Programming Languages and Compilers (CS 421)

Elsa L Gunter

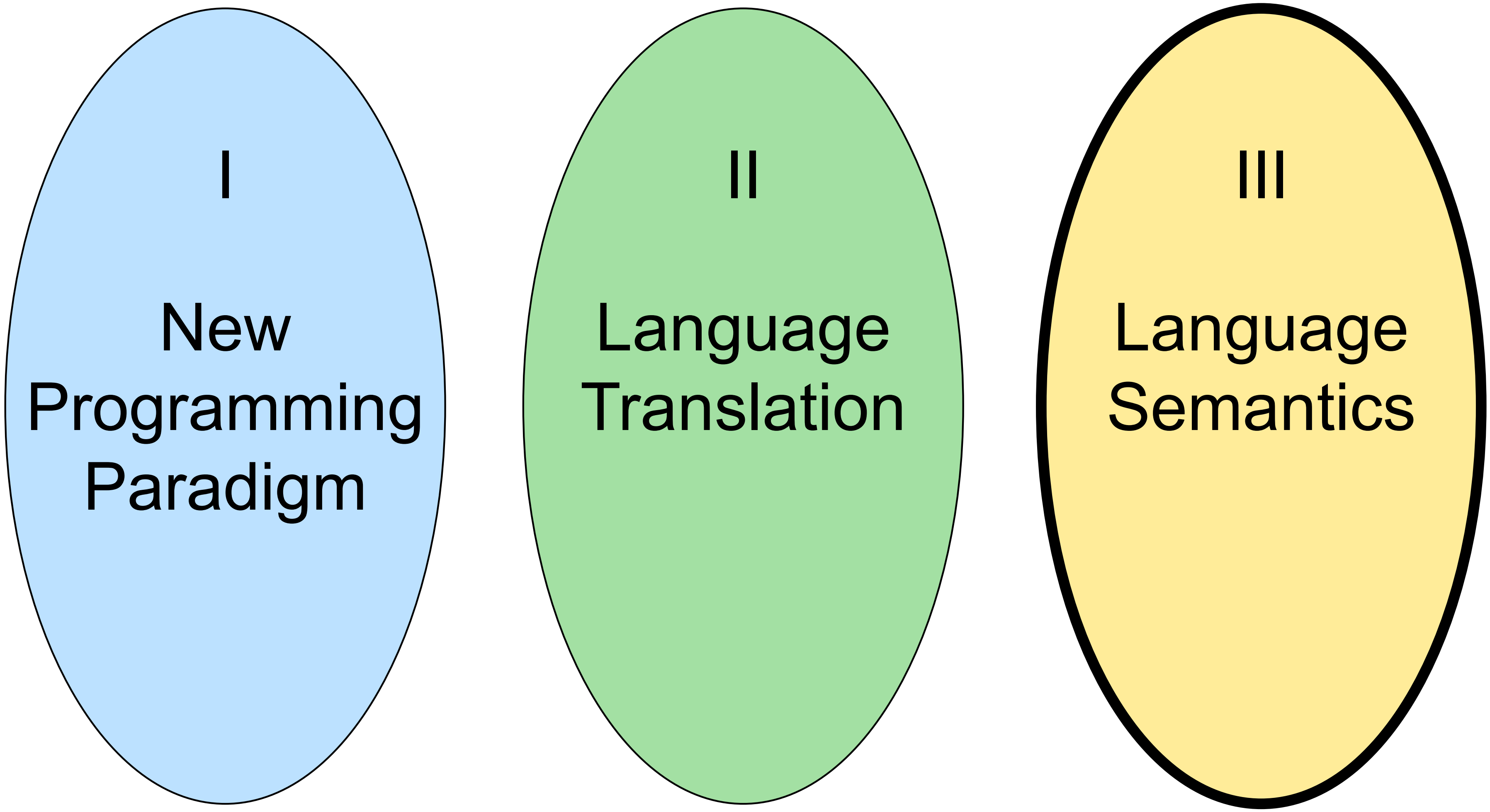
2112 SC, UIUC

https://courses.engr.illinois.edu/cs421/sp2023

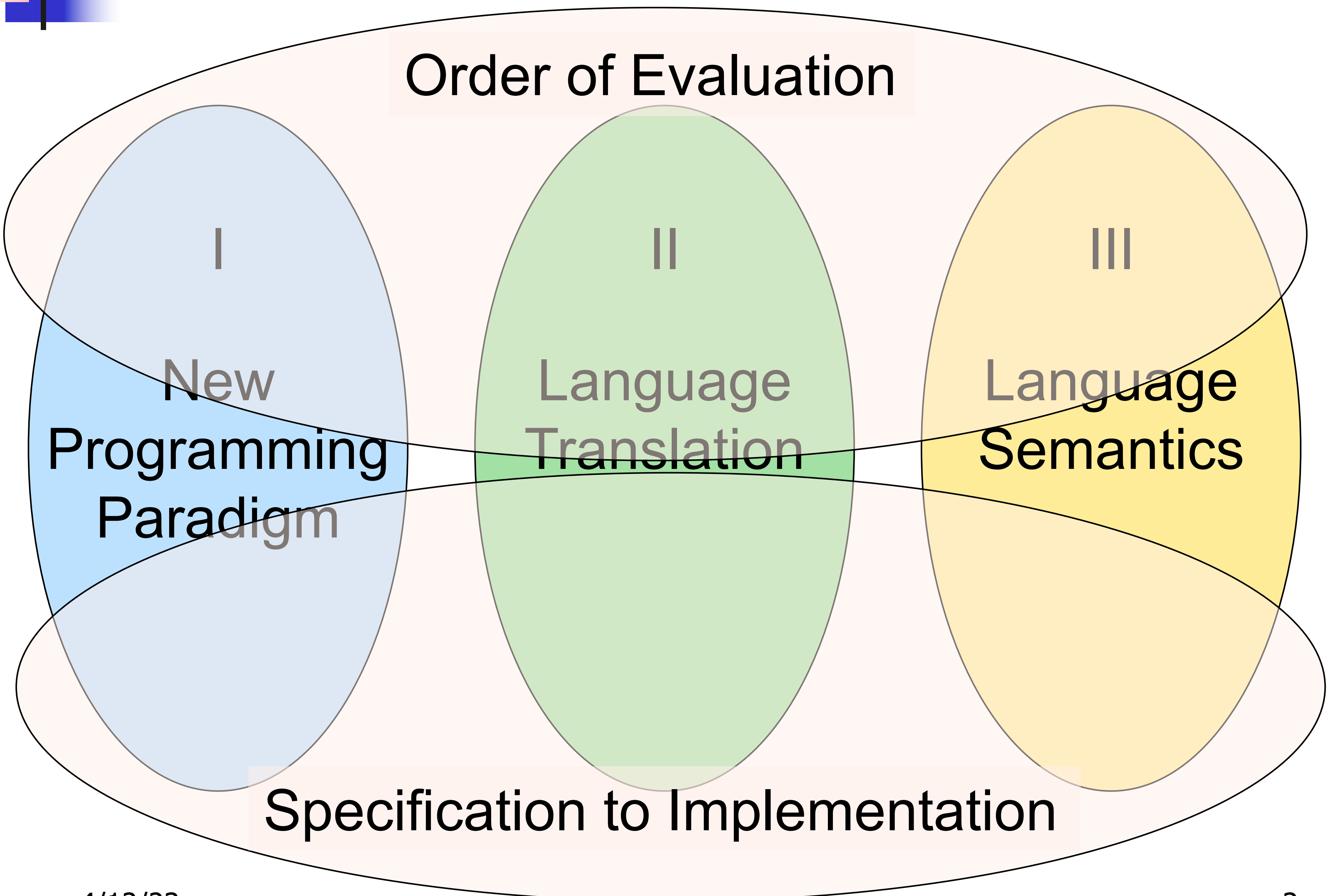
Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Programming Languages & Compilers

Three Main Topics of the Course



Programming Languages & Compilers



Programming Languages & Compilers

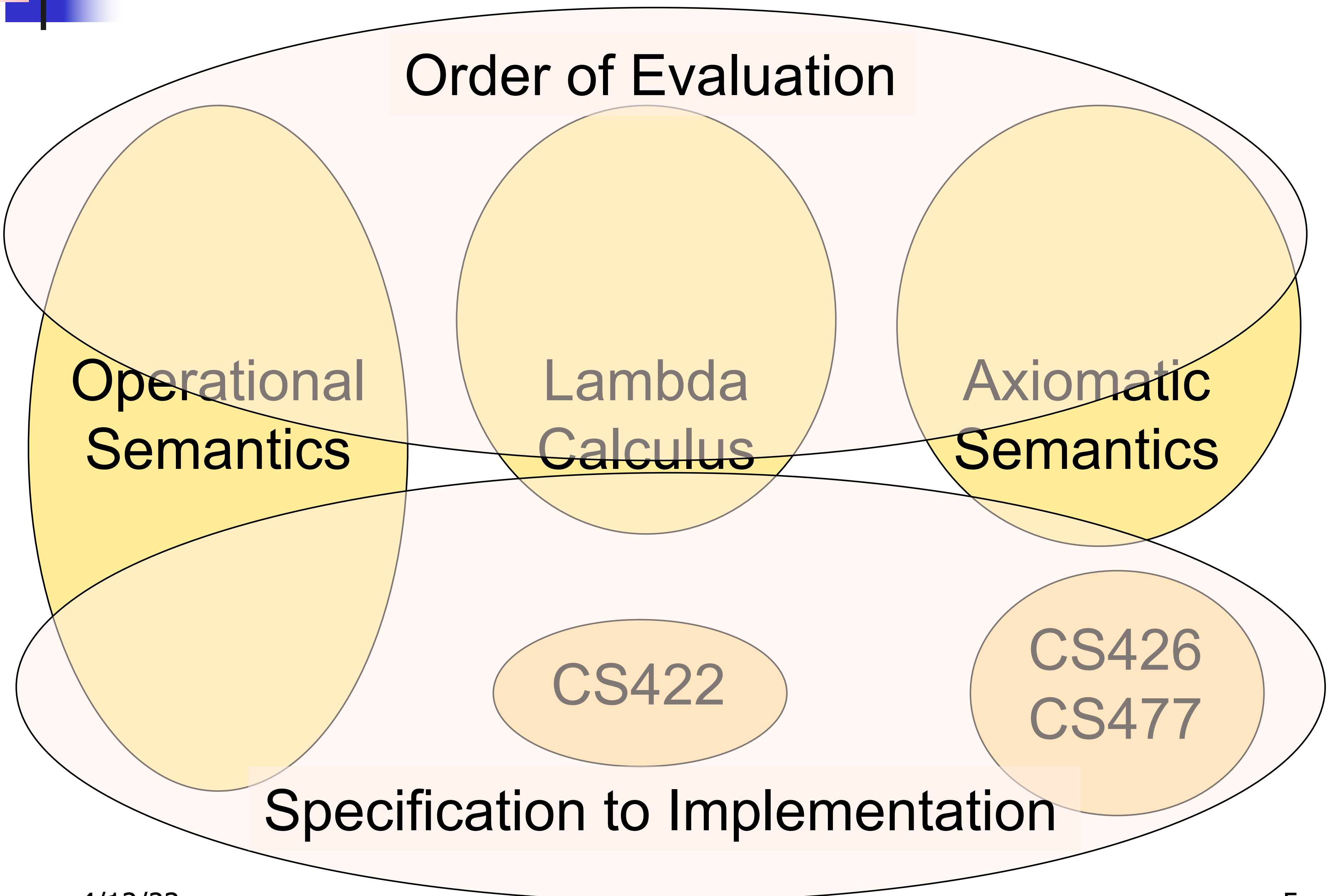
III : Language Semantics

Operational Semantics

Lambda Calculus

Axiomatic Semantics

Programming Languages & Compilers



Semantics

- Expresses the meaning of syntax
- Static semantics
  - Meaning based only on the form of the expression without executing it
  - Usually restricted to type checking / type inference

Dynamic semantics

- Method of describing meaning of executing a program
- Several different types:
  - Operational Semantics
  - Axiomatic Semantics
  - Denotational Semantics

Dynamic Semantics

- Different languages better suited to different types of semantics
- Different types of semantics serve different purposes

Operational Semantics

- Start with a simple notion of machine
- Describe how to execute (implement) programs of language on virtual machine, by describing how to execute each program statement (ie, following the *structure* of the program)
- Meaning of program is how its execution changes the state of the machine
- Useful as basis for implementations

Axiomatic Semantics

- Also called Floyd-Hoare Logic
- Based on formal logic (first order predicate calculus)
- Axiomatic Semantics is a logical system built from *axioms* and *inference rules*
- Mainly suited to simple imperative programming languages

Axiomatic Semantics

- Used to formally prove a property (*post-condition*) of the *state* (the values of the program variables) after the execution of program, assuming another property (*pre-condition*) of the state before execution
- Written :

  {Precondition} Program {Postcondition}
- Source of idea of *loop invariant*

Denotational Semantics

- Construct a function $\mathcal{M}$ assigning a mathematical meaning to each program construct
- Lambda calculus often used as the range of the meaning function
- Meaning function is compositional: meaning of construct built from meaning of parts
- Useful for proving properties of programs

Natural Semantics

- Aka "Big Step Semantics"
- Provide value for a program by rules and derivations, similar to type derivations
- Rule conclusions look like

$$(C, m) \Downarrow m'$$

or

$$(E, m) \Downarrow v$$

Simple Imperative Programming Language

- $I \in$ *Identifiers*
- $N \in$ *Numerals*
- B ::= true | false | B & B | B or B | not B
  | $E < E$ | $E = E$
- $E ::= N \mid I \mid E + E \mid E * E \mid E - E \mid - E$
- C ::= skip | C;C | I ::= E
  | if B then C else C fi | while B do C od

Natural Semantics of Atomic Expressions

- Identifiers: $(I,m) \Downarrow m(I)$
- Numerals are values: $(N,m) \Downarrow N$
- Booleans: $(\text{true},m) \Downarrow \text{true}$

  $(\text{false} ,m) \Downarrow \text{false}$

Booleans:

$$\frac{(B, m) \Downarrow \text{false}}{(B \,\&\, B', m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{true} \quad (B', m) \Downarrow b}{(B \,\&\, B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(B \text{ or } B', m) \Downarrow \text{true}} \qquad \frac{(B, m) \Downarrow \text{false} \quad (B', m) \Downarrow b}{(B \text{ or } B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(\text{not } B, m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{false}}{(\text{not } B, m) \Downarrow \text{true}}$$

Relations

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \sim V = b}{(E \sim E', m) \Downarrow b}$$

- By $U \sim V = b$, we mean does (the meaning of) the relation ~ hold on the meaning of U and V
- May be specified by a mathematical expression/equation or rules matching U and V

Arithmetic Expressions

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \; op \; V = N}{(E \; op \; E', m) \Downarrow N}$$

where *N* is the specified value for *U op V*

Commands

Skip: $(\text{skip}, m) \Downarrow m$

Assignment:

$$\frac{(E,m) \Downarrow V}{(I::=E,m) \Downarrow m[I <\text{--}\ V]}$$

Sequencing:

$$\frac{(C,m) \Downarrow m' \quad (C',m') \Downarrow m''}{(C;C', m) \Downarrow m''}$$

If Then Else Command

$$\frac{(B,m) \Downarrow \text{true} \quad (C,m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'}$$

$$\frac{(B,m) \Downarrow \text{false} \quad (C',m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'}$$

While Command

$$\frac{(B,m) \Downarrow \text{false}}{(\text{while } B \text{ do } C \text{ od}, m) \Downarrow m}$$

$$\frac{(B,m)\Downarrow\text{true} \quad (C,m)\Downarrow m' \quad (\text{while } B \text{ do } C \text{ od}, m')\Downarrow m''}{(\text{while } B \text{ do C od}, m) \Downarrow m''}$$

Example: If Then Else Rule

$$\overline{\text{(if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi, } \{x \text{ -> } 7\}) \Downarrow\ ?}$$

Example: If Then Else Rule

$$\frac{\overline{(x > 5, \{x \to 7\}) \Downarrow ?}}{(\text{if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi}, \{x \to 7\}) \Downarrow ?}$$

Example: Arith Relation

$$\frac{\dfrac{? > ? = ? \quad \dfrac{}{} }{}}{}$$

? > ? = ?

$$\frac{\dfrac{(x,\{x\text{->}7\})\Downarrow ? \quad (5,\{x\text{->}7\})\Downarrow ?}{(x > 5, \{x \text{->} 7\})\Downarrow ?}}{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi}, \{x \text{->} 7\}) \Downarrow ?}$$

Example: Identifier(s)

$$\dfrac{\dfrac{\dfrac{(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5}{}\ 7 > 5 = \text{true}}{(x > 5, \{x \text{->} 7\})\Downarrow ?}}{(\text{if } x > 5 \text{ then } y{:}= 2 + 3 \text{ else } y{:}=3 + 4 \text{ fi}, \{x \text{->} 7\}) \Downarrow ?}$$

Example: Arith Relation

7 > 5 = true

(x,{x->7})⇓7   (5,{x->7})⇓5

(x > 5, {x -> 7})⇓true

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓ ?

Example: If Then Else Rule

7 > 5 = true

(x,{x->7})⇓7   (5,{x->7})⇓5
____

(x > 5, {x -> 7})⇓true

____

(y:= 2 + 3,{x-> 7})

⇓ ?   .

____

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓ ?

Example: Assignment

7 > 5 = true

(x,{x->7})⇓7 (5,{x->7})⇓5

(x > 5, {x -> 7})⇓true

(2+3, {x->7})⇓?

(y:= 2 + 3,{x-> 7})

⇓ ?

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓ ?

Example: Arith Op

? + ? = ?

(2,{x->7})⇓?   (3,{x->7}) ⇓?

7 > 5 = true   (2+3, {x->7})⇓?

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true   ⇓?   .

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓ ?

Example: Numerals

2 + 3 = 5

(2,{x->7})⇓2   (3,{x->7}) ⇓3

(2+3, {x->7})⇓?

7 > 5 = true

(x,{x->7})⇓7   (5,{x->7})⇓5

(y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true

⇓ ?

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓ ?

Example: Arith Op

2 + 3 = 5

(2,{x->7})⇓2 (3,{x->7}) ⇓3

(2+3, {x->7})⇓5

7 > 5 = true

(x,{x->7})⇓7 (5,{x->7})⇓5

(y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true ⇓?

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓ ?

Example: Assignment

2 + 3 = 5

(2,{x->7})⇓2   (3,{x->7}) ⇓3
____

7 > 5 = true   (2+3, {x->7})⇓5
____

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})
____

(x > 5, {x -> 7})⇓true   ⇓ {x->7, y->5}
____

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓?

Example: If Then Else Rule

2 + 3 = 5

(2,{x->7})⇓2   (3,{x->7}) ⇓3

___

7 > 5 = true   (2+3, {x->7})⇓5

___

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

___

(x > 5, {x -> 7})⇓true   ⇓ {x->7, y->5}

___

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7}) ⇓ {x->7, y->5}

Let in Command

$$\frac{(E,m) \Downarrow v \quad (C,m[I\text{<-}v]) \Downarrow m'}{(\text{let } I = E \text{ in } C,\ m) \Downarrow m''}$$

Where $m''(y) = m'(y)$ for $y \neq I$ and $m''(I) = m(I)$ if $m(I)$ is defined, and $m''(I)$ is undefined otherwise

Example

$$\frac{(5,\{x\text{->}17\}) \Downarrow 5 \qquad \dfrac{\dfrac{(x,\{x\text{->}5\}) \Downarrow 5 \quad (3,\{x\text{->}5\}) \Downarrow 3}{(x+3,\{x\text{->}5\}) \Downarrow 8}}{(x:=x+3,\{x\text{->}5\}) \Downarrow \{x\text{->}8\}}}{(\text{let } x = 5 \text{ in } (x:=x+3),\ \{x \text{ -> } 17\}) \Downarrow\ ?}$$

Example

$$\dfrac{(5,\{x\text{->}17\}) \Downarrow 5 \qquad \dfrac{\dfrac{(x,\{x\text{->}5\}) \Downarrow 5 \quad (3,\{x\text{->}5\}) \Downarrow 3}{(x+3,\{x\text{->}5\}) \Downarrow 8}}{(x{:=}x+3,\{x\text{->}5\}) \Downarrow \{x\text{->}8\}}}{(\text{let } x = 5 \text{ in } (x{:=}x+3),\ \{x \text{ -> } 17\}) \Downarrow \{x\text{->}17\}}$$

Comment

- Simple Imperative Programming Language introduces variables *implicitly* through assignment
- The let-in command introduces scoped variables *explictly*
- Clash of constructs apparent in awkward semantics

Interpretation Versus Compilation

- A **compiler** from language L1 to language L2 is a program that takes an L1 program and for each piece of code in L1 generates a piece of code in L2 of same meaning
- An **interpreter** of L1 in L2 is an L2 program that executes the meaning of a given L1 program
- Compiler would examine the body of a loop once; an interpreter would examine it every time the loop was executed

Interpreter

- An *Interpreter* represents the operational semantics of a language L1 (source language) in the language of implementation L2 (target language)
- Built incrementally
  - Start with literals
  - Variables
  - Primitive operations
  - Evaluation of expressions
  - Evaluation of commands/declarations