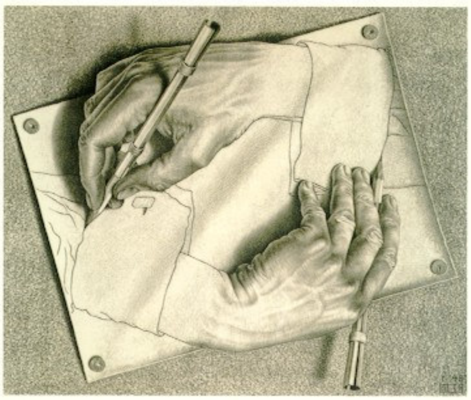


Class 3: Rules of Evaluation

David Evans
cs1120 Fall 2009

Menu

- Questions from Notes
  - Computing photomosaics, non-recursive languages, hardest language elements to learn
- Scheme's Rules of Evaluation
  - (break: Survey Responses)

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?

```
> (/ (* (- 2012 2009) 12) 18)
2
> (/ 60 (* 2 2))
15
> (/ (* (- 2015 2009) 12) 18)
4
> (/ 60 (* 2 2 2 2))
15/4
> (exact->inexact (/ 60 (* 2 2 2 2)))
3.75
```

Difference in years * 12 = number of months
Number of months / 18 = number of doublings according to Moore's Law

60 seconds today, 2 doublings by 2012
15 seconds in 2012

60 seconds today, 4 doublings by 2015
3.75 seconds in 2015

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

"Its 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!

Language Elements

When learning a foreign language, which elements are hardest to learn?

- Primitives: lots of them, and hard to learn real *meaning*
- Means of Combination
  - Complex, but, all natural languages have similar ones [Chomsky]

SOV (45% of all languages)	*Sentence ::= Subject Object Verb* (Korean)
SVO (42%)	*Sentence ::= Subject Verb Object* (English)
VSO (9%)	*Sentence ::= Verb Subject Object* (Welsh) "Lladdodd y ddraig y dyn." (Killed the dragon the man.)
OSV (<1%):	Tobati (New Guinea)
Scheme:	*Expression ::= (Verb Object)*

- Means of Abstraction: few of these, but tricky to learn differences across languages

  English: I, we

  Tok Pisin (Papua New Guinea): mi (I), mitupela (he/she and I), mitripela (both of them and I), mipela (all of them and I), yumitupela (you and I), yumitripela (both of you and I), yumipela (all of you and I)

  Scheme: define

	Pages in *Revised⁵ Report on the Algorithmic Language Scheme*
Primitives	
Means of Combination	
Means of Abstraction	
	48 pages total (includes formal specification and examples)

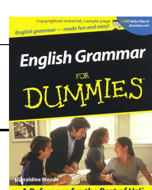
	Pages in *Revised⁵ Report on the Algorithmic Language Scheme*
Primitives	Standard Procedures 18 Primitive expressions 2 Identifiers, numerals 1
Means of Combination	Expressions 2 Program structure 2
Means of Abstraction	Definitions ½
	48 pages total (includes formal specification and examples)

	Pages in *Revised⁵ Report on the Algorithmic Language Scheme*	Pages in C++ Language Specification (1998)
Primitives	Standard Procedures 18 Primitive expressions 2 Identifiers, numerals 1	
Means of Combination	Expressions 2 Program structure 2	
Means of Abstraction	Definitions ½	
	48 pages total (includes formal specification and examples)	

	Pages in *Revised⁵ Report on the Algorithmic Language Scheme*	Pages in *C++ Language Specification* (1998)
Primitives	Standard Procedures 18 Primitive expressions 2 Identifiers, numerals 1	Standard Procedures 356 Primitive expressions 30 Identifiers, numerals 10
Means of Combination	Expressions 2 Program structure 2	Expressions, Statements 197 Program Structure 35
Means of Abstraction	Definitions ½	Declarations, Classes 173
	48 pages total (includes formal specification and examples)	**776** pages total (includes no formal specification or examples)

C++ Core language issues list has 948 items!

	Pages in *Revised⁵ Report on the Algorithmic Language Scheme*	English
Primitives	Standard Procedures 18 Primitive expressions 2 Identifiers, numerals 1	Morphemes ? Words in Oxford English Dictionary 500,000
Means of Combination	Expressions 2 Program structure 2	Grammar Rules 100s (?) *English Grammar for Dummies* Book 384 pages
Means of Abstraction	Definitions ½	Pronouns ~20
	48 pages total (includes formal specification and examples)	



Scheme Grammar

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** $Expression_{Pred}$ $Expression_{Consequent}$ $Expression_{Alt}$**)**

Assigning Meanings

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** $Expression_{Pred}$ $Expression_{Consequent}$ $Expression_{Alt}$**)**

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?

Definitions

Program ::= ε | *ProgramElement Program*
ProgramElement ::= *Expression* | *Definition*
Definition ::= **(define** ***Name*** *Expression***)**

A definition associates the value of its expression with the name.

(define two 2)

After this definition, the value associated with the name **two** is 2.

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:

Eval(*Expression*) ⇒ **Value**

Today we do this informally with rules in English.

Primitive Expressions

PrimitiveExpression ::= ***Number*** | **true** | **false** | ***PrimitiveProcedure***

Evaluation Rule 1: Primitives

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

```
> 2
2
> true
#t
> +
#<primitive:+>
```

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 PS5).

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.

($Expression_0$ $Expression_1$ $Expression_2$ …)

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.

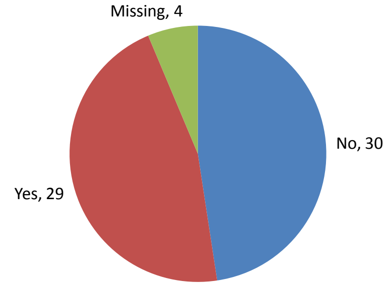
Eval and **Apply** are defined in terms of each other.

Without **Eval**, there would be no **Apply**, without **Apply** there would be no **Eval**!

Survey Responses: Majors

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

Survey Responses: PS Partners



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

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** $Expression_{Pred}$ $Expression_{Consequent}$ $Expression_{Alt}$**)**

Making Procedures

lambda means "make a procedure"

Expression ::= *ProcedureExpression*
ProcedureExpression ::=
(lambda (*Parameters***)** *Expression***)**
Parameters ::= ε
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.

ProcedureExpression ::= **(lambda (***Parameters***)** *Expression***)**
Parameters ::= ε | ***Name*** *Parameters*

Lambda Example: Tautology Function

```
(lambda      make a procedure
 ()          with no parameters
 true)       with body true
```

```
> ((lambda () true) 1120)
#<procedure>: expects no arguments, given 1: 1120
> ((lambda () true))
#t
> ((lambda (x) x) 1120)
1120
```

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

Evaluation Rule 5: If

IfExpression
::= **(if** $Expression_{Predicate}$
$Expression_{Consequent}$
$Expression_{Alternate}$**)**

To evaluate an if expression:

(a) Evaluate $Expression_{Predicate}$.

(b) If it evaluates to a false value, the value of the if expression is the value of $Expression_{Alternate}$; otherwise, the value of the if expression is the value of $Expression_{Consequent}$.

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** $Expression_{Pred}$ $Expression_{Consequent}$ $Expression_{Alt}$**)**

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