# Design of Programming Languages

## CS263

Prof. Kathy Yelick

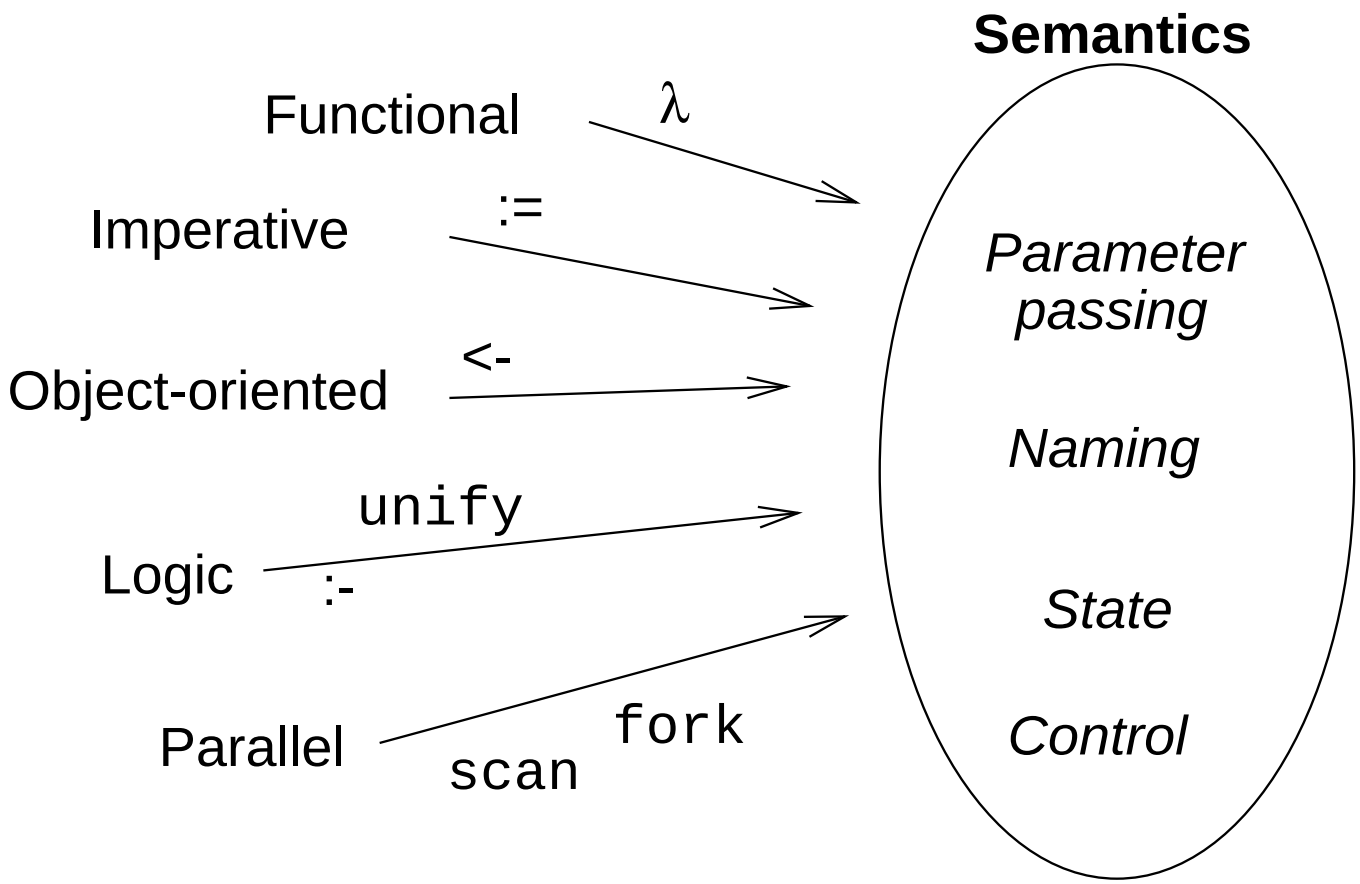
777 Soda Hall

e-mail: yelick@cs.berkeley.edu

class home page: http://www.cs.berkeley.edu/~yelick/263

secretary: Bob Untiedt, bobu@cs

# Programming Paradigms



## Evaluating Languages

Two schools of thought in language design:

fast prototyping vs. production

Language evaluations based on:

Can it be implemented efficiently?

Ease of writing programs, programmer productivity.

Readability, maintainability, programming in the large.

Does is satisfy a need in some problem domain?

How are widely-used languages "chosen"?

# An Approach to Language Research

Most real programming languages are large and complex, and many are tied to a programming environment.

Two approaches:

1) consider example programs and write them in the language of interest

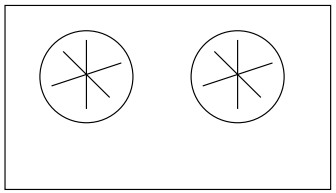
2) study a smaller language that has the essential features of the language

- prove formally that certain properties hold
- show conversions between different languages to understand the relationship between various features
- build mathematical models of the language to characterize its features

# Semantic Foundations

*A more precise basis for evaluation...*

## Operational Semantics



*machine define by rewrite rules*

$$\frac{\text{(if true E1 E2)}}{\text{E1}}$$

## Denotational Semantics

1 → *the number one*

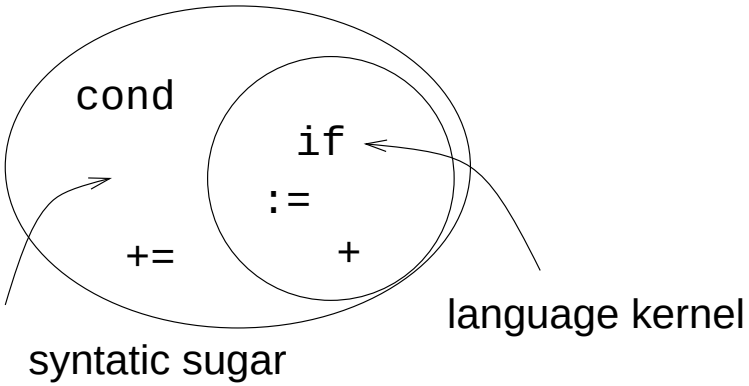
int f (int x) {return x;} → *the identity function*

*an interpretation for each construct*

# Semantic Foundations

## Desugaring

*reduce a larger to smaller "kernel"*




## Axiomatic Semantics

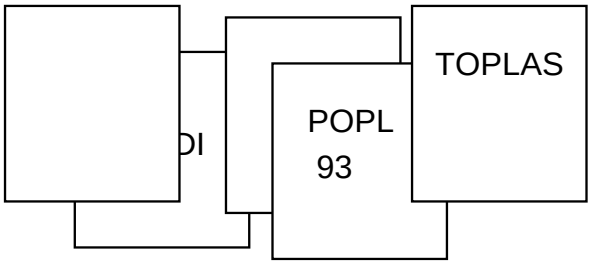
*rules for proving properties of programs*

```
P {program statment} Q
```

Pre and Post conditions

# Goals of this course

Programming language research




ML Lisp C++ Smalltalk Prolog Eiffel CLU

Reasoning about correctness

"Programming language" design

SQL VHDL LaTex Make AVL Postscript

# Digression on Syntax

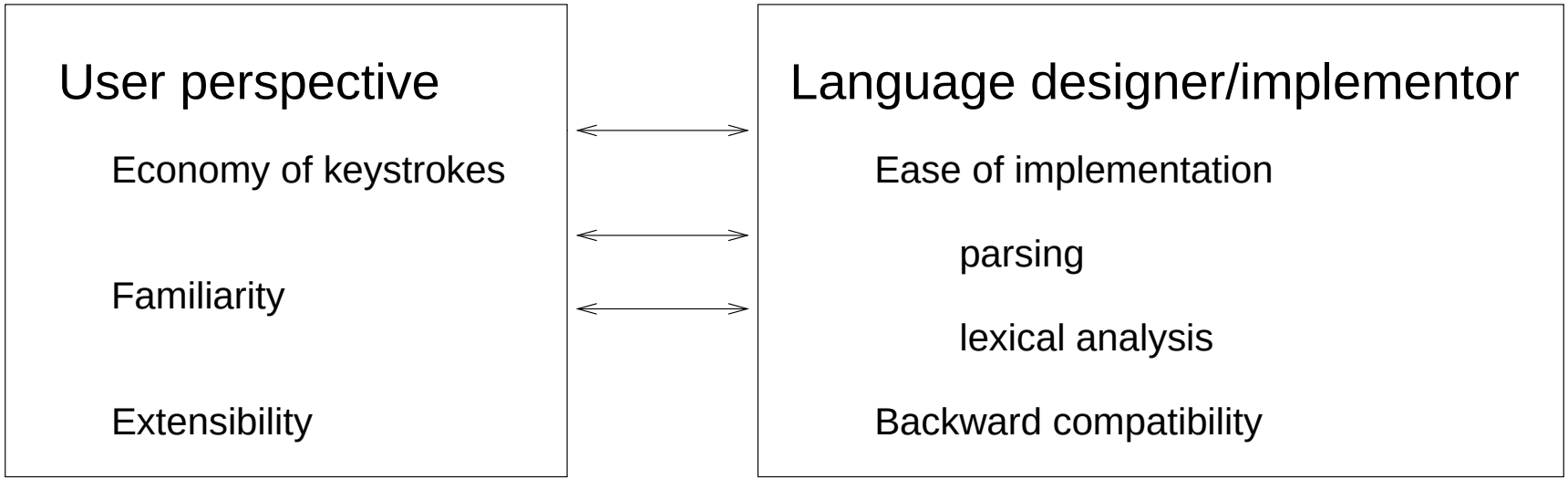
```
BEGIN          {start = 1;}
/started on/   { last = substr($0, index($0, ";  ")+2)
```

```
find / \( -name a.out -o -name '*.0' \)
  -atime +7 -exec rm {} ;
```

```
template <class Type>
void argle<Type>::display(Type *a[])
     { cout << bar << *(a+3) << endl;
```

*...syntax is important*

# Syntactic Motivations




s(ymbolic)-expressions, a simple syntax

```
(apply
   (apply (lambda (x)(lambda (y)(x+y))) 3)
   (+ 3 5)
```

# Abstract vs. Concret Syntax

Concrete Syntax: syntax of actual programming language. Typically a set of strings.

Abstract Syntax: an idealized syntax for the language that omits certain details required for parsing or reading the language.

Both of these define a tree structure to any correct (well-formed) program, called a parse tree or abstract syntax tree (AST).

Concrete

E ::= T | E + T
T ::= F | T * F
F ::= ( E ) | ID | NUM |
F(E) | op+ | op*

Abstract

E ::= IND | NUM | E E
IND ::= + | * | ID

# A Small Model Language

PCF: a language for Programming Computable Functions, based on the lambda calculus. Defined by Dana Scott. Our description is from a book chapter by John Mitchell (see the class home page for a copy -- about 80 pages).

PCF is:

* based on the typed (rather than untyped) lambda calculus
* is a function language
* uses lazy (left-most) evaluation

Read pages 50-68 for next time.

# PCF Overview

Booleans and natural numbers:

general form of conditionals

if <boolean> then <boolean> else <boolean>

if <boolean> then <natural_number> else <natural_number>

specific example

if Eq? (6+5) 11 then (1+1) else 43

Implicit function definitions:

x + 6 is a function that maps one integer to another