If it takes 60 seconds to compute a photomosaic for Problem Set 1 today on a typical PC, estimate how long it will take cs1120 students in 2012 to compute the same photomosaic? How long will it take in 2015?

\[
\frac{60 \times (2012 - 2009) \times 12}{18} = 15\text{ seconds in 2012, 4 doublings by 2015, 3.75 seconds in 2015.}
\]

Difference in years \times 12 = number of months
Number of months \div 18 = number of doublings according to Moore's Law

Reality check: Moore's "law" is just an "observation".

Are there any non-recursive natural languages? What would happen to a society that spoke one?

Not for humans at least. They would run out of original things to say.

Chimps and Dolphins are able to learn non-recursive "languages", but only humans have learned recursive languages.

Running out of Ideas

"It's all been said before."

Eventually true for a non-recursive language.

Never true for a recursive language.

There is always something original left to say!
### Pages in Revised Report on the Algorithmic Language Scheme

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitives</td>
<td>Standard Procedures 18</td>
<td>Standard Procedures 18</td>
</tr>
<tr>
<td></td>
<td>Primitive expressions 2</td>
<td>Primitive expressions 2</td>
</tr>
<tr>
<td></td>
<td>Identifiers, numerals 1</td>
<td>Identifiers, numerals 1</td>
</tr>
<tr>
<td>Means of Combination</td>
<td>Expressions 2</td>
<td>Expressions 2</td>
</tr>
<tr>
<td></td>
<td>Program structure 2</td>
<td>Program structure 2</td>
</tr>
<tr>
<td>Means of Abstraction</td>
<td>Definitions ½</td>
<td>Definitions ½</td>
</tr>
<tr>
<td></td>
<td>48 pages total (includes formal specification and examples)</td>
<td>48 pages total (includes formal specification and examples)</td>
</tr>
</tbody>
</table>

### English

<table>
<thead>
<tr>
<th>Morphemes</th>
<th>English Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>500,000</td>
<td>500,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grammar Rules</th>
<th>English Grammar for Dummies Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>100s (7)</td>
<td>384 pages</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pronouns</th>
<th>Rules of Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>~20</td>
<td></td>
</tr>
</tbody>
</table>

C++ Core language issues list has 948 items!
Assigning Meanings

This grammar generates (nearly) all surface forms in the Scheme language.

What do we need to do to know the meaning of every Scheme program?

Expressions and Values

• (Almost) every expression has a value
  – Have you seen any expressions that don’t have values?
• When an expression with a value is evaluated, its value is produced

Our goal is to define a meaning function, Eval, that defines the value of every Scheme expression:

\[ \text{Eval}(\text{Expression}) \Rightarrow \text{Value} \]

Today we do this informally with rules in English.

Evaluation Rule 1: Primitives

If the expression is a primitive, it evaluates to its pre-defined value.

\[
\begin{align*}
> 2 \\
2 \\
> \text{true} \\
\#t \\
> + \\
\text{#(primitive:+)}
\end{align*}
\]

Primitives are the smallest units of meaning: they can’t be broken down further, you need to know what they mean.
Name Expressions

Expression ::= NameExpression
NameExpression ::= Name

Evaluation Rule 2: Names

A name evaluates to the value associated with that name.

> (define two 2)
> two
2

Caveat: this simple rule only works if the value associated with a name never changes (until PSS).

Application Expressions

Expression ::= Application Expression
ApplicationExpression ::= (Expression MoreExpressions)
MoreExpressions ::= ε
MoreExpressions ::= Expression MoreExpressions

Evaluation Rule 3: Application

3. To evaluation an application expression:
   a) Evaluate all the subexpressions (in any order)
   b) Apply the value of the first subexpression to the values of all the other subexpressions.

(Application₀ Expression₁ Expression₂ ... )

Rules for Application

1. Primitives. If the procedure to apply is a primitive procedure, just do it.

2. Constructed Procedures. If the procedure is a constructed procedure, evaluate the body of the procedure with each parameter name bound to the corresponding input expression value.

Eval and Apply are defined in terms of each other.

Without Eval, there would be no Apply, without Apply there would be no Eval!
Survey Responses: Majors

- Cognitive Science: 28
- Computer Science: 20
- Psychology: 7
- Math: 4
- Economics: 3
- Commerce/Pre-Commerce: 3
- Undeclared: 2
- Physics: 2
- Environmental Sciences, English, Music: 1

Survey Responses: PS Partners

- Yes: 29
- No: 30
- Missing: 4

For PS2 everyone will be assigned a partner. For other problem sets, you’ll have different options.

Survey Responses: Office Hours

- Scheduling office hours: **Set Cover Problem**
  - Input: a set of sets of available times
  - Output: the minimum size set that includes at least one element from each of the input sets
  
  Later in the course, we’ll see that this problem is equivalent to the problem of computing an optimal photomosaic!

- My office hours will be:
  - Mondays, 1:30-2:30pm [Olsson 236A]
  - Tuesdays, 10:30-11:30am [Wilsdorf Cafe]

  Not a set cover: Everyone who selected at least three possible times can make at least one of these. If you can’t make office hours, email to arrange an appointment.

Honor Pledge

Finishing Scheme Meanings

\[
\text{Program} ::= \epsilon | \text{ProgramElement Program} \\
\text{ProgramElement} ::= \text{Expression} | \text{Definition} \\
\text{Definition} ::= (\text{define Name Expression}) \\
\text{Expression} ::= \text{PrimitiveExpression} | \text{NameExpression} | \text{ApplicationExpression} | \text{ProcedureExpression} | \text{IfExpression} \\
\text{PrimitiveExpression} ::= \text{Number} | \text{true} | \text{false} | \text{PrimitiveProcedure} \\
\text{NameExpression} ::= \text{Name} \\
\text{ApplicationExpression} ::= \text{Expression MoreExpressions} \\
\text{MoreExpressions} ::= \epsilon | \text{Expression MoreExpressions} \\
\text{ProcedureExpression} ::= (\lambda (\text{Parameters}) \text{Expression}) \\
\text{Parameters} ::= \epsilon | \text{Name Parameters} \\
\text{IfExpression} ::= \text{if Expression Pred Expression Consequent Alt} \\
\text{Expression} ::= \text{ProcedureExpression} \\
\text{Parameters} ::= \epsilon | \text{Name Parameters}
\]

Making Procedures

- \text{lambda} means “make a procedure”

- \text{ProcedureExpression ::= (lambda (Parameters) Expression)}

- \text{Parameters ::= \epsilon}

- \text{Parameters ::= Name Parameters}
Evaluation Rule 4: Lambda

A lambda expression evaluates to a procedure that takes the given parameters and has the expression as its body.

\[
\text{ProcedureExpression} ::= \text{lambda} \text{(Parameters) Expression}
\]

Parameters ::= ε | Name Parameters

Lambda Example: Tautology Function

\[
\text{(lambda () } \text{true)} \text{ with no parameters with body true}
\]

> (\(\text{(lambda () } \text{true)}\))

#<procedure>: expects no arguments, given 1: 1120

> (\(\text{(lambda () } \text{true)}\))

#t

> (\(\text{(lambda (x) x)}\))

1120

Next class we’ll follow the evaluation rules through more interesting examples.

Evaluation Rule 5: If

\[
\text{IfExpression} ::= \text{if} \text{Expression Predicate Expression Consequent Alternate}
\]

To evaluate an if expression:
(a) Evaluate Expression\text{Predicate}.
(b) If it evaluates to a false value, the value of the if expression is the value of Expression\text{Alternate}; otherwise, the value of the if expression is the value of Expression\text{Consequent}.

Completeness of Evaluation Rules

Program ::= ε | ProgramElement Program
ProgramElement ::= Expression | Definition
Definition ::= (define Name Expression)
Expression ::= PrimitiveExpression | NameExpression
 | ApplicationExpression | ProcedureExpression | IfExpression
PrimitiveExpression ::= Number | true | false | PrimitiveProcedure
NameExpression ::= Name
ApplicationExpression ::= (Expression MoreExpressions)
MoreExpressions ::= ε | Expression MoreExpressions
ProcedureExpression ::= (lambda (Parameters) Expression)
Parameters ::= ε | Name Parameters
IfExpression ::= (if Expression\text{Predicate} Expression\text{Consequent} Expression\text{Alternate})

Since we have an evaluation rule for each grammar rule, we can determine the meaning of any Scheme program!

Now You Can Write Any Program!

- You know enough now to define a procedure that performs every possible computation!
  - We’ll prove this later in the course
- We’ll learn some more useful Scheme forms:
  - There are a few more special forms (like if)
  - But, none of these are necessary...just helpful
- We have not defined the evaluation rules precisely enough to unambiguously understand all programs (e.g., what does “value associated with a name” mean?)

Charge

- PS1 Due at beginning of class Wednesday
- Read Chapter 4 by Friday
- Now you know enough to produce every computation, the rest is just gravy:
  - More efficient, elegant ways to express computations
  - Ways to analyze the computations