Structure and Interpretation of Computer Programs

COMP200
Today
Interpretation

- Parts of an interpreter
- Arithmetic calculator
- Names
- Conditionals and if
- Store procedures in the environment
- Environment as explicit parameter
- Defining new procedures
5. ENVIRONMENT AS EXPLICIT PARAMETER

Why?
• change from

\[
\text{(eval ' (plus* 6 4))}
\]

to

\[
\text{(eval ' (plus* 6 4) environment)}
\]
5. ENVIRONMENT AS EXPLICIT PARAMETER

Trivial Changes

• change from

  (eval ' (plus* 6 4))

to

  (eval ' (plus* 6 4) environment)

• All procedures that call eval have extra argument

• lookup and define use environment from argument
5. ENVIRONMENT AS EXPLICIT PARAMETER

Code
5. ENVIRONMENT AS EXPLICIT PARAMETER

Non-trivial Change

• change from

\[
\text{eval } '(\text{plus}* 6 4))
\]

to

\[
\text{eval } '(\text{plus}* 6 4) \text{ environment})
\]

• All procedures that call \texttt{eval} have extra argument

• \texttt{lookup} and \texttt{define} use \texttt{environment} from argument

• No other change from evaluator 4

• Only non-trivial code: case for \texttt{application?} in \texttt{eval}
6. DEFINING NEW PROCEDURES
A scheme* procedure:

```
(define* twice* (lambda* (x*) (plus* x* x*))
(twice* 4)
```
6. DEFINING NEW PROCEDURES

Strategy

• Add a case for \texttt{lambda*} to \texttt{eval}
  • the value of \texttt{lambda*} is a \texttt{compound procedure}
6. DEFINING NEW PROCEDURES

Strategy

• Add a case for \texttt{lambda*} to \texttt{eval}
  • the value of \texttt{lambda*} is a compound procedure

• Extend \texttt{apply} to handle compound procedures
6. DEFINING NEW PROCEDURES

Strategy

• Add a case for \texttt{lambda*} to \texttt{eval}
  • the value of \texttt{lambda*} is a compound procedure
• Extend \texttt{apply} to handle compound procedures
• Implement environment model
6. DEFINING NEW PROCEDURES

Code
IMPLEMENTATION OF $\lambda^*$
IMPLEMENTATION OF lambda*

(eval '(%lambda* (x*) (plus* x* x*)) GE)
IMPLEMENTATION OF lambda*

(eval '(lambda* (x*) (plus* x* x*)) GE)
(eval-lambda '(lambda* (x*) (plus* x* x*)) GE)
IMPLEMENTATION OF lambda*

(eval '(lambda* (x*) (plus* x* x*)) GE)
(eval-lambda '(lambda* (x*) (plus* x* x*)) GE)
(make-compound '(x*) '(plus* x* x*) GE)
IMPLEMENTATION OF \texttt{lambda*}

\texttt{(eval '(lambda* (x*) (plus* x* x*)) GE)}
\texttt{(eval-lambda '(lambda* (x*) (plus* x* x*)) GE)}
\texttt{(make-compound '(x*) '(plus* x* x*) GE)}
\texttt{(list 'compound '(x*) '(plus* x* x*) GE)}
IMPLEMENTATION OF \texttt{lambda*}

\begin{verbatim}
(eval '(lambda* (x*) (plus* x* x*)) GE)
(eval-lambda '(lambda* (x*) (plus* x* x*)) GE
(make-compound '(x*) '(plus* x* x*) GE)
(list 'compound '(x*) '(plus* x* x*) GE)
\end{verbatim}
IMPLEMENTATION OF lambda*

(eval '(%lambda* (x*) (plus* x* x*)) GE)
(eval-lambda '(%lambda* (x*) (plus* x* x*)) GE
(make-compound '(%x*) '(%plus* x* x*) GE)
(list 'compound '(%x*) '(%plus* x* x*) GE)
DEFINING A NAMED PROCEDURE

(eval '(define* twice*
   (lambda* (x*) (plus* x* x*))))) GE)
DEFINING A NAMED PROCEDURE

(eval '(define* twice*
       (lambda* (x*) (plus* x* x*)) GE)

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z*</td>
<td>9</td>
</tr>
<tr>
<td>true*</td>
<td>#t</td>
</tr>
</tbody>
</table>
DEFINING A NAMED PROCEDURE

(eval '(define* twice*
  (lambda* (x*) (plus* x* x*))))) GE

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z*</td>
<td>9</td>
</tr>
<tr>
<td>true*</td>
<td>#t</td>
</tr>
<tr>
<td>plus*</td>
<td></td>
</tr>
</tbody>
</table>
DEFINING A NAMED PROCEDURE

(eval '(define* twice* (lambda* (x*) (plus* x* x*))) GE)

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z*</td>
<td>9</td>
</tr>
<tr>
<td>true*</td>
<td>#t</td>
</tr>
<tr>
<td>plus*</td>
<td></td>
</tr>
</tbody>
</table>
DEFINING A NAMED PROCEDURE

(eval ' (define* twice* (lambda* (x*) (plus* x* x*)) ) ) GE

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z*</td>
<td>9</td>
</tr>
<tr>
<td>true*</td>
<td>#t</td>
</tr>
<tr>
<td>plus*</td>
<td></td>
</tr>
</tbody>
</table>
DEFINING A NAMED PROCEDURE

```
(eval '(define* twice*
  (lambda* (x*) (plus* x* x*))))) GE)
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z*</td>
<td>9</td>
</tr>
<tr>
<td>true*</td>
<td>#t</td>
</tr>
<tr>
<td>plus*</td>
<td></td>
</tr>
</tbody>
</table>
DEFINING A NAMED PROCEDURE

(eval '[(define* twice* (lambda* (x*) (plus* x* x*))]) GE)

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z*</td>
<td>9</td>
</tr>
<tr>
<td>true*</td>
<td>#t</td>
</tr>
<tr>
<td>plus*</td>
<td></td>
</tr>
<tr>
<td>twice*</td>
<td></td>
</tr>
</tbody>
</table>
DEFINING A NAMED PROCEDURE

(eval '(define* twice* (lambda* (x*) (plus* x* x*)))) GE

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z*</td>
<td>9</td>
</tr>
<tr>
<td>true*</td>
<td>#t</td>
</tr>
<tr>
<td>plus*</td>
<td></td>
</tr>
<tr>
<td>twice*</td>
<td></td>
</tr>
</tbody>
</table>
IMPLEMENTATION OF \texttt{apply}(1)

\begin{verbatim}
(eval '(twice* 4) GE)
(apply (eval 'twice* GE)
(map (lambda (e) (eval e GE)) '(4)))
(apply (list 'compound '(x*) '(plus* x* x*) GE) '(4))
(eval '(plus* x* x*)
(extend-env-with-new-frame '(x*) '(4) GE))
(eval '(plus* x* x*) E1)
\end{verbatim}
IMPLEMENTATION OF apply (1)

(eval '(twice* 4) GE)
(apply (eval 'twice* GE)
  (map (lambda (e) (eval e GE)) '(4))))
IMPLEMENTATION OF apply (1)

(eval '(twice* 4) GE)

(apply (eval 'twice* GE)
       (map (lambda (e) (eval e GE)) '(4))

(apply (list 'compound '(x*) '(plus* x* x*) GE)
       '(4))
IMPLEMENTATION OF apply (1)

(eval '(twice* 4) GE)

(apply (eval 'twice* GE)
         (map (lambda (e) (eval e GE)) '(4)))

(apply (list 'compound '(x*) '(plus* x* x*) GE)
        '(4))

(eval '(plus* x* x*)
      (extend-env-with-new-frame '(x*) '(4) GE))
IMPLEMENTATION OF apply (1)

```
(eval '(twice* 4) GE)
(apply (eval 'twice* GE)
  (map (lambda (e) (eval e GE)) '(4)))
(apply (list 'compound '(x*) '(plus* x* x*) GE) '(4))
(eval '(plus* x* x*)
  (extend-env-with-new-frame '(x*) '(4) GE))
(eval '(plus* x* x*) E1)
```
IMPLEMENTATION OF apply(1)

(eval '(twice* 4) GE)
(apply (eval 'twice* GE)
  (map (lambda (e) (eval e GE)) '(4)))
(apply (list 'compound '(x*) '(plus* x* x*) GE)
  '(4))
(eval '(plus* x* x*)
  (extend-env-with-new-frame '(x*) '(4) GE))
(eval '(plus* x* x*) E1)

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x*</td>
<td>9</td>
</tr>
</tbody>
</table>
IMPLEMENTATION OF apply (1)

(eval '(twice* 4) GE) some-other-environment

(apply (eval 'twice* GE)
   (map (lambda (e) (eval e GE)) '(4)))

(apply (list 'compound '(x*) '(plus* x* x*) GE) '(4))

(eval '(plus* x* x*)
   (extend-env-with-new-frame '(x*) '(4) GE))

(eval '(plus* x* x*) E1)
IMPLEMENTATION OF \texttt{apply} (1)

\begin{verbatim}
(eval '(twice* 4) \text{GE}) \text{some-other-environment}

(apply (eval 'twice* \text{GE})
   (map (lambda (e) (eval e \text{GE})) '(4)))

(apply (list 'compound '(x*) '(plus* x* x*) \text{GE})
       '(4))

(eval '(plus* x* x*)
   (extend-env-with-new-frame '(x*) '(4) \text{GE}))

(eval '(plus* x* x*) E1)
\end{verbatim}

\begin{tabular}{|c|c|}
\hline
\textbf{name} & \textbf{value} \\
\hline
x* & 9 \\
\hline
\end{tabular}
IMPLEMENTATION OF `apply (1)`

```
(eval '(twice* 4) GE)
(apply (eval 'twice* GE)
   (map (lambda (e) (eval e GE)) '(4)))
(apply (list 'compound '(x*) '(plus* x* x*) GE) '(4))
(eval '(plus* x* x*)
   (extend-env-with-new-frame '(x*) '(4) GE))
(eval '(plus* x* x*) E1)
```
IMPLEMENTATION OF apply (2)
IMPLEMENTATION OF apply (2)

(eval '(plus* x* x*) E1)
(apply (eval 'plus* E1)
  (map (lambda (e) (eval e E1)) '(x* x*))
)
(apply '(primitive #\[add]) (list (eval 'x* E1)
  (eval 'x* E1)))

(apply '(primitive #\[add]) '(4 4))
(scheme-apply #\[add] '(4 4))

8
IMPLEMENTATION OF ENVIRONMENT MODEL
IMPLEMENTATION OF ENVIRONMENT MODEL

• Environment = list<table>
IMPLEMENTATION OF ENVIRONMENT MODEL

- Environment = list<table>

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^*$</td>
<td>9</td>
</tr>
</tbody>
</table>
**IMPLEMENTATION OF ENVIRONMENT MODEL**

- Environment = list<table>

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x*</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>plus*</td>
<td>(primitive #[add])</td>
</tr>
<tr>
<td>greater*</td>
<td>(primitive #[grt])</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of environment model with tables and primitives]
IMPLEMENTATION OF ENVIRONMENT MODEL

Code
SUMMARY

• Cycle between eval and apply is the core of the evaluator
  • eval calls apply with operator and argument values
  • apply calls eval with expression and environment
  • no pending operations on either call
  an iterative algorithm if the expression is iterative
SUMMARY

- Cycle between eval and apply is the core of the evaluator
  - eval calls apply with operator and argument values
  - apply calls eval with expression and environment
  - no pending operations on either call
    - an iterative algorithm if the expression is iterative

- What is still missing from scheme*?
  - ability to evaluate a sequence of expressions
  - data types other than numbers and booleans
SUMMARY

• Cycle between eval and apply is the core of the evaluator
  • eval calls apply with operator and argument values
  • apply calls eval with expression and environment
  • no pending operations on either call
    an iterative algorithm if the expression is iterative

• What is still missing from scheme* ?
  • ability to evaluate a sequence of expressions
  • data types other than numbers and booleans

Everything in these lectures would still work if you deleted the stars from the names!