be inside a + form

Quiz

- Question #8: Is the following a well-formed expression?

  +(**proc**(**x**)**x**, 5)

- Answer: **No**

Quiz

- Question #9: Is the following a well-formed expression?

  +((**proc**(**x**)**x** 7), 5)

- Answer: Depends on what we meant by *inside* in our most recent agreement
  - *Anywhere inside* - **No**
  - *Immediately inside* - **Yes**

- Since our intrepreter produces **12**, and since that result makes sense, let's agree on *immediately inside*

Quiz

- Question #10: Is the following a well-formed expression?

  +((**proc**(**x**)**x true**), 5)

- Answer: **Yes**, but we don't want it to be!

Quiz

- Question #11: Is it possible to define ***well-formed*** (as a decidable property) so that we reject all expressions that produce errors?

- Answer: **Yes**: reject *all* expressions!

Quiz

- Question #12: Is it possible to define ***well-formed*** (as a decidable property) so that we reject *only* expressions that produce errors?

- Answer: **No**

+(1, **if ... then** 1 **else proc**(**x**)**x**)

- If we always knew whether **...** produces true or false, we could solve the halting problem

Types

- Solution to our dilemma
  - In the process of rejecting expressions that are certainly bad, also reject some expressions that are good

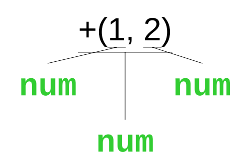
    +(1, **if** (**prime?** 131101) **then** 1 **else proc**(**x**)**x**)

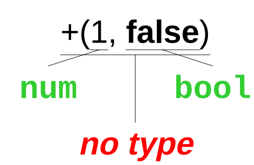
- Overall strategy:
  - Assign a ***type*** to each expression *without evaluating*
  - Compute the type of a complex expression based on the types of its subexpressions

Types

1 : `num`

true : `bool`





Type Rules

$\langle\text{num}\rangle : \texttt{num}$

$\langle\text{bool}\rangle : \texttt{bool}$

$$\frac{\langle\text{expr}\rangle_1 : \texttt{num} \qquad \langle\text{expr}\rangle_2 : \texttt{num}}{+(\langle\text{expr}\rangle_1, \langle\text{expr}\rangle_2) : \texttt{num}}$$

$1 : \texttt{num}$

true : $\texttt{bool}$

$$\frac{1 : \texttt{num} \qquad 2 : \texttt{num}}{+(1, 2) : \texttt{num}}$$

$$\frac{1 : \texttt{num} \qquad \textbf{false} : \texttt{bool}}{+(1, \textbf{false}) : \textit{no type}}$$

Type Rules

$\langle\text{num}\rangle : \texttt{num}$

$\langle\text{bool}\rangle : \texttt{bool}$

$$\frac{\langle\text{expr}\rangle_1 : \texttt{num} \qquad \langle\text{expr}\rangle_2 : \texttt{num}}{+(\langle\text{expr}\rangle_1, \langle\text{expr}\rangle_2) : \texttt{num}}$$

$$\frac{\dfrac{1 : \texttt{num} \qquad 2 : \texttt{num}}{+(1, 2) : \texttt{num}} \qquad 3 : \texttt{num}}{+(+(1, 2),3) : \texttt{num}}$$

Types: Conditionals

if true **then** 1 **else** 2

bool num num

num

if +(1,2) **then** 1 **else** 2

num num num

no type

if false **then** 2 **else** false

bool num bool

no type

Conditional Type Rules

$$\frac{\langle\text{expr}\rangle_1 : \texttt{bool} \qquad \langle\text{expr}\rangle_2 : \langle\text{type}\rangle_0 \qquad \langle\text{expr}\rangle_3 : \langle\text{type}\rangle_0}{\textbf{if } \langle\text{expr}\rangle_1 \textbf{ then } \langle\text{expr}\rangle_2 \textbf{ else } \langle\text{expr}\rangle_3 : \langle\text{type}\rangle_0}$$

$$\frac{\textbf{true} : \texttt{bool} \qquad 1 : \texttt{num} \qquad 2 : \texttt{num}}{\textbf{if true then } 1 \textbf{ else } 2 : \texttt{num}}$$

$$\frac{+(1,2) : \texttt{num} \qquad 1 : \texttt{num} \qquad 2 : \texttt{num}}{\textbf{if } +(1,2) \textbf{ then } 1 \textbf{ else } 2 : \textit{no type}}$$

$$\frac{\textbf{false} : \texttt{bool} \qquad 2 : \texttt{num} \qquad \textbf{false} : \texttt{bool}}{\textbf{if false then } 2 \textbf{ else false} : \textit{no type}}$$

Types: Variables and Functions

x : *no type*

proc(**bool x**)**x**

bool

(bool → bool)

proc(**bool x**)**if x then** 1 **else** 2

bool   num   num

num

(bool → num)

Variable and Function Type Rules

$$\{\ \ldots\ \langle id\rangle : T\ \ldots\ \} \vdash \langle id\rangle : T$$

$$\frac{\{\ \langle id\rangle : T_1\ \} + E \vdash e : T_2}{E \vdash \textbf{proc}(T_1\ \langle id\rangle)e : (T_1 \rightarrow T_2)}$$

Abbreviations: T = <type> e = <expr> E = <env>

Variable and Function Type Rules

$$\{\ \ldots\ \langle id\rangle : T\ \ldots\ \} \vdash \langle id\rangle : T$$

$$\frac{\{\ \langle id\rangle : T_1\ \} + E \vdash e : T_2}{E \vdash \textbf{proc}(T_1\ \langle id\rangle)e : (T_1 \rightarrow T_2)}$$

$\{\ \} \vdash$ **x** : *no type*

$$\frac{\{\textbf{x} : \texttt{bool}\} \vdash \textbf{x} : \texttt{bool}}{\{\ \} \vdash \textbf{proc}(\textbf{bool x})\textbf{x} : (\texttt{bool} \rightarrow \texttt{bool})}$$

$$\frac{\dfrac{\{\textbf{x} : \texttt{bool}\} \vdash \textbf{x} : \texttt{bool} \qquad \{\textbf{x} : \texttt{bool}\} \vdash 1 : \texttt{num} \qquad \{\textbf{x} : \texttt{bool}\} \vdash 2 : \texttt{num}}{\{\textbf{x} : \texttt{bool}\} \vdash \textbf{if x then } 1 \textbf{ else } 2 : \texttt{num}}}{\{\ \} \vdash \textbf{proc}(\textbf{bool x})\textbf{if x then } 1 \textbf{ else } 2 : (\texttt{bool} \rightarrow \texttt{num})}$$

Resvised Rules

$$E \vdash \langle num\rangle : \texttt{num}$$

$$E \vdash \langle bool\rangle : \texttt{bool}$$

$$\frac{E \vdash e_1 : \texttt{num} \qquad E \vdash e_1 : \texttt{num}}{E \vdash +(e_1, e_2) : \texttt{num}}$$

$$\frac{E \vdash e_1 : \texttt{bool} \qquad E \vdash e_2 : T_0 \qquad E \vdash e_3 : T_0}{E \vdash \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : T_0}$$

Types: Function Calls

(proc(**bool x**)**if x then** 1 **else** 2   **true**)

(`bool` → **num**)   `bool`

num

(proc(**bool x**)**if x then** 1 **else** 2   5)

(`bool` → **num**)   **num**

no type

(7 5)

num   **num**

no type

Function Call Type Rule

$$\frac{E \vdash e_1 : (T_2 \rightarrow T_3) \qquad E \vdash e_2 : T_2}{E \vdash (e_1\ e_2) : T_3}$$

$$\frac{\{\,\} \vdash \textbf{proc}(\textbf{bool x})\textbf{if x then } 1 \textbf{ else } 2 : (\texttt{bool} \rightarrow \texttt{num}) \qquad \{\,\} \vdash \textbf{true} : \texttt{bool}}{\{\,\} \vdash (\textbf{proc}(\textbf{bool x})\textbf{if x then } 1 \textbf{ else } 2 \quad \textbf{true}) : \texttt{num}}$$

$$\frac{\{\,\} \vdash \textbf{proc}(\textbf{bool x})\textbf{if x then } 1 \textbf{ else } 2 : (\texttt{bool} \rightarrow \texttt{num}) \qquad \{\,\} \vdash 5 : \texttt{num}}{\{\,\} \vdash (\textbf{proc}(\textbf{bool x})\textbf{if x then } 1 \textbf{ else } 2 \quad 5) : \textit{no type}}$$

$$\frac{\{\,\} \vdash 7 : \texttt{num} \qquad \{\,\} \vdash 5 : \texttt{num}}{\{\,\} \vdash (7\ \ 5) : \textit{no type}}$$

Types: Multiple Arguments

proc(**num x**, **num y**)+(**x**, **y**)

num   **num**

num

(**num** × **num** → **num**)

(**proc**(**num x**, **num y**)+(**x**, **y**)   5 6)

(**num** × **num** → **num**)   **num**   **num**

num

(**proc**(**num x**, **num y**)+(**x**, **y**)   5)

(**num** × **num** → **num**)   **num**

no type

Revised Function and Call Rules

$$\frac{\{\ \texttt{<id>}_1 : T_1, \ldots\ \texttt{<id>}_n : T_n\ \} + E \vdash e : T_0}{E \vdash \textbf{proc}(T_1\ \texttt{<id>}_1, \ldots\ T_n\ \texttt{<id>}_n)e : (T_1 \times \ldots\ T_n \rightarrow T_0)}$$

$$\frac{E \vdash e_0 : (T_1 \times \ldots\ T_n \rightarrow T_0) \qquad E \vdash e_1 : T_1 \qquad \ldots \qquad E \vdash e_n : T_n}{E \vdash (e_0\ e_1 \ldots\ e_n) : T_0}$$

New Interpreter and Checker

- Change our interpreter:
  - Add types for arguments and letrec results to the grammar
- Implement a type-checker:
  - Produces the same type that the rules procedure
  - Calls itself recusrively to get types for sub-expressions
  - Treat primitives as built-in functions

$$+ : (\texttt{num} \times \texttt{num} \to \texttt{num})$$