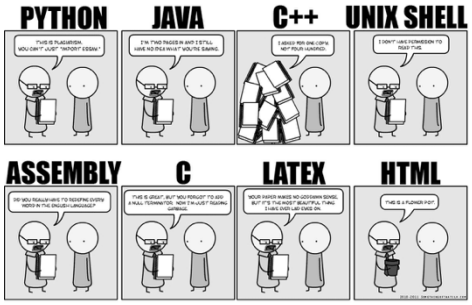
When You Write Your Essays in Programming Languages



http://www.somethingofthatilk.com/index.php?id=135

CS 252:
Advanced Programming Language Principles

Operational Semantics

Prof. Tom Austin
San José State University

Why do we need formal semantics?

What is the value of x?

```
if true then
  x = 1
else
  x = 0
```

Value of x is 1

What is the value of x?

```
if false then
  x = 1
else
  x = 0
```

Value of x is 0

What is the value of x?

```
if 0 then
  x = 1
else
  x = 0
```

Will x be set to 0, like in C/C++?

Will x be set to 1, like in Ruby?

Or will it be an error, like in Java?

What is the value of `x`?

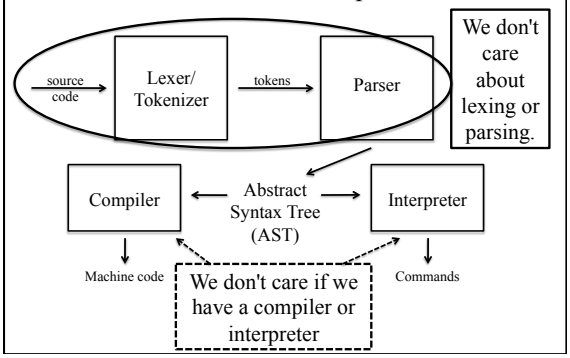
```
x = if true
      then 1
      else 0
```

Is assignment valid or an error?

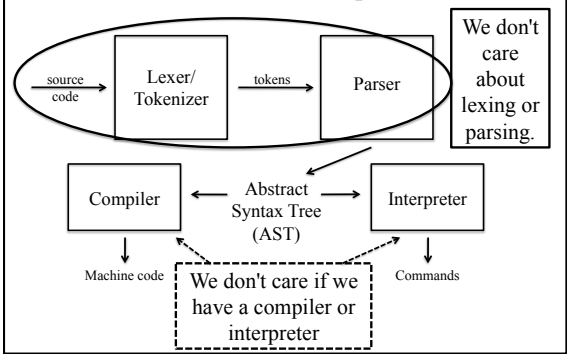
Formal semantics *crisply* define how our language features work.

To demonstrate, let's make a small language.

A Review of Compilers



A Review of Compilers



Abstract Syntax Tree (AST)

ASTs are the focus of PL research

Bool* Language

`e ::=`	*expressions:*
`true`	constant true
`| false`	constant false
`| if e then e else e`	conditional

Despite appearances, these are really ASTs

Values in Bool*

`v ::=`		*values:*
	`true`	constant true
`|`	`false`	constant false

Formal Semantic Styles

- Operational semantics
  - Big-step (or "natural")
  - Small-step (or "structural")
- Axiomatic semantics
- Denotational semantics

Operational semantics specify how expressions should be evaluated.

There are **two** different approaches.

B**ig-step** operational semantics evaluate every expression to a value.

The expression e …

e ⇓ v

… evaluates to …

… the value v.

Small-step operational semantics evaluate an expression until it is in *normal form*.

"normal form" – it cannot be evaluated further.

Small-Step Evaluation Relation

One step in the evaluation

e -> e' -> e'' -> v

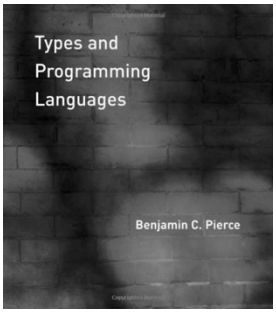
Many steps in the evaluation

e ->* v

TAPL

The top reference for more details on PL formalisms.

Available at library.



Small-step semantics for Bool*

(in-class)

Bool* Small-Step Semantics

E-IfTrue

```
if true then e2 else e3 -> e2
```

E-IfFalse

```
if false then e2 else e3 -> e3
```

E-If

```
          e1 -> e1'
    ------------------------------
    if e1 then e2 else e3
  -> if e1' then e2 else e3
```

Bool* small-step example

```
   if (if true then false
              else true)
      then true
      else false
-> if false then true else false
-> false
```

By the **E-IfTrue** rule

By the **E-IfFalse** rule

Bool* extension: numbers

- **0**
- **succ** creates a new number
  - **succ 0** represents 1
  - **succ succ 0** represents 2, etc.
- **pred** gets the predecessor of a number

Extended Bool* Language

```
e ::= true
    | false
    | if e then e else e
    | 0
    | succ e
    | pred e
```

Extended values and semantic rules

(in-class)

Literate Haskell

- Files use .lhs extension (rather than .hs)
- Code lines begin with >
- All other lines are comments

```
-- Regular .hs     Literate Haskell
-- source file     src file (.lhs)

foo x = 1          > foo x = 1
  + (foo (x - 1))  >   + (foo (x - 1))
```

Lab 2: Write a Bool* Interpreter

- Starter code is available at http://cs.sjsu.edu/~austin/cs252-spring16/labs/lab2/bool.lhs
- Part 1: Complete evaluate function
- Part 2: Extend Bool* with `0`, `succ`, and `pred`