# The Mechanical Evaluation Of Expressions

Aditya Siram (@deech)

January 30, 2017

# Outline

# Motivation

- Arithmetic mechanized by a calculator
  - only 3 years before!
- The computational equivalent?
- What's a code "calculator" look like?

# VM/Abstract Machine

- Transform lambda calculus into machine with 4 registers
- The 1st virtual machine!
- The 1st bytecode!

# Where

- One of the first uses of 'where'

$$(u - 1)(u + 2) \quad \{\lambda u.(u - 1)(u + 2)\}[7 - 3]$$
$$\textbf{where } u = 7 - 3.$$

- Significant indentation

$$u/(u + 5) \quad \{\lambda u.\, u/(u + 5)\}$$
$$\textbf{where } u = a(a + 1) \quad [\{\lambda a.\, a(a + 1)\}[7 - 3]].$$
$$\textbf{where } a = 7 - 3$$

# Syntactic Sugar

- "Syntactic sugar" invented here!

  details unsettled, but should be enough to make the use of **where** in what follows (*a*) comprehensible and (*b*) plausibly a mere "syntactic sugaring" of AEs. Further

# Predicates/Selectors/Constructors

- Given:

  ```
  {\x. x + 1}
  ```

- Constructor:

  
  

  ```
  constructlambdaexp(x, [x + 1])
  ```

- Selector:

  ```
  bv(constructlambdaexp(x, [x + 1])) = x
  body(constructlambdaexp(x, [x + 1])) = [x + 1]
  ```

- Predicate:

  ```
  lambdaexp(constructlambdaexp(x, [x + 1])) = true
  ```

# ADTs!

- Almost ADT's:

```
data Exp =
     ... |
     Lambda [String] Exp

f (Lambda args body) = ...
```

# Closures

- "Closures" invented here!
  - lambda expression + environment = closure.

> a *closure* has
>
> an *environment part* which is a list whose two items are:
>
> (1) an environment
> (2) an identifier or list of identifiers,
>
> and a *control part* which consists of a list whose sole item is an AE.

- Not much different from:

```
data Exp = ...
    | Closure [(String, Exp)] [String] Exp
```

# Closures

- Constructor, just like for lambda expressions:

$$\mathit{constructclosure}((E, \mathit{bvX}), \mathit{unitlist}(\mathit{bodyX})).$$

# SECD

- SECD - "the virtual machine", 4 linked list registers:
  - Stack : the current stack frame
  - Enviroment: lookup list of bound identifiers
  - Control: the next thing to evaluate
  - Dump: a swap space of SECD's, temporarily holds old states

# Transform

- The transformation function:

$$
\begin{array}{l}
\mathit{Transform}(S,E,C,D) = \\
\quad \mathit{nullC} \rightarrow [hS{:}S',E',C',D'] \\
\qquad\qquad \textbf{where } S',E',C',D' = D \\
\quad \textbf{else} \rightarrow \\
\qquad \mathit{identifier}X \rightarrow [\mathit{locationEXE}{:}S,\ E,\ tC,\ D] \\
\qquad \lambda \mathit{exp}X \rightarrow \\
\qquad\quad [\mathit{constructclosure}((E,\ \mathit{bv}X),\mathit{unitlist}(\mathit{body}X)){:}S, \\
\qquad\quad E,\ tC,\ D] \\
\qquad X = \mathit{ap} \rightarrow \mathit{closure}(hS) \rightarrow \\
\qquad\qquad\qquad [(\,),\ \mathit{derive}(\mathit{assoc}(J,2\mathit{nd}S)E'), \\
\qquad\qquad\qquad\quad C, \\
\qquad\qquad\qquad\quad (t(tS),\ E,\ tC,\ D)] \\
\qquad\qquad\qquad \textbf{where } E',J = \mathit{environmentpart}(hS) \\
\qquad\qquad\qquad\quad \textbf{and } C' = \mathit{controlpart}(hS) \\
\qquad\qquad\quad \textbf{else} \rightarrow [(1\mathit{st}S)(2\mathit{nd}){:}t(tS),\ E,\ tC,\ D] \\
\qquad \textbf{else} \rightarrow [S,\ E,\ \mathit{rand}X{:}(\mathit{rator}X{:}(\mathit{ap}{:}tC)),\ D] \\
\qquad \textbf{where } X = hC
\end{array}
$$

- Notice destructuring!

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = []
E = []
C = [sq(2)]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifierX → [locationEXE:S, E, tC, D]
    λexpX →
      [constructclosure((E, bvX),unitlist(bodyX)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = []
E = []
C = [\n . n * n, 2]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = []
E = []
C = [2, \n . n * n, ap]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = [2]
E = []
C = [\n . n * n, ap]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = [closure [] x [x * x], 2]
E = []
C = [ap]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = []
E = [[(x 2)]]
C = [x * x]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = []
E = [[(x 2)]]
C = [x x * ap]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifierX → [locationEXE:S, E, tC, D]
    λexpX →
      [constructclosure((E, bvX),unitlist(bodyX)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = [[x x]]
E = [[(x 2)]]
C = [* ap]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = [[2 2]]
E = [[(x 2)]]
C = [* ap]
D = []
```

# SECD - Square Root

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifierX → [locationEXE:S, E, tC, D]
    λexpX →
      [constructclosure((E, bvX),unitlist(bodyX)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
sq = \n . n * n
S = [4]
E = [[(x 2)]]
C = []
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = []
E = []
C = [inc(double(2))]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = []
E = []
C = [double(2) inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = []
E = []
C = [2 double ap inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [2]
E = []
C = [double ap inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [2]
E = []
C = [(\x. x * 2) ap inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [[closure [] x [x * 2] 2]
E = []
C = [ap inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(rator X:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = []
E = [[x 2]]
C = [[x * 2] inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = []
E = [[x 2]]
C = [x 2 * ap inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [2]
E = [[x 2]]
C = [2 * ap inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(rator X:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [4]
E = [[x 2]]
C = [inc ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [4]
E = [[x 2]]
C = [(\x. x + 1) ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [[closure [x 2] x [x + 1]] 4
E = [[x 2]]
C = [ap]
D = []
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                 [( ), derive(assoc(J,2ndS)E'),
                      C,
                      (t(tS), E, tC, D)]
                 where E',J = environmentpart(hS)
                   and C' = controlpart(hS)
               else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = []
E = [[x 4] [x 2]]
C = [x + 1]
D = [[] [x 2] [] []]
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = []
E = [[x 4] [x 2]]
C = [1 x + ap]
D = [[] [x 2] [] []]
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(ratorX:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [+ 4 1]
E = [[x 4] [x 2]]
C = [ap]
D = [[] [x 2] [] []]
```

# SECD - Compose

```
Transform(S,E,C,D) =
  nullC → [hS:S',E',C',D']
          where S',E',C',D' = D
  else →
    identifier X → [locationEXE:S, E, tC, D]
    λexp X →
      [constructclosure((E, bvX),unitlist(body X)):S,
      E, tC, D]
    X = ap → closure(hS) →
                  [( ), derive(assoc(J,2ndS)E'),
                       C,
                       (t(tS), E, tC, D)]
                  where E',J = environmentpart(hS)
                    and C' = controlpart(hS)
                else → [(1stS)(2nd):t(tS), E, tC, D]
    else → [S, E, randX:(rator X:(ap:tC)), D]
    where X = hC
```

```
inc = \x. x + 1
double = \x. x * 2
S = [5]
E = [[x 4] [x 2]]
C = []
D = [[] [x 2] [] []]
```

# Offhand mentions

- Just some casual suggestions
  - Decades of PhDs!

# Offhand mentions

- Non-strict evaluation!

ducing the value, as specified above. For instance, it is not essential that the operand of a combination be evaluated before its operator. The operand might be

# Offhand mentions

- Partial evaluation

  evaluated after the operator; it might even be evaluated piecemeal when and if it is required during the application

# Offhand mentions

- Inlining

tion of its body. The AE might be subjected to pre-processing of various kinds, e.g. to disentangle combinations once for all or to remove its dependence on an arbitrary choice of identifiers occurring bound in it. The pre-processing might be more elaborate and

# Offhand mentions

- DSLs

Another separation achieved above is that between considerations special to a particular subject-matter, and considerations relevant to every subject-matter (or "universe of discourse," or "field of application," or "problem orientation"). The subject-matter is deter-

# Thanks!

- "The Mechanical Evaluation Of Expressions" `https://www.cs.cmu.edu/afs/cs/user/crary/www/819-f09/Landin64.pdf`