

Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC

<http://courses.engr.illinois.edu/cs421>

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

4/4/24

1

Disambiguating a Grammar

- Idea: Each non-terminal represents all strings having some property
 - In fact each string of terminals and non-terminals represents all strings having some property (Remember reg exp semantics)
- Identify these properties (often in terms of things that can't happen)
- Use these properties to inductively guarantee every string in language has a unique parse

4/4/24

2

Steps to Grammar Disambiguation

- Identify the rules and a smallest use that display ambiguity
- Decide which parse to keep; why should others be thrown out?
- What syntactic restrictions on subexpressions are needed to throw out the bad (while keeping the good)?
- Add a new non-terminal and rules to describe this set of restricted subexpressions (called stratifying, or refactoring)
- Characterize each non-terminal by a language invariant**
- Replace old rules to use new non-terminals
- Rinse and repeat

4/4/24

3

Operator Precedence

- Operators of highest precedence evaluated first (bind more tightly).
- Precedence for infix binary operators given in following table
- Needs to be reflected in grammar
 - Saw how last class

10/4/07

4

Precedence Table - Sample

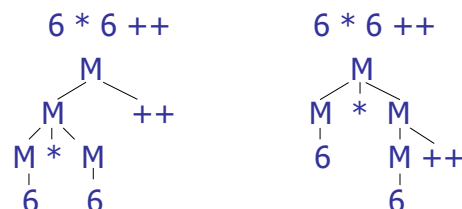
	Fortan	Pascal	C/C++	Ada	SML
highest	**	*, /, div, mod	++, --	**	div, mod, /, *
	*, /	+, -	*, /, %	*, /, mod	+, -, ^
	+, -		+, -	+, -	::

10/4/07

5

More Disambiguating Grammars

- $M ::= M * M \mid (M) \mid M ++ \mid 6$
- Ambiguous because of associativity of $*$
- because of conflict between $*$ and $++$:



4/4/24

6

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- How to disambiguate?
- Choose associativity for *
- Choose precedence between * and ++
- Four possibilities
- Four different approaches
- Some easier than others
- Will do --- My choice, then yours if time

4/4/24

7

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- Think about $6 * 6 ++ * 6 * 6 ++$
- Let's start with observations
- If * binds less tightly than ++, then no * can be the immediate subtree to a ++.
 - We would need a language for things that don't parse as *
- If * binds more tightly than ++, then ...
- The right subtree to * can't be a ++
- But the left can!
- Need different languages of the left and right

4/4/24

8

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- * higher prec than ++
 - $6 * 6 ++ \quad 6 ++ * 6$
- $M ::= M ++ \mid \text{StarExp} \mid (M) \mid 6$
- What is **StarExp**
- It is everything that parses as a * and can't parse as a ++
- But what is the associativity of *?
- I'll choose left

4/4/24

9

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- * higher prec than ++
 - $6 * 6 ++ * 6 ++ * 6$
- * Left assoc
- $M ::= M ++ \mid \text{StarExp} \mid (M) \mid 6$
- $\text{StarExp} ::= \text{PossStar} * \text{NotStarNotPlusPlus}$
- What is **PossStar**? It could be a *, but it also doesn't have to be.
- Can it be ++? YES! It can be anything
- It is **M**!

4/4/24

10

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- * higher prec than ++
 - $6 * 6 ++ \quad 6 ++ * 6$
- * Left assoc
- $M ::= M ++ \mid \text{StarExp} \mid (M) \mid 6$
- $\text{StarExp} ::= M * \text{NotStarNotPlusPlus}$

4/4/24

11

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- * higher prec than ++
 - $6 * 6 ++ \quad 6 ++ * 6$
- * Left assoc
- $M ::= M ++ \mid \text{StarExp} \mid (M) \mid 6$
- $\text{StarExp} ::= M * \text{NotStarNotPlusPlus}$
- But what is **NotStarNotPlusPlus**?
- Well, the other two original rules: $(M) \mid 6$

4/4/24

12

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- * higher prec than ++
 - $6 * 6 ++ 6 ++ * 6$
- * Left assoc
- $M ::= M ++ \mid \text{StarExp} \mid (M) \mid 6$
- $\text{StarExp} ::= M * \text{NotStarNotPlusPlus}$
- $\text{NotStarNotPlusPlus} ::= (M) \mid 6$
- But we have $(M) \mid 6$ twice, and it's the same language each time. Let's have one

4/4/24

13

$M ::= M * M \mid (M) \mid M ++ \mid 6$

- * higher prec than ++
 - $6 * 6 ++ 6 ++ * 6$
- * Left assoc
- $M ::= M ++ \mid \text{StarExp} \mid \text{NotStarNotPlusPlus}$
- $\text{StarExp} ::= M * \text{NotStarNotPlusPlus}$
- $\text{NotStarNotPlusPlus} ::= (M) \mid 6$

4/4/24

14

Parser Code

- `<grammar>.ml` defines one parsing function per entry point
- Parsing function takes a lexing function (lexer buffer to token) and a lexer buffer as arguments
- Returns semantic attribute of corresponding entry point

4/4/24

15

Ocamlyacc Input

- File format:

```
%{  
  <header>  
}%  
  <declarations>  
%%  
  <rules>  
%%  
  <trailer>
```

4/4/24

16

Ocamlyacc `<header>`

- Contains arbitrary Ocaml code
- Typically used to give types and functions needed for the semantic actions of rules and to give specialized error recovery
- May be omitted
- `<footer>` similar. Possibly used to call parser

4/4/24

17

Ocamlyacc `<declarations>`

- `%token symbol ... symbol`
 - Declare given symbols as tokens
- `%token <type> symbol ... symbol`
 - Declare given symbols as token constructors, taking an argument of type `<type>`
- `%start symbol ... symbol`
 - Declare given symbols as entry points; functions of same names in `<grammar>.ml`

4/4/24

18

Ocamlyacc <declarations>

- **%type** <type> symbol ... symbol
Specify type of attributes for given symbols.
Mandatory for start symbols
- **%left** symbol ... symbol
- **%right** symbol ... symbol
- **%nonassoc** symbol ... symbol
Associate precedence and associativity to given symbols. Same line, same precedence; earlier line, lower precedence (broadest scope)

4/4/24

19

Ocamlyacc <rules>

- **nonterminal** :
symbol ... symbol { semantic_action }
| ...
| symbol ... symbol { semantic_action }
;
- Semantic actions are arbitrary Ocaml expressions
- Must be of same type as declared (or inferred) for *nonterminal*
- Access semantic attributes (values) of symbols by position: \$1 for first symbol, \$2 to second ...

4/4/24

20

Example - Base types

```
(* File: expr.ml *)
type expr =
  Term_as_Expr of term
  | Plus_Expr of (term * expr)
  | Minus_Expr of (term * expr)
and term =
  Factor_as_Term of factor
  | Mult_Term of (factor * term)
  | Div_Term of (factor * term)
and factor =
  Id_as_Factor of string
  | Parenthesized_Expr_as_Factor of expr
```

4/4/24

21

Example - Lexer (exprlex.mll)

```
{ (*open Exprparse*) }
let numeric = ['0' - '9']
let letter = ['a' - 'z' 'A' - 'Z']
rule token = parse
  | "+" {Plus_token}
  | "-" {Minus_token}
  | "*" {Times_token}
  | "/" {Divide_token}
  | "(" {Left_parenthesis}
  | ")" {Right_parenthesis}
  | letter (letter|numeric|"_")* as id {Id_token id}
  | [' '\t' '\n'] {token lexbuf}
  | eof {EOL}
```

4/4/24

22

Example - Parser (exprparse.mly)

```
%{ open Expr
%}
%token <string> Id_token
%token Left_parenthesis Right_parenthesis
%token Times_token Divide_token
%token Plus_token Minus_token
%token EOL
%start main
%type <expr> main
%%
```

4/4/24

23

Example - Parser (exprparse.mly)

```
expr:
  term
  { Term_as_Expr $1 }
| term Plus_token expr
  { Plus_Expr ($1, $3) }
| term Minus_token expr
  { Minus_Expr ($1, $3) }
```

4/4/24

24

Example - Parser (exprparse.mly)

```
term:
  factor
    { Factor_as_Term $1 }
| factor Times_token term
    { Mult_Term ($1, $3) }
| factor Divide_token term
    { Div_Term ($1, $3) }
```

4/4/24

25

Example - Parser (exprparse.mly)

```
factor:
  Id_token
    { Id_as_Factor $1 }
| Left_parenthesis expr Right_parenthesis
    { Parenthesized_Expr_as_Factor $2 }
main:
  | expr EOL
    { $1 }
```

4/4/24

26

Example - Using Parser

```
# #use "expr.ml";;
...
# #use "exprparse.ml";;
...
# #use "exprlex.ml";;
...
# let test s =
  let lexbuf = Lexing.from_string (s^"\n") in
  main token lexbuf;;
```

4/4/24

27

Example - Using Parser

```
# test "a + b";;
- : expr =
Plus_Expr
(Factor_as_Term (Id_as_Factor "a"),
Term_as_Expr (Factor_as_Term
(Id_as_Factor "b")))
```

4/4/24

28

LR Parsing

- Read tokens left to right (L)
- Create a rightmost derivation (R)
- How is this possible?
- Start at the bottom (left) and work your way up
- Last step has only one non-terminal to be replaced so is right-most
- Working backwards, replace mixed strings by non-terminals
- Always proceed so that there are no non-terminals to the right of the string to be replaced

4/4/24

29

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= (0 + 1) + 0$ shift

4/4/24

30

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= (0 + 1) + 0$ shift
 $= (0 + 1) + 0$ shift

4/4/24 31

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$\Rightarrow (0 + 1) + 0$ reduce
 $= (0 + 1) + 0$ shift
 $= (0 + 1) + 0$ shift

4/4/24 32

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= (\langle \text{Sum} \rangle + 1) + 0$ shift
 $\Rightarrow (0 + 1) + 0$ reduce
 $= (0 + 1) + 0$ shift
 $= (0 + 1) + 0$ shift

4/4/24 33

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= (\langle \text{Sum} \rangle + 1) + 0$ shift
 $= (\langle \text{Sum} \rangle + 1) + 0$ shift
 $\Rightarrow (0 + 1) + 0$ reduce
 $= (0 + 1) + 0$ shift
 $= (0 + 1) + 0$ shift

4/4/24 34

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$\Rightarrow (\langle \text{Sum} \rangle + 1) + 0$ reduce
 $= (\langle \text{Sum} \rangle + 1) + 0$ shift
 $= (\langle \text{Sum} \rangle + 1) + 0$ shift
 $\Rightarrow (0 + 1) + 0$ reduce
 $= (0 + 1) + 0$ shift
 $= (0 + 1) + 0$ shift

4/4/24 35

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + 0$ reduce
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + 0$ reduce
 $= (\langle \text{Sum} \rangle + 1) + 0$ shift
 $= (\langle \text{Sum} \rangle + 1) + 0$ shift
 $\Rightarrow (0 + 1) + 0$ reduce
 $= (0 + 1) + 0$ shift
 $= (0 + 1) + 0$ shift

4/4/24 36

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \bullet \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet$	reduce
$\Rightarrow \langle \text{Sum} \rangle + 0 \bullet$	reduce
$= \langle \text{Sum} \rangle + \bullet 0$	shift
$= \langle \text{Sum} \rangle \bullet + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle) \bullet + 0$	reduce
$= (\langle \text{Sum} \rangle \bullet) + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
$\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
$\Rightarrow (0 \bullet + 1) + 0$	reduce
$= (\bullet 0 + 1) + 0$	shift
$= \bullet (0 + 1) + 0$	shift

4/4/24

43

Example

(0 + 1) + 0



4/4/24

44

Example

(0 + 1) + 0



4/4/24

45

Example

(0 + 1) + 0



4/4/24

46

Example

($\langle \text{Sum} \rangle$
0 + 1) + 0



4/4/24

47

Example

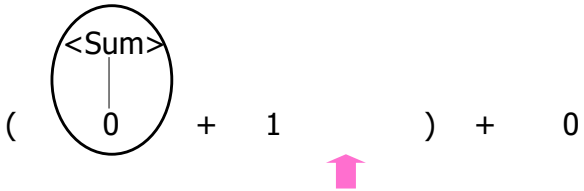
($\langle \text{Sum} \rangle$
0 + 1) + 0



4/4/24

48

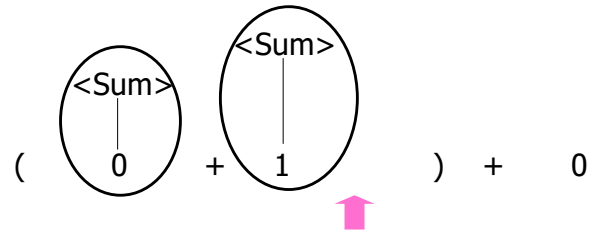
Example



4/4/24

49

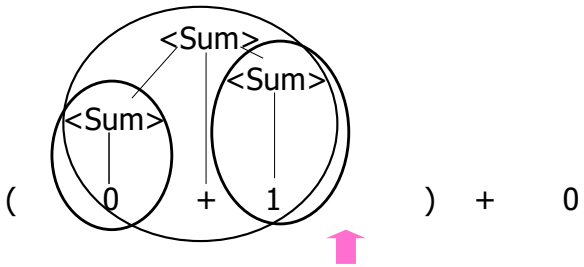
Example



4/4/24

50

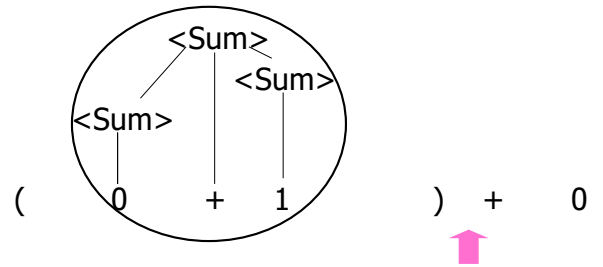
Example



4/4/24

51

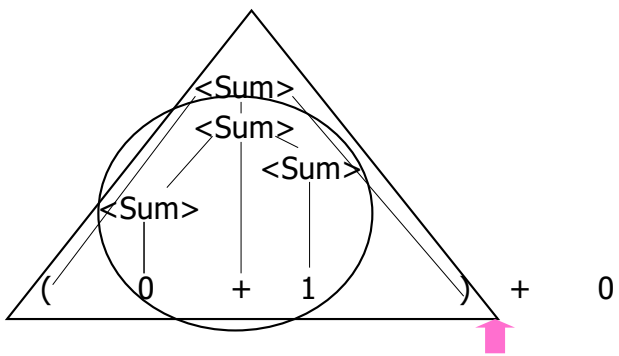
Example



4/4/24

52

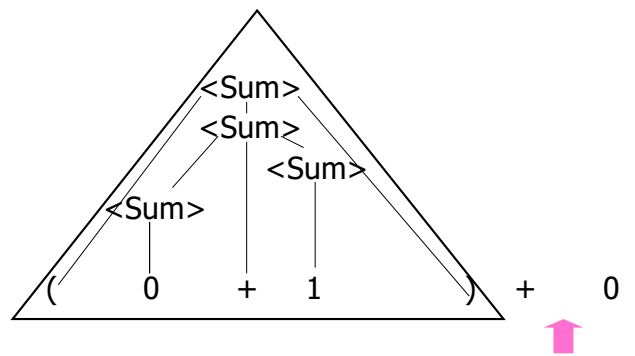
Example



4/4/24

53

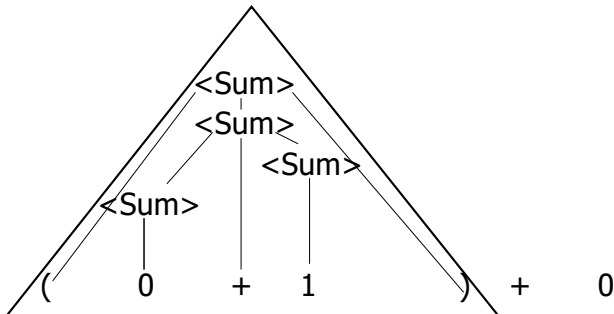
Example



4/4/24

54

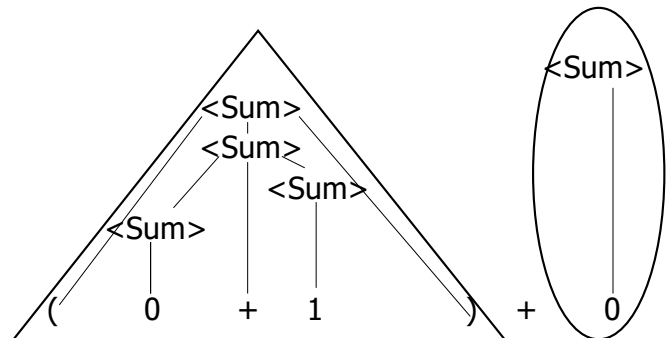
Example



4/4/24

55

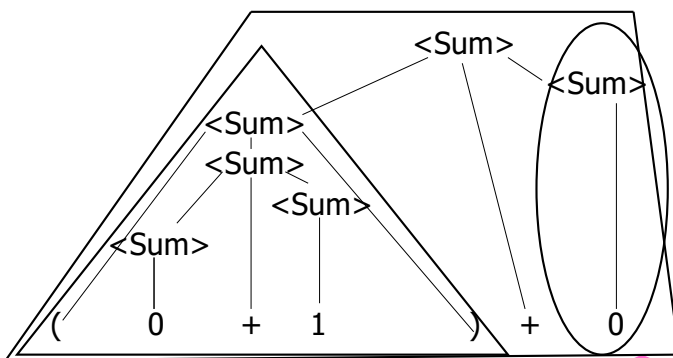
Example



4/4/24

56

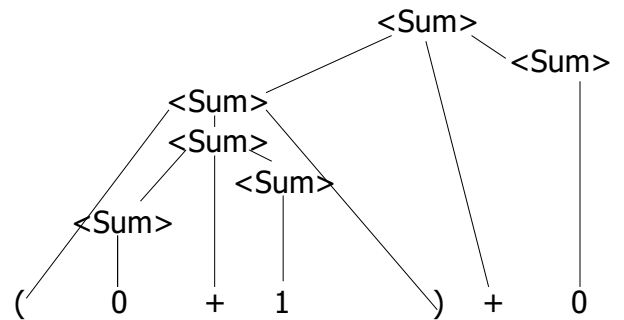
Example



4/4/24

57

Example



4/4/24

58

LR Parsing Tables

- Build a pair of tables, Action and Goto, from the grammar
 - This is the hardest part, we omit here
 - Rows labeled by states
 - For Action, columns labeled by terminals and “end-of-tokens” marker
 - (more generally strings of terminals of fixed length)
 - For Goto, columns labeled by non-terminals

4/4/24

59

Action and Goto Tables

- Given a state and the next input, Action table says either
 - **shift** and go to state n , or
 - **reduce** by production k (explained in a bit)
 - **accept** or **error**
- Given a state and a non-terminal, Goto table says
 - go to state m

4/4/24

60

LR(i) Parsing Algorithm

- Based on push-down automata
- Uses states and transitions (as recorded in Action and Goto tables)
- Uses a stack containing states, terminals and non-terminals

4/4/24

61

LR(i) Parsing Algorithm

0. Insure token stream ends in special “end-of-tokens” symbol
1. Start in state 1 with an empty stack
2. Push **state**(1) onto stack
- 3. Look at next i tokens from token stream ($toks$) (don't remove yet)
4. If top symbol on stack is **state**(n), look up action in Action table at $(n, toks)$

4/4/24

62

LR(i) Parsing Algorithm

5. If action = **shift** m ,
 - a) Remove the top token from token stream and push it onto the stack
 - b) Push **state**(m) onto stack
 - c) Go to step 3

4/4/24

63

LR(i) Parsing Algorithm

6. If action = **reduce** k where production k is $E ::= u$
 - a) Remove $2 * \text{length}(u)$ symbols from stack (u and all the interleaved states)
 - b) If new top symbol on stack is **state**(m), look up new state p in $\text{Goto}(m, E)$
 - c) Push E onto the stack, then push **state**(p) onto the stack
 - d) Go to step 3

4/4/24

64

LR(i) Parsing Algorithm

7. If action = **accept**
 - Stop parsing, return success
8. If action = **error**,
 - Stop parsing, return failure

4/4/24

65

Adding Synthesized Attributes

- Add to each **reduce** a rule for calculating the new synthesized attribute from the component attributes
- Add to each non-terminal pushed onto the stack, the attribute calculated for it
- When performing a **reduce**,
 - gather the recorded attributes from each non-terminal popped from stack
 - Compute new attribute for non-terminal pushed onto stack

4/4/24

66

Shift-Reduce Conflicts

- **Problem:** can't decide whether the action for a state and input character should be **shift** or **reduce**
- Caused by ambiguity in grammar
- Usually caused by lack of associativity or precedence information in grammar

4/4/24

67

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\bullet 0 + 1 + 0$ shift
 $\rightarrow 0 \bullet + 1 + 0$ reduce
 $\rightarrow \langle \text{Sum} \rangle \bullet + 1 + 0$ shift
 $\rightarrow \langle \text{Sum} \rangle + \bullet 1 + 0$ shift
 $\rightarrow \langle \text{Sum} \rangle + 1 \bullet + 0$ reduce
 $\rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet + 0$

4/4/24

68

Example - cont

- **Problem:** shift or reduce?
- You can shift-shift-reduce-reduce or reduce-shift-shift-reduce
- Shift first - right associative
- Reduce first- left associative

4/4/24

69

Reduce - Reduce Conflicts

- **Problem:** can't decide between two different rules to reduce by
- Again caused by ambiguity in grammar
- **Symptom:** RHS of one production suffix of another
- Requires examining grammar and rewriting it
- Harder to solve than shift-reduce errors

4/4/24

70

Example

■ $S ::= A \mid aB$ $A ::= abc$ $B ::= bc$

$\bullet abc$ shift
 $a \bullet bc$ shift
 $ab \bullet c$ shift
 $abc \bullet$

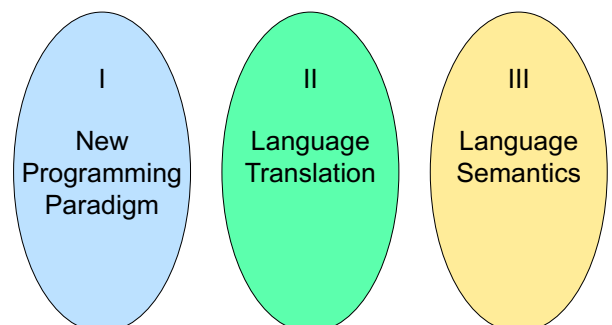
- Problem: reduce by $B ::= bc$ then by $S ::= aB$, or by $A ::= abc$ then $S ::= A$?

4/4/24

71

Programming Languages & Compilers

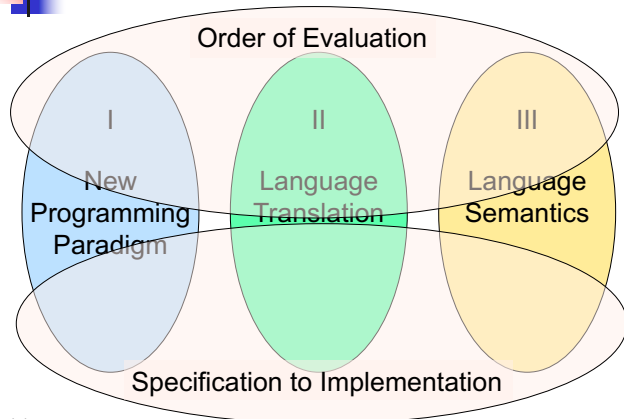
Three Main Topics of the Course



4/4/24

72

Programming Languages & Compilers

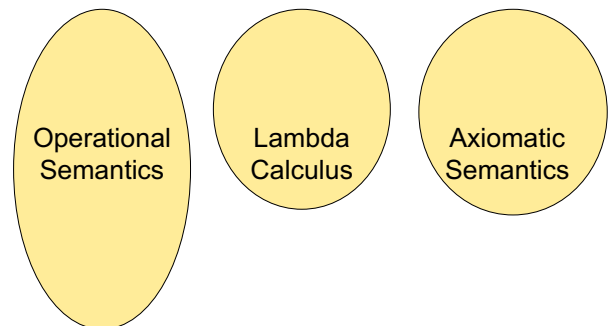


4/4/24

73

Programming Languages & Compilers

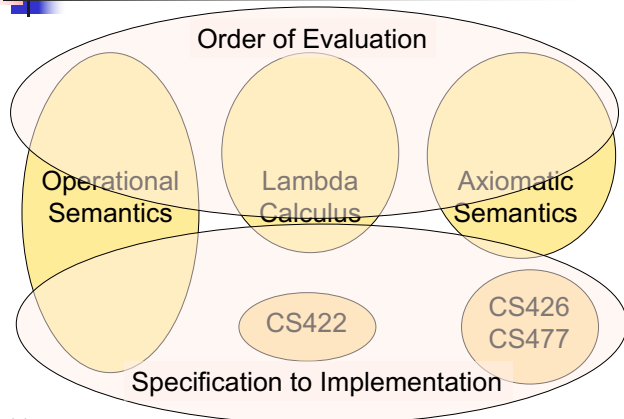
III : Language Semantics



4/4/24

74

Programming Languages & Compilers



4/4/24

75

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

4/4/24

76

Dynamic semantics

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

4/4/24

77

Dynamic Semantics

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

4/4/24

78

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

4/4/24

79

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

4/4/24

80

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*

4/4/24

81

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

4/4/24

82

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

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

4/4/24

83

Simple Imperative Programming Language

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

4/4/24

84

Natural Semantics of Atomic Expressions

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

4/4/24

85

Booleans:

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

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

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

4/4/24

86

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 \sim hold on the meaning of U and V
- May be specified by a mathematical expression/equation or rules matching U and V

4/4/24

87

Arithmetic Expressions

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

where N is the specified value for $U \text{ op } V$

4/4/24

88

Commands

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

Assignment: $\frac{(E, m) \Downarrow V}{(I ::= E, m) \Downarrow m[I \leftarrow V]}$

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

4/4/24

89

If Then Else Command

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

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

4/4/24

90

While Command

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

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

4/4/24

91

Example: If Then Else Rule

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

4/4/24

92

Example: If Then Else Rule

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

4/4/24

93

Example: Arith Relation

$$\frac{? > ? = ? \quad (x, \{x \rightarrow 7\}) \Downarrow ? \quad (5, \{x \rightarrow 7\}) \Downarrow ?}{(x > 5, \{x \rightarrow 7\}) \Downarrow ?}$$

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

4/4/24

94

Example: Identifier(s)

$$\frac{7 > 5 = \text{true} \quad (x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5}{(x > 5, \{x \rightarrow 7\}) \Downarrow ?}$$

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

4/4/24

95

Example: Arith Relation

$$\frac{7 > 5 = \text{true} \quad (x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true}}$$

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

4/4/24

96

Example: If Then Else Rule

$$\frac{\frac{7 > 5 = \text{true}}{(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5} \quad \frac{(y := 2 + 3, \{x \rightarrow 7\})}{\Downarrow ?}}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \quad \Downarrow ?} \cdot$$

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

4/4/24

97

Example: Assignment

$$\frac{\frac{7 > 5 = \text{true}}{(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5} \quad \frac{(2 + 3, \{x \rightarrow 7\}) \Downarrow ?}{(y := 2 + 3, \{x \rightarrow 7\}) \Downarrow ?}}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \quad \Downarrow ?} \cdot$$

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

4/4/24

98

Example: Arith Op

$$\frac{\frac{7 > 5 = \text{true}}{(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5} \quad \frac{\frac{? + ? = ?}{(2, \{x \rightarrow 7\}) \Downarrow ? \quad (3, \{x \rightarrow 7\}) \Downarrow ?} \quad \frac{(2 + 3, \{x \rightarrow 7\}) \Downarrow ?}{(y := 2 + 3, \{x \rightarrow 7\}) \Downarrow ?}}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \quad \Downarrow ?} \cdot$$

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

4/4/24

99

Example: Numerals

$$\frac{\frac{7 > 5 = \text{true}}{(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5} \quad \frac{\frac{2 + 3 = 5}{(2, \{x \rightarrow 7\}) \Downarrow 2 \quad (3, \{x \rightarrow 7\}) \Downarrow 3} \quad \frac{(2 + 3, \{x \rightarrow 7\}) \Downarrow ?}{(y := 2 + 3, \{x \rightarrow 7\}) \Downarrow ?}}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \quad \Downarrow ?} \cdot$$

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

4/4/24

100

Example: Arith Op

$$\frac{\frac{7 > 5 = \text{true}}{(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5} \quad \frac{\frac{2 + 3 = 5}{(2, \{x \rightarrow 7\}) \Downarrow 2 \quad (3, \{x \rightarrow 7\}) \Downarrow 3} \quad \frac{(2 + 3, \{x \rightarrow 7\}) \Downarrow 5}{(y := 2 + 3, \{x \rightarrow 7\}) \Downarrow ?}}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \quad \Downarrow ?} \cdot$$

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

4/4/24

101

Example: Assignment

$$\frac{\frac{7 > 5 = \text{true}}{(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5} \quad \frac{\frac{2 + 3 = 5}{(2, \{x \rightarrow 7\}) \Downarrow 2 \quad (3, \{x \rightarrow 7\}) \Downarrow 3} \quad \frac{(2 + 3, \{x \rightarrow 7\}) \Downarrow 5}{(y := 2 + 3, \{x \rightarrow 7\}) \Downarrow \{x \rightarrow 7, y \rightarrow 5\}}}}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \quad \Downarrow \{x \rightarrow 7, y \rightarrow 5\}} \cdot$$

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

4/4/24

102

Example: If Then Else Rule

$$\begin{array}{c}
 2 + 3 = 5 \\
 \frac{(2, \{x \rightarrow 7\}) \Downarrow 2 \quad (3, \{x \rightarrow 7\}) \Downarrow 3}{(2+3, \{x \rightarrow 7\}) \Downarrow 5} \\
 \frac{7 > 5 = \text{true} \quad (y := 2 + 3, \{x \rightarrow 7\}) \Downarrow 5}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \quad \Downarrow \{x \rightarrow 7, y \rightarrow 5\}} \\
 \frac{}{(if\ x > 5\ then\ y := 2 + 3\ else\ y := 3 + 4\ fi, \\ \{x \rightarrow 7\}) \Downarrow \{x \rightarrow 7, y \rightarrow 5\}}
 \end{array}$$

4/4/24

103

Let in Command

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

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

4/4/24

104

Example

$$\begin{array}{c}
 \frac{(x, \{x \rightarrow 5\}) \Downarrow 5 \quad (3, \{x \rightarrow 5\}) \Downarrow 3}{(x+3, \{x \rightarrow 5\}) \Downarrow 8} \\
 \frac{(5, \{x \rightarrow 17\}) \Downarrow 5 \quad (x := x+3, \{x \rightarrow 5\}) \Downarrow \{x \rightarrow 8\}}{(let\ x = 5\ in\ (x := x+3), \{x \rightarrow 17\}) \Downarrow ?}
 \end{array}$$

4/4/24

105

Example

$$\begin{array}{c}
 \frac{(x, \{x \rightarrow 5\}) \Downarrow 5 \quad (3, \{x \rightarrow 5\}) \Downarrow 3}{(x+3, \{x \rightarrow 5\}) \Downarrow 8} \\
 \frac{(5, \{x \rightarrow 17\}) \Downarrow 5 \quad (x := x+3, \{x \rightarrow 5\}) \Downarrow \{x \rightarrow 8\}}{(let\ x = 5\ in\ (x := x+3), \{x \rightarrow 17\}) \Downarrow \{x \rightarrow 17\}}
 \end{array}$$

4/4/24

106

Comment

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

4/4/24

107

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

4/4/24

108

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

4/4/24

109

Interpreter

- Takes abstract syntax trees as input
 - In simple cases could be just strings
- One procedure for each syntactic category (nonterminal)
 - eg one for expressions, another for commands
- If Natural semantics used, tells how to compute final value from code
- If Transition semantics used, tells how to compute next “state”
 - To get final value, put in a loop

4/4/24

110

Natural Semantics Example

- $\text{compute_exp}(\text{Var}(v), m) = \text{look_up } v \text{ } m$
- $\text{compute_exp}(\text{Int}(n), _) = \text{Num}(n)$
- ...
- $\text{compute_com}(\text{IfExp}(b, c1, c2), m) =$
if $\text{compute_exp}(b, m) = \text{Bool}(\text{true})$
then $\text{compute_com}(c1, m)$
else $\text{compute_com}(c2, m)$

4/4/24

111

Natural Semantics Example

- $\text{compute_com}(\text{While}(b, c), m) =$
if $\text{compute_exp}(b, m) = \text{Bool}(\text{false})$
then m
else $\text{compute_com}(\text{While}(b, c), \text{compute_com}(c, m))$
- May fail to terminate - exceed stack limits
- Returns no useful information then

4/4/24

112