Welcome to CS301

Programming Languages: the ultimate user interface

Why programming languages?

- Language influences thought
- P.L. features as tools for specifying computation
- Raise consciousness of language features
- Different programming styles: more powerful problem solving

Some language features

- Familiar:
  - Automatic storage management
  - Inheritance
- Strange:
  - Parametric polymorphism
  - First-class functions

Classifying languages

- Imperative, object-oriented, functional, logic programming and more
- Most are hybrids, e.g. Java is object-oriented and imperative
- Isolate features to understand what classifications mean

Formal semantics

- A taste of formal semantics will give you an idea of how we say precisely what a program will do

Impcore: an imperative core language

Impcore features

- Assignment: `(set x e)`
- Loop: `(while e1 e2)`
- Conditional: `(if e1 e2 e3)`
- Sequencing: `(begin e1 ... en)`
- Procedure: `(f e1 ... en)`

What is “imperative”?

- Computations work on a mutable store
- Order matters, e.g.

```
(begin (set x 1) (set x 2))
```

- is different from

```
(begin (set x 2) (set x 1))
```


An example program

```
(define gcd (m n)
  (begin
    (while (!= (set r (mod m n)) 0)
      (begin
        (set m n)
        (set n r)))
    n))
```


Abstract syntax

Sample languages

- Impcore:

```
(while e1 e2)
```

- uScheme:

```
(lambda (x) (+ x 1))
```

- uSmalltalk:

```
(spend:for: account 50 #plumber)
```

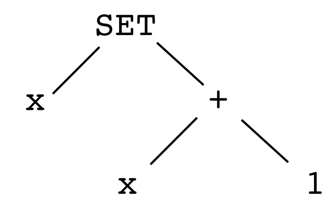
Ignoring concrete syntax

- All our languages look alike!
- ...on the surface, that is
- ...so we can concentrate on what's underneath:

Abstract syntax

Abstract syntax

- The **tree structure** of the language
- Data structure used by interpreters & compilers
- `(set x (+ x 1))`
  `x = x+1;`
  `x := x+1`



Specifying abstract syntax

- CFG notation
- Label nodes with all-caps *constructors*
- Child nodes in parentheses

```
Exp = LITERAL (Value)
    | VAR (Name)
    | SET (Name, Exp)
    | ...
```


Impcore's abstract syntax

```
Toplevel
 = EXP (Exp)
 | DEFINE (Name, Namelist, Exp)
 | VAL (Name, Exp)
 | USE (Name)
```

...continued

```
Exp = LITERAL (Value)
    | VAR (Name)
    | SET (Name, Exp)
    | IF (Exp, Exp, Exp)
    | WHILE (Exp, Exp)
    | BEGIN (Explist)
    | APPLY (Name, Explist)
```


Free variables

- A variable name is an expression

  `x`

- but it means nothing in isolation; it is a *free variable*
- To give meaning to free variables, we use

  Environments

Environments

- Environment: a mapping from names to meanings
- In Impcore meanings are *values*
- To bind a name to a value we write

  `(val x 2)`

- ...adding the mapping $x \mapsto 2$ to the current environment

On Notations:

Metalanguage

- Metalanguage is language about language
- Object language is the thing metalanguage is talking about
- Know which is which! Clues: fonts, Greek letters
- A metavariable: x; an object var: $\mathtt{x}$

Greek letters

- Learn to pronounce them! We will use a small number in a stereotyped way.
- $\rho, \xi, \phi, \mu, \tau$
- Spelled “rho, xi, phi, mu, tau”
- Pronounced “roe, ksigh, fie, myou, tau”

Assignments

- Read R&K chapter 2 through 2.4
- Problem set one is due Friday at 11:59 PM
- Come to lab this afternoon

Next time

- Introduction to operational semantics

$$\frac{x \notin \mathrm{dom}\ \rho \qquad x \in \mathrm{dom}\ \xi}{\langle \mathrm{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \xi(x), \xi, \phi, \rho \rangle}$$

Course mechanics

Useful information

- Course home page
- Syllabus
- My home page
- About lab assignments
- About grading, and doing your own work

CS301
Session 2

Overview

Operational semantics

- Concise, precise guide to what the language *means*
- Specification for interpreter or compiler
- Supports proofs of language and program properties

How it works

- A set of inference rules specifies the behavior of a hypothetical *abstract machine*
- Use the rules to see how a particular expression is evaluated in a given context
- Reason about the system of rules to prove general properties of the object language

Inference systems

- Remember your logic course:
  - formal logical system: axioms and inference rules
- Axiom says what is unconditionally true
- Inference rule says that the conclusion is true if the premises are

Top-level items

- The judgment is $\langle t, \xi, \phi \rangle \rightarrow \langle \xi', \phi' \rangle$
- In other words, we execute top-level items for their *effect*

The global environment

- Top-level expressions

$$\frac{\langle e, \xi, \phi, \{\} \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{EXP}(e), \xi, \phi \rangle \rightarrow \langle \xi', \phi \rangle}$$

- Variable declarations

$$\frac{\langle e, \xi, \phi, \{\} \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{VAL}(x, e), \xi, \phi \rangle \rightarrow \langle \xi'\{x \mapsto v\}, \phi \rangle}$$

The function environment

- We just bind the function name to a piece of abstract syntax

$$\frac{x_1, \ldots, x_n \text{ all distinct}}{\langle \text{DEFINE}(f, \langle x_1, \ldots, x_n \rangle, e), \xi, \phi \rangle \rightarrow \langle \xi, \phi\{f \mapsto \text{USER}(\langle x_1, \ldots, x_n \rangle, e)\} \rangle}$$

Expressions

- The judgment is $\langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle$
- In other words, we evaluate an expression to produce a value, and for its effects

Literal values

- Axiom: literal values

$$\frac{}{\langle \mathrm{LITERAL}(v), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi, \phi, \rho \rangle}$$

Using variables

- Variables are either parameters

$$\frac{x \in dom\ \rho}{\langle \mathrm{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \rho(x), \xi, \phi, \rho \rangle}$$

- ...or globals - note “shadowing”

$$\frac{x \notin dom\ \rho \qquad x \in dom\ \xi}{\langle \mathrm{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \xi(x), \xi, \phi, \rho \rangle}$$

Assignment

- Our first recursive rule: assignment updates the appropriate environment

$$\frac{x \in dom\ \rho \qquad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \mathrm{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho'\{x \mapsto v\} \rangle}$$

- How do we modify the rule for assignment to a global variable?

Conditional

- What can we conclude about an implementation? Should it evaluate all three subexpressions?

$$\frac{\langle e_1, \xi, \phi, \rho\rangle \Downarrow \langle v_1, \xi', \phi, \rho'\rangle \qquad v_1 \neq 0 \qquad \langle e_2, \xi', \phi, \rho'\rangle \Downarrow \langle v_2, \xi'', \phi, \rho''\rangle}{\langle \mathrm{IF}(e_1, e_2, e_3), \xi, \phi, \rho\rangle \Downarrow \langle v_2, \xi'', \phi, \rho''\rangle}$$

- What's the other rule?

Iteration

- Specify iteration in terms of recursion!

$$\frac{\langle e_1, \xi, \phi, \rho\rangle \Downarrow \langle v_1, \xi', \phi, \rho'\rangle \qquad v_1 \neq 0 \qquad \langle e_2, \xi', \phi, \rho'\rangle \Downarrow \langle v_2, \xi'', \phi, \rho''\rangle \qquad \langle \mathrm{WHILE}(e_1, e_2), \xi'', \phi, \rho''\rangle \Downarrow \langle v_3, \xi''', \phi, \rho'''\rangle}{\langle \mathrm{WHILE}(e_1, e_2), \xi, \phi, \rho\rangle \Downarrow \langle v_3, \xi''', \phi, \rho'''\rangle}$$

- Even a "null" loop can have an effect:

$$\frac{\langle e_1, \xi, \phi, \rho\rangle \Downarrow \langle v_1, \xi', \phi, \rho'\rangle \qquad v_1 = 0}{\langle \mathrm{WHILE}(e_1, e_2), \xi, \phi, \rho\rangle \Downarrow \langle v_1, \xi', \phi, \rho'\rangle}$$

Sequencing

- This rule has a variable number of premises

$$\frac{\langle e_1, \xi_0, \phi, \rho_0\rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1\rangle \qquad \langle e_2, \xi_1, \phi, \rho_1\rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2\rangle \qquad \vdots \qquad \langle e_n, \xi_{n-1}, \phi, \rho_{n-1}\rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n\rangle}{\langle \mathrm{BEGIN}(e_1, e_2, \ldots, e_n), \xi_0, \phi, \rho_0\rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n\rangle}$$

- There's also an axiom for empty BEGIN

Function application

- Here we create a parameter environment

$$\frac{\phi(f) = \mathrm{USER}(\langle x_1, \ldots, x_n\rangle, e) \qquad x_1, \ldots, x_n \text{ all distinct} \qquad \langle e_1, \xi_0, \phi, \rho_0\rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1\rangle \qquad \vdots \qquad \langle e_n, \xi_{n-1}, \phi, \rho_{n-1}\rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n\rangle \qquad \langle e, \xi_n, \phi, \{x_1 \mapsto v_1, \ldots x_n \mapsto v_n\}\rangle \Downarrow \langle v, \xi', \phi, \rho_n\rangle}{\langle \mathrm{APPLY}(f, e_1, e_2, \ldots, e_n), \xi_0, \phi, \rho_0\rangle \Downarrow \langle v, \xi', \phi, \rho_n\rangle}$$

ρ'

- We also have rules for all the primitive functions

The PRINT primitive

- If PRINT is a function, its application must return a value. We arbitrarily specify 0.

$$\frac{\phi(f) = \text{PRIMITIVE}(\text{print}) \quad \langle e, \xi, \phi, \rho\rangle \Downarrow \langle v, \xi', \phi, \rho'\rangle}{\langle \text{APPLY}(f, e), \xi, \phi, \rho\rangle \Downarrow \langle 0, \xi', \phi, \rho'\rangle}$$

- Why are the output environments different from the input environments?

An example deduction

- Let's construct a deduction showing how to evaluate

```
(while x (set x (- x 1)))
```

- in the global environment $\{x \mapsto 1\}$ and the empty parameter environment $\{\}$

$\langle \text{WHILE}(\text{VAR}(x), e), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle ?, ?, \phi, ?\rangle$

✓ $\langle \text{VAR}(x), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle ?, ?, \phi, ?\rangle$ $\langle 1, \{x \mapsto 1\}, \phi, \{\}\rangle$

✓ $x \notin \text{dom}(\{\})$ $\quad x \in \text{dom}(\{x \mapsto 1\})$ $\quad$ ✓ $1 \neq 0$

$\langle \text{SET}(x, \text{APPLY}(-, \text{VAR}(x), \text{LITERAL}(1))), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow$

✓ $x \notin \text{dom}(\{\})$ $\quad x \in \text{dom}(\{x \mapsto 1\})$ $\quad \langle ?, ?, \phi, ?\rangle$

$\langle \text{APPLY}(-, \text{VAR}(x), \text{LITERAL}(1))), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow$

$\langle ?, ?, \phi, ?\rangle$

$\langle \text{WHILE}(\text{VAR}(x), e), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle ?, ?, \phi, ?\rangle$

✓ $\langle \text{VAR}(x), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle 1, \{x \mapsto 1\}, \phi, \{\}\rangle$

✓ $\langle \text{SET}(x, \text{APPLY}(-, \text{VAR}(x), \text{LITERAL}(1))), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow$

✓ $x \notin \text{dom}(\{\})$ $\quad x \in \text{dom}(\{x \mapsto 1\})$ $\quad \langle ?, ?, \phi, ?\rangle$ $\langle 0, \{x \mapsto 1\}, \phi, \{\}\rangle$

✓ $\langle \text{APPLY}(-, \text{VAR}(x), \text{LITERAL}(1))), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow$

✓ $\phi(-) = \text{PRIMITIVE}(\text{minus})$ $\quad \langle ?, ?, \phi, ?\rangle$ $\langle 0, \{x \mapsto 1\}, \phi, \{\}\rangle$

✓ $\langle \text{VAR}(x), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle 1, \{x \mapsto 1\}, \phi, \{\}\rangle$

✓ $\langle \text{LITERAL}(1), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle 1, \{x \mapsto 1\}, \phi, \{\}\rangle$

$\langle \mathrm{WHILE}(\mathrm{VAR}(x), e), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle 0, \{x \mapsto 0\}, \phi, \{\}\rangle$

$\langle \mathrm{VAR}(x), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow \langle 1, \{x \mapsto 1\}, \phi, \{\}\rangle$

$\langle \mathrm{SET}(x, \mathrm{APPLY}(-, \mathrm{VAR}(x), \mathrm{LITERAL}(1))), \{x \mapsto 1\}, \phi, \{\}\rangle \Downarrow$

$\langle 0, \{x \mapsto 0\}, \phi, \{\}\rangle$

$\langle \mathrm{WHILE}(\mathrm{VAR}(x), e), \{x \mapsto 0\}, \phi, \{\}\rangle \Downarrow \langle 0, \{x \mapsto 0\}, \phi, \{\}\rangle$

$\langle \mathrm{VAR}(x), \{x \mapsto 0\}, \phi, \{\}\rangle \Downarrow \langle 0, \{x \mapsto 0\}, \phi, \{\}\rangle$

$x \notin \mathrm{dom}(\{\}) \quad x \in \mathrm{dom}(\{x \mapsto 0\})$

$0 = 0$

Properties of the semantics

- We can prove by inspecting the rules that - for this system - evaluation is deterministic
- ... and other properties (see exercise 8-15)

Using the semantics

- For us, the primary use of the semantics is to serve as a specification for an interpreter
- We'll see this first with the ML-based interpreter for uScheme

A look backward

- Impcore characteristics:
  - Program by defining functions
  - Run programs by evaluating expressions
  - Recursion
  - Lispish concrete syntax
  - Separate environments for global variables, parameters, and functions
  - Formal operational syntax

A look forward

- uScheme: a dialect of Lisp
- Extends Impcore
  - first-class functions
  - S-expressions
  - anonymous functions
  - local variables