Lecture 3: Rules of Evaluation

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/cs1xx

Menu

- Language Elements
- Why don't we just program computers using English?
- Evaluation
- Procedures

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" (some linguists argue they are not really "languages"), but **only humans can learn 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*	
VSO (9%)	*Sentence* ::= *Verb Subject Object*	(Welsh)
	"Lladdodd y ddraig y dyn." (Killed the dragon the man.)	
OSV (<1%):	Tobati (New Guinea)	
Schemish:	*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)

	Pages in *Revised⁵ Report on the Algorithmic Language Scheme*		Pages in C++ Language Specification (1998)	
Primitives	Standard Procedures	18	Standard Procedures	356
	Primitive expressions	2	Primitive expressions	30
	Identifiers, numerals	1	Identifiers, numerals	10
Means of Combination	Expressions	2	Expressions, Statements	197
	Program structure	2	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 529 items!

	Pages in *Revised⁵ Report on the Algorithmic Language Scheme*		English	
Primitives	Standard Procedures	18	Morphemes	7
	Primitive expressions	2	Words in Oxford English Dictionary	500,000
	Identifiers, numerals	1		
Means of Combination	Expressions	2	Grammar Rules	100s (?)
	Program structure	2	*English Grammar for Dummies* Book	384 pages
Means of Abstraction	Definitions	½	Pronouns	~20
	48 pages total (includes formal specification and examples)			

Why don't we just program computers using English?

- Too hard and complex

  Non-native English speakers don't need convincing. The rest of you have spent your whole life learning English (and first 5 years of your life doing little else) and still don't know useful words like floccipaccinihilipilification! There are thoughts that even native speakers find it hard to express.

  By the end of today you will know enough Scheme (nearly the entire language) to express and understand *every* computation. By PS7, you will know enough to completely and precisely describe Scheme in terms of itself (try doing that in English!)

Why don't we just program computers using English?

- Not concise enough

  English:
  To find the maximum of two numbers, compare them. If the first number is greater than the second number, the maximum is the first number. Otherwise, the maximum is the second number.

  Scheme:
  (define (max a b) (if (> a b) a b))

Why don't we just program computers using English?

- Limited means of abstraction

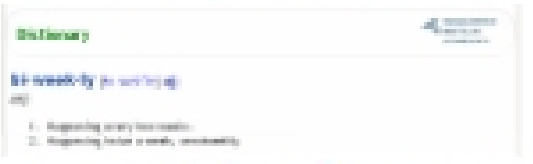
  There are only a few pronouns: he, she, it, they, these, ...
  (English doesn't even have a gender-neutral pronoun for a person!)
  Only Webster and Oxford can make up new ones.

  define allows any programmer to make up as many pronouns as she wants, and use them to represent anything.

Why don't we just program computers using English?

- Mapping between surface forms and meanings are ambiguous and imprecise

  Would you rather be paid biweekly or every week?



  The exact meaning(s) of every Scheme expression is determined by simple, unambiguous rules we will learn today (and refine later in the course).

Essential Scheme

Expression ::= (*Expression*$_1$ *Expression**)
Expression ::= (**if** *Expression*$_1$ *Expression*$_2$ *Expression*$_3$)
Expression ::= (**define** *name Expression*)
Expression ::= *Primitive*
Primitive ::= *number*
Primitive ::= + | - | * | ...
Primitive ::= ...

Grammar is clear, just follow the replacement rules. But what does it all mean?

Evaluation

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

Evaluation Rule 1: Primitives

If the expression is a *primitive*, it is self-evaluating.

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


Evaluation Rule 2: Names

If the expression is a *name*, it evaluates to the value associated with that name.

```
> (define two 2)
> two
2
```


Evaluation Rule 3: Application

3. If the expression is an application:
   a) **Evaluate** all the subexpressions of the combination (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. If the procedure to apply is a *primitive*, just do it.
2. If the procedure is a *compound procedure*, **evaluate** the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.