Review of Prerequisites by Topics

Regular Expression

λ	empty string
a	one a
a+	one or more a's
a*	zero or more a's
a | b	a or b
ab	a followed by b

[a] means a | λ

{a} means a*

Not particularly used, but some concepts (or similar notation) will be used in EBNF syntax description

Context-Free Grammar (CFG)

- For programming language syntax description
  - Example 1

  G1: E -> E + E | E * E | (E) | num | id

  - Example 2

  G2: E -> E + T | T
  T -> T * F | F
  F -> (E) | num | id

  - Example 3

  G3: S -> i(E)S | i(E)SeS | a
  E -> b

Accepting Strings

- Is the following string syntactically correct according to G1?

G1: E -> E + E | E * E | (E) | num | id

a) num * (id + id)

b) num * id (id + id)

Accepting Strings

- Are the following syntactically correct?

According to G1:

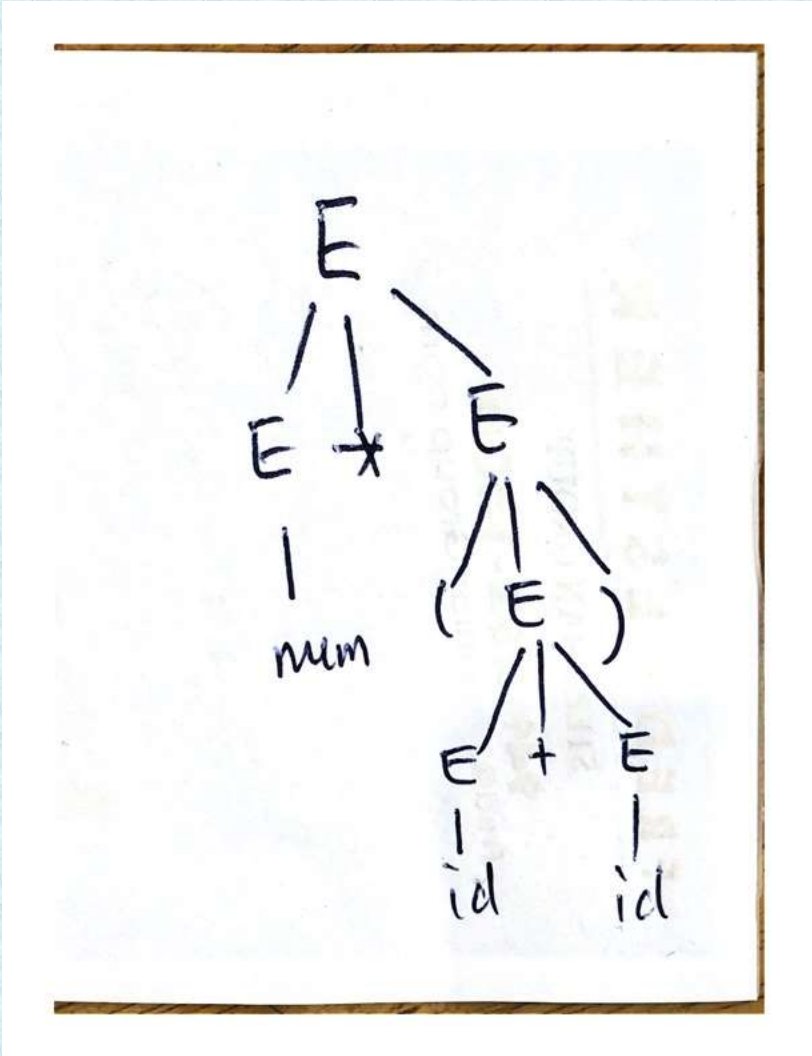
num * (id + id)

accepted: see parse tree

num * id (id + id)

rejected:

failed to create a parse tree



Real-World Example of CFG

Java Statements (simplified version)

Statement:
- Block
- ;
- StatementExpression ;
- if ParExpression Statement [else Statement]
- switch ParExpression { SwitchBlockStatementGroups }
- while ParExpression Statement
- do Statement while ParExpression ;
- for (ForControl) Statement
- break [Identifier] ;
- continue [Identifier] ;
- return [Expression] ;
- throw Expression ;
- try Block (Catches | [Catches] Finally)

Expression:
Expression1 [AssignmentOperator Expression1]

AssignmentOperator:
=
+=
-=
*=
/=
&=
|=
^=
%=
<<=
>>=
>>>=

Expression1:
Literal [Expression1Rest]

Expression1Rest:
{ InfixOp Expression1 }

Literal:
IntegerLiteral
FloatingPointLiteral
CharacterLiteral
StringLiteral
BooleanLiteral
NullLiteral

InfixOp:
||
&&
|
^
&
==
!=
<
>
<=
>=
<<
>>
>>>
+
-
*
/
%

Ambiguous Grammars

- Example 1

G1: E -> E + E | E * E | (E) | num | id

- Example 2

G2: E -> E + T | T

T -> T * F | F

F -> (E) | num | id

- Example 3

G3: S -> i(E) S | i(E)SeS | a

E -> b

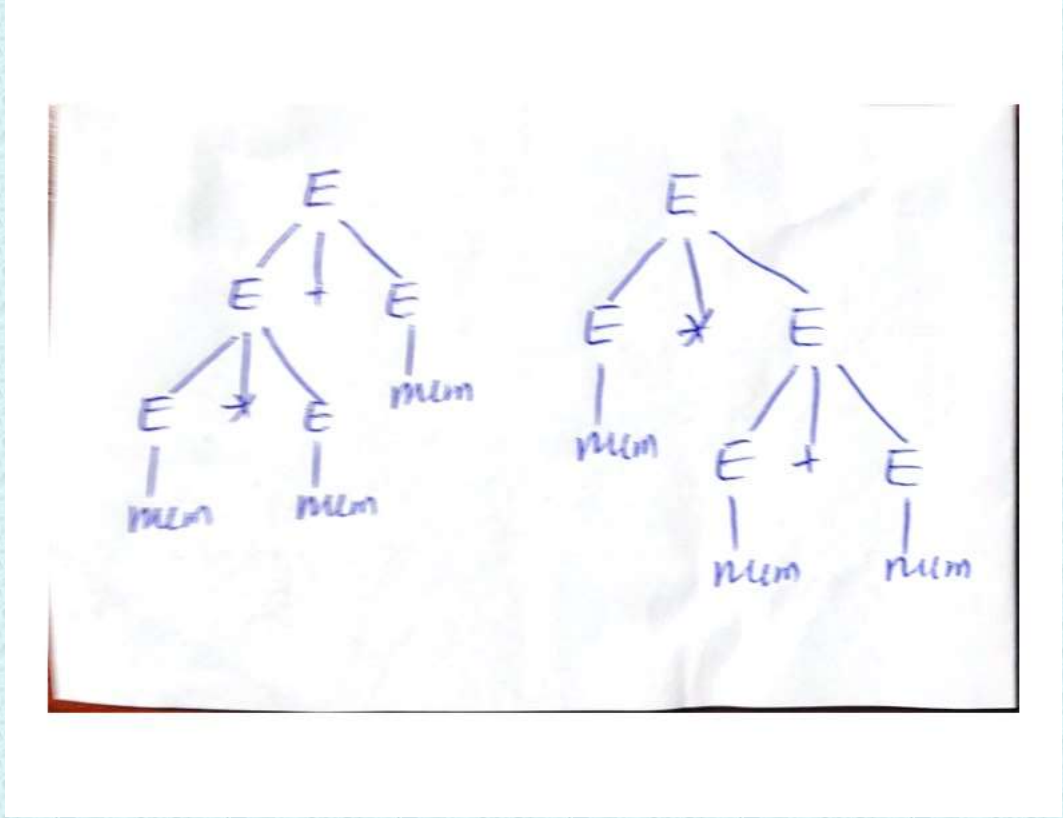
Is G1 ambiguous?
What about G2? G3?

Prove a CFG is ambiguous

G1: E -> E + E | E * E | (E) | num | id

Find a string, num * num + num

Create two distinct parse trees, both accepting the string



What's wrong with ambiguous grammar?

- Multiple meaning, e.g.

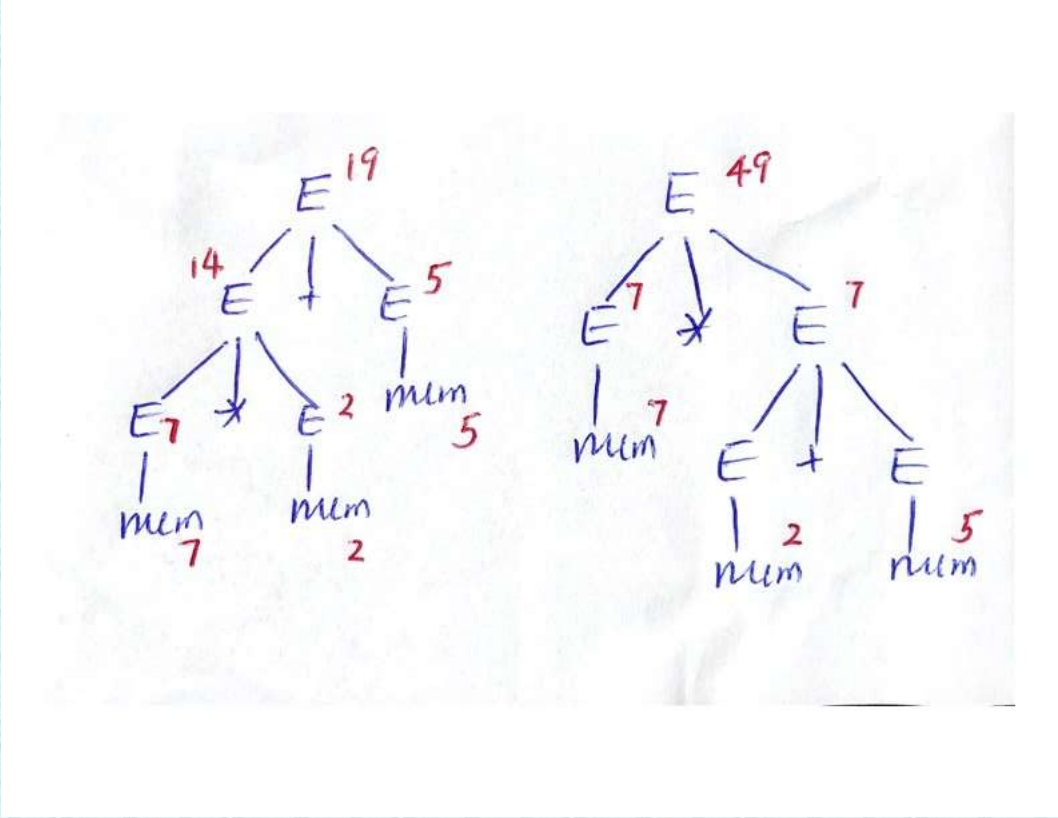
  7 * 2 + 5

Could be interpreted as

(7*2)+5 = 19

Or

7*(2+5) = 49



How to avoid ambiguity?
– Common Strategies

1. Rewrite the grammar into equivalent but unambiguous grammar, e.g.

G1: E -> E + E | E * E | (E) | num | id

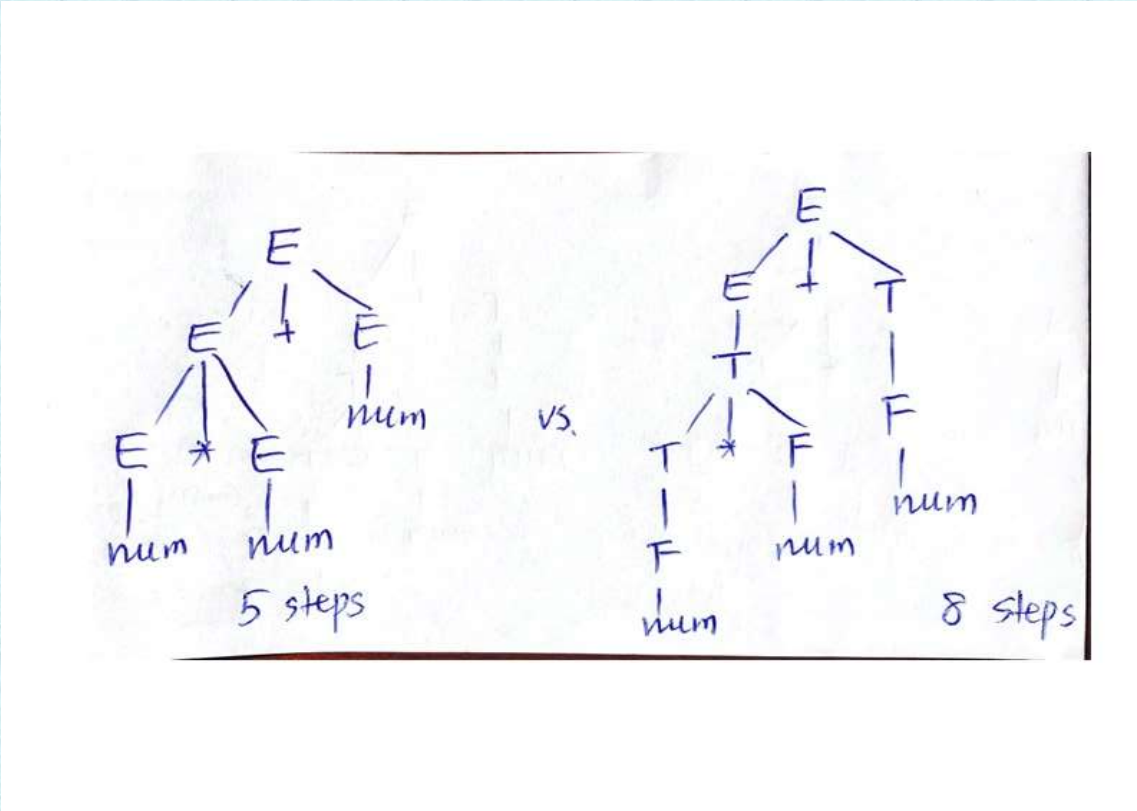
G2: E -> E + T | T
T -> T * F | F
F -> (E) | num | id

G1 and G2 are equivalent (proof omitted), but G2 unambiguous

G2 unambiguous: can you prove it?

G2 vs. G1: Pros and Cons

- G2 unambiguous
  - no confusion, unique meaning for a given input string
- Takes more steps for G2 to accept a string



Common Strategies

2. Enforcing semantic meanings

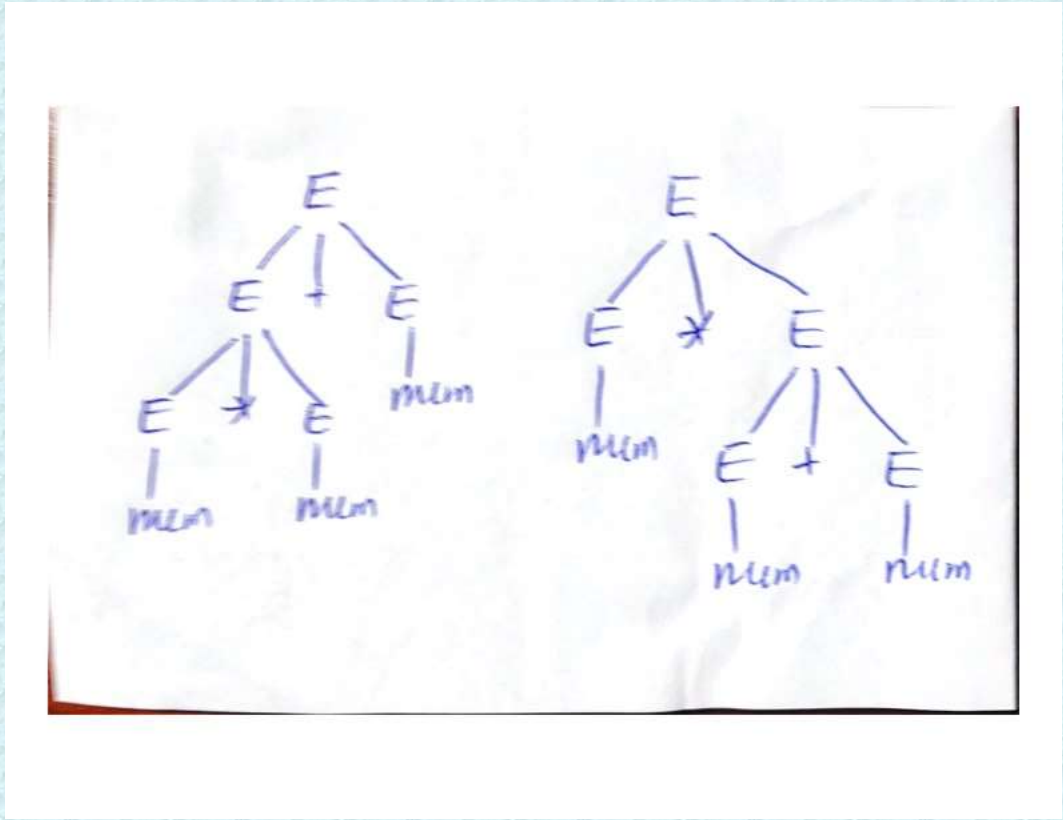
e.g. operator precedence

left side tree:

* is higher than +

right side tree:

+ is higher than *



One of them trees will be eliminated

depending on the definition of operator precedence

Case Study

G3: S -> i(E)S | i(E)SeS | a
E -> b

G3': Stmt -> if (Expr) Stmt | if (Expr) Stmt else Stmt

String: if (a < b)
if (c < d) {
System.out.println("Hi");
}
else {
System.out.println("Bye");
}

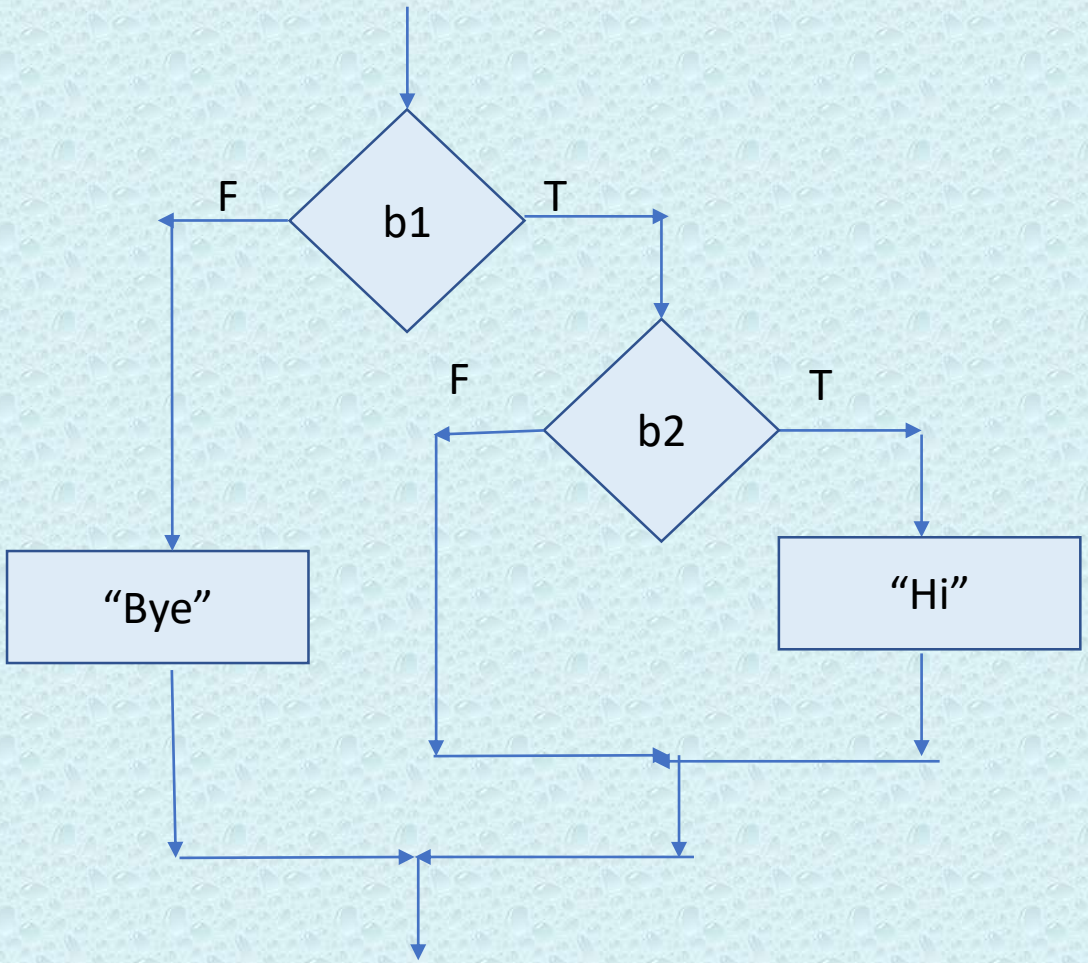
Which one is right interpretation?
i.e. which if should the else match?

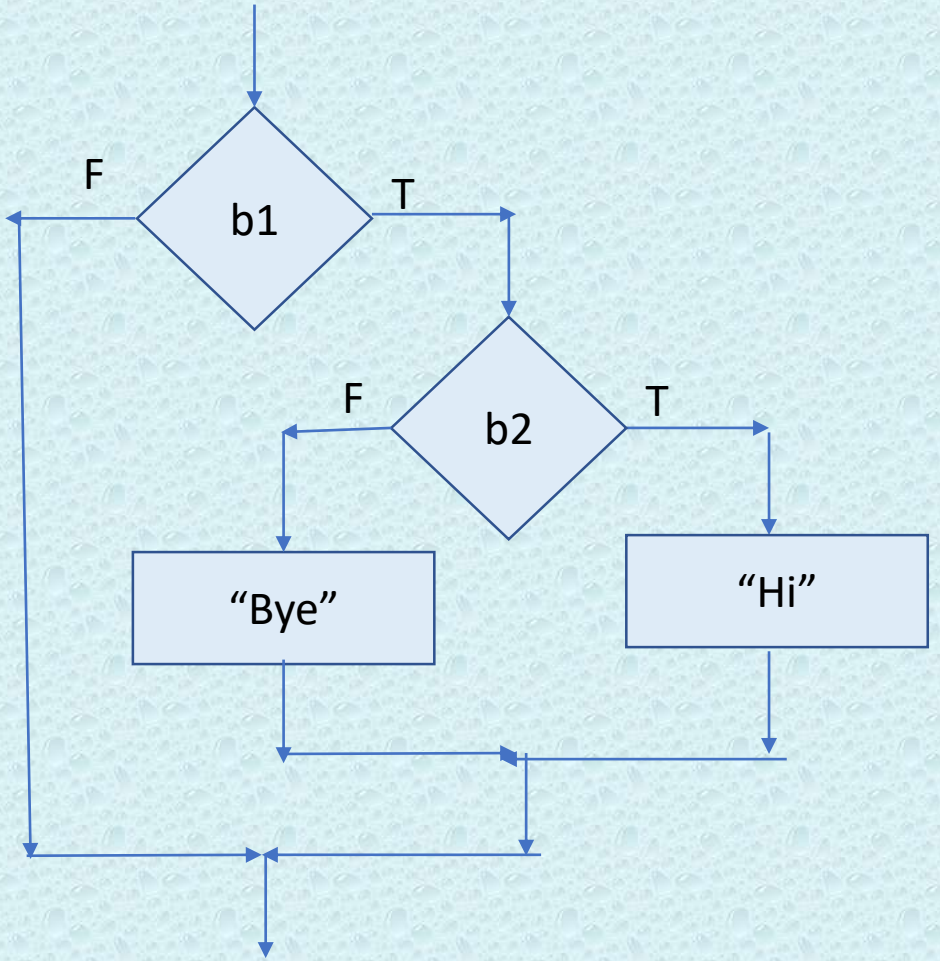
```
if (a < b)
    if (c < d) {
        System.out.println("Hi");
    }
else {
    System.out.println("Bye");
}
```

```
if (a < b)
    if (c < d) {
        System.out.println("Hi");
    }
else {
    System.out.println("Bye");
}
```

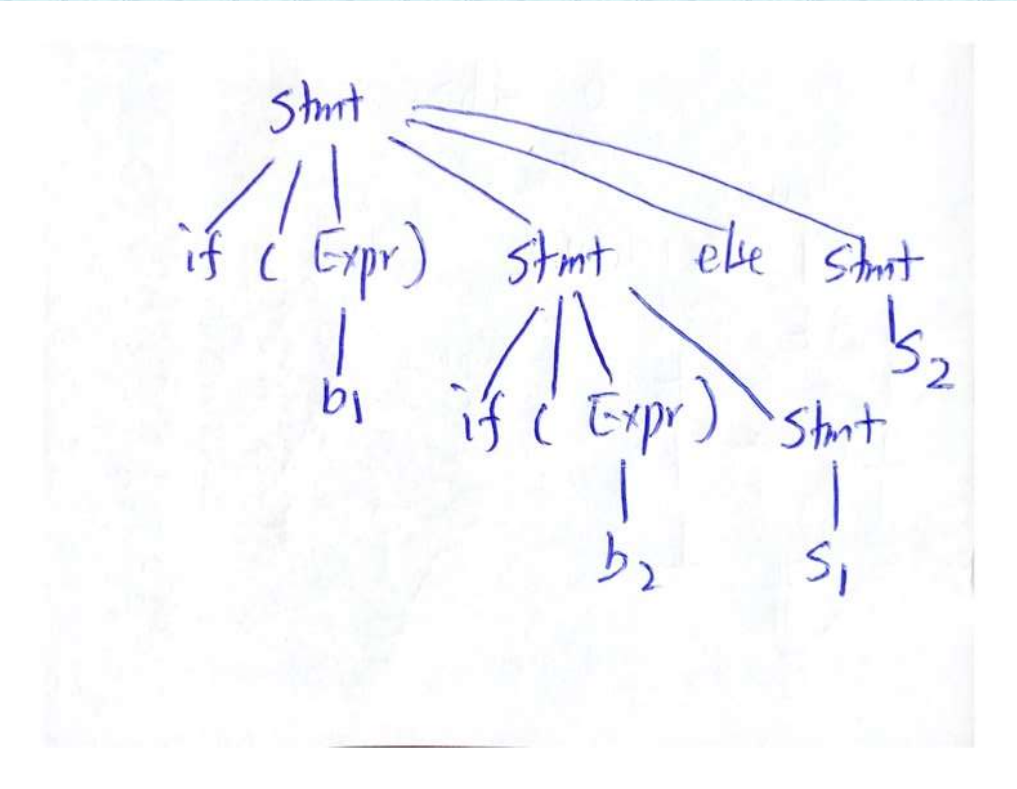
#note: { } unnecessary in this case

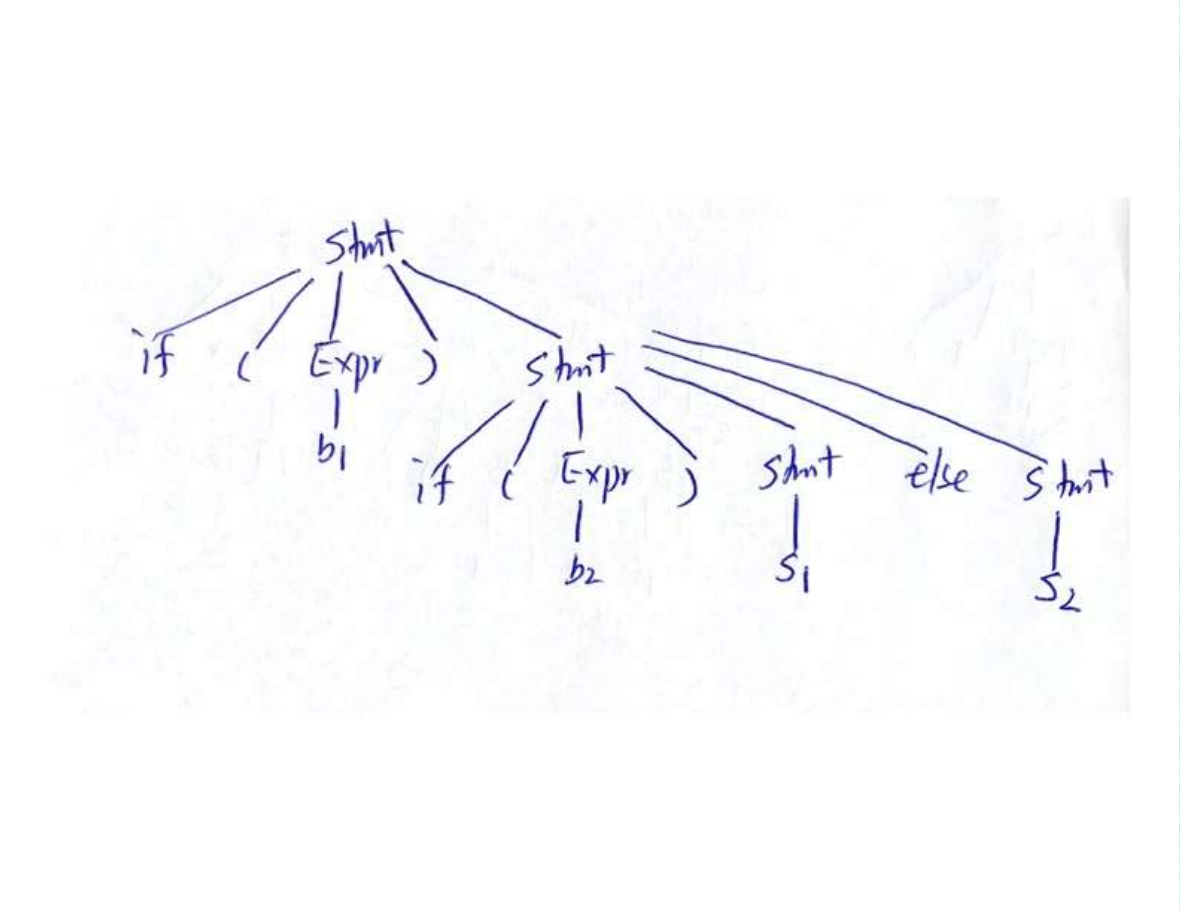
Different interpretation different semantic meaning





Parse/Derivation Tree Representation





Resolving Ambiguity for “dangling else”

(simplify to) if (b1) if (b2) S1 else S2

1. Rewrite to unambiguous CFG

   next slide

2. Semantic rule

   “else” matches the closest “if”

Ambiguous vs. Unambiguous Grammar

- Ambiguous

IfStatement :

if (Expression) Statement

if (Expression) Statement else Statement

- Oracle:

IfThenStatement:
if (Expression) Statement

IfThenElseStatement:
if (Expression) StatementNoShortIf else Statement

IfThenElseStatementNoShortIf:
if (Expression) StatementNoShortIf else StatementNoShortIf

here: StatementNoShortIf refers to any statement without an ifThenStatement in it. The syntax definition for StatementNoShortIf is omitted here.

Unambiguous grammar – another solution

G3’: Stmt -> MS | US

#MS: matched statement

#US: unmatched statement

MS -> if (E) MS else MS | a

US -> if (E) S | if (E) MS else US

E -> b

Practice

- Use unambiguous grammar for if statement to build up a parse tree for the string if (b1) if (b2) a1 else a2

Practice

Study the grammar of your language

- Expressions
  - ambiguous or unambiguous grammar?
  - operator precedence rules?
  - sample accepting strings, non accepting strings
- Statements: assignment, if, while, …
  - ambiguous grammar or not?
  - sample accepting strings, non accepting strings

Pseudo Assembly Language

```
a = b                          #move instruction
a = b op c                     #e.g. a = b * c, a multiplication instruction
        Note: a = b * c + b  not appropriate, no corresponding instruction
goto Label                     #jump instruction
if a relop b goto Label        #conditional branching instruction
                               #e.g. beq, bne, bgt, bge, ble, blt, …
Label:                         #statement can have label
```

Translating to low-level instructions

- Make sure same meaning
- For classwork, do not optimize
- Example

```
total = 1
for (j=0; j < n; j++)
        total = total + j * 5
```

```
        total = 1
        j=0
Loop:
        if j >= n goto Exit
        t = j * 5
        total = total + t
        j = j + 1
        goto Loop
Exit:
```

Practice

```
while (j++ < n) {
    total = total + j * 2
    n = n / 2
}
…print(“Done”)
```

```
switch (n) {
 case 1:  …print(“A”);
 case 2: … print(“B”);
              break;
 case 3: … print(“C”);
              break;
 default:  … print(”D”);
}
```

Practice

Translate each of the following to low-level code and compare the semantic meaning

```
if (a < b)
   if (c < d)
        a = a + b;
   else c = c – d;
```

```
if (a < b) {
    if (c < d)
         a = a + b;
    }
else c = c – d;
```

Activity

- Team to study your language
  - Study syntax definition
  - Starting with simple expressions, assignments, …
  - Making up codes, some with no syntax errors, some with syntax errors