Lecture 11

EECS 483: COMPILER CONSTRUCTION

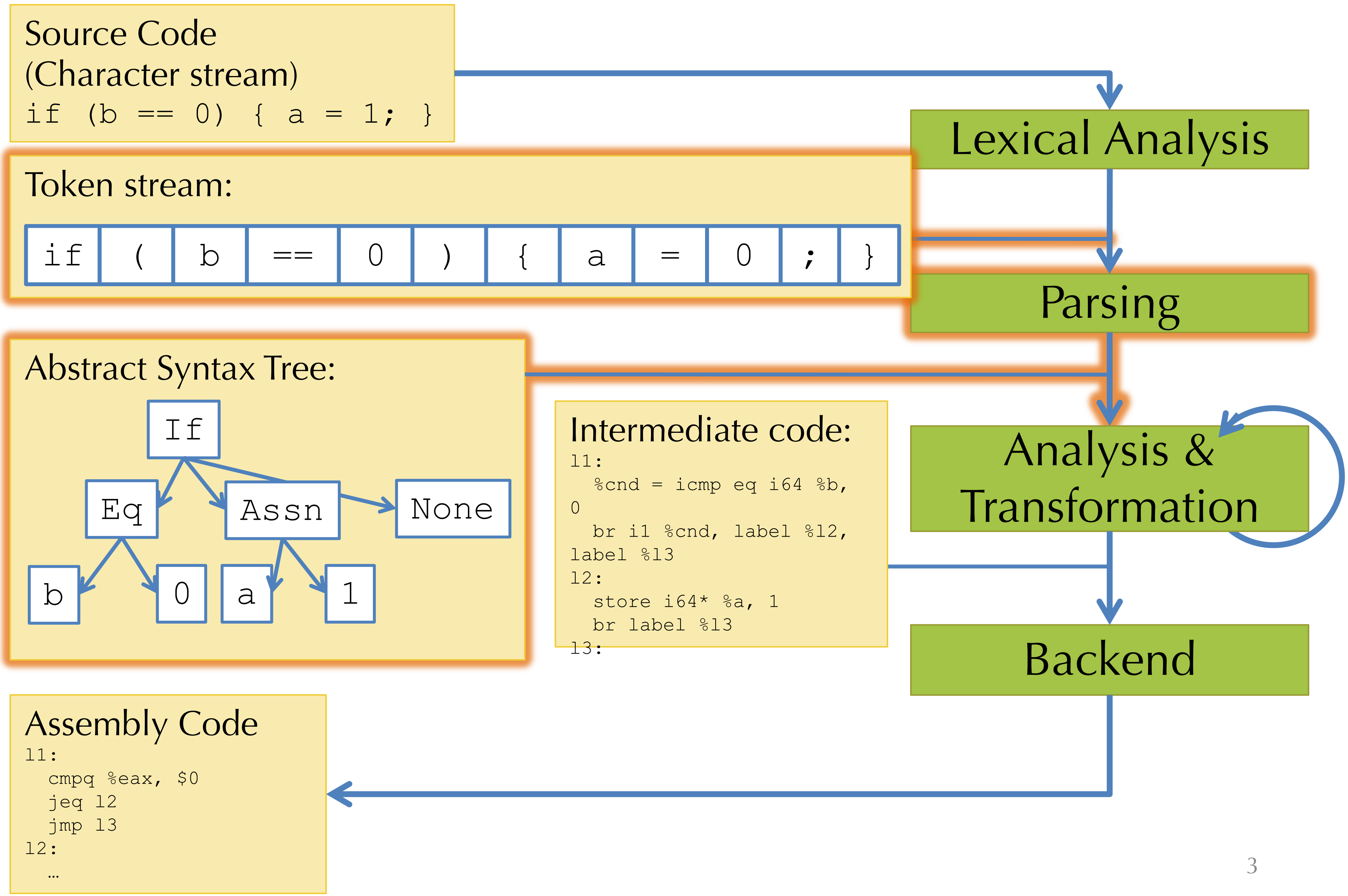
Announcements

- HW3: LLVM lite
  - Available on the course web pages.
  - Due: Tues., February 20th at 11:59pm

it is officially too late to
START EARLY!!

- Midterm: Tuesday, March 12th
  - 7-9pm, DOW 1013 and 1014 (seat assignments will be announced later)
  - One-page, letter-sized, double-sided "cheat sheet" of notes permitted
  - Coverage: interpreters / program transformers / x86 / calling conventions / IRs / LLVM / Lexing / Parsing
  - See examples of previous exams on the web pages
  - March 11 class: review/office hours, no lecture

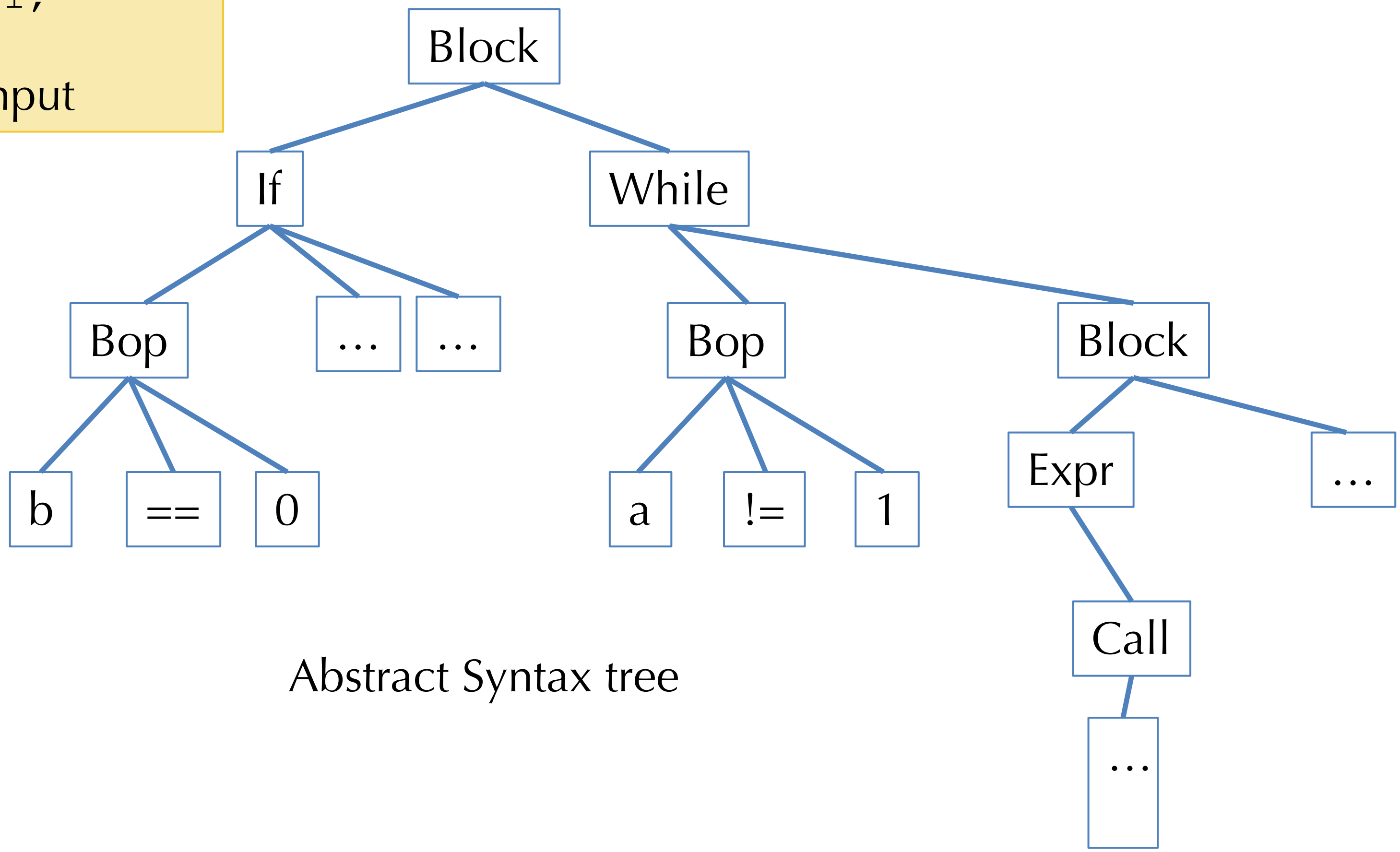
Parsing


Source Code
(Character stream)
if (b == 0) { a = 1; }
Lexical Analysis
Token stream:
if (b == 0) { a = 0 ; }
Parsing
Abstract Syntax Tree:
If
Eq
Assn
None
b
0
a
1
Intermediate code:
l1:
%cnd = icmp eq i64 %b, 0
br i1 %cnd, label %l2, label %l3
l2:
store i64* %a, 1
br label %l3
l3:
Analysis &
Transformation
Backend
Assembly Code
l1:
cmpq %eax, $0
jeq l2
jmp l3
l2:
…

Parsing: Finding Syntactic Structure

```
{
  if (b == 0) a = b;
  while (a != 1) {
    print_int(a);
    a = a - 1;
  }
}
```

Source input



Abstract Syntax tree

CONTEXT FREE GRAMMARS

Context-free Grammars

- Here is a specification of the language of balanced parens:

$$S \longmapsto (S)S$$

$$S \longmapsto \varepsilon$$

Note: Once again we have to take care to distinguish meta-language elements (e.g. "S" and "$\longmapsto$") from object-language elements (e.g. "(").*

- The definition is *recursive* – S mentions itself.

- Idea: "derive" a string in the language by starting with S and rewriting according to the rules:
  - Example: $S \longmapsto (S)S \longmapsto ((S)S)S \longmapsto ((\varepsilon)S)S \longmapsto ((\varepsilon)S)\varepsilon \longmapsto ((\varepsilon)\varepsilon)\varepsilon = (())$

- You can replace the "nonterminal" S by one of its definitions anywhere
- A context-free grammar accepts a string iff there is a derivation from the start symbol

* And, since we're writing this description in English, we are careful distinguish the meta-meta-language (e.g. words) from the meta-language and object-language (e.g. symbols) by using quotes.

CFGs Mathematically

- A Context-free Grammar (CFG) consists of
  - A set of *terminals* (e.g., a lexical token or ε)
  - A set of *nonterminals* (e.g., S and other syntactic variables)
  - A designated nonterminal called the *start symbol*
  - A set of productions: LHS $\longmapsto$ RHS
    - LHS is a nonterminal
    - RHS is a *string* of terminals and nonterminals

- Example: The balanced parentheses language:

$$S \longmapsto (S)S$$

$$S \longmapsto \varepsilon$$

- How many terminals? How many nonterminals? Productions?

Another Example: Sum Grammar

- A grammar that accepts parenthesized sums of numbers:

$$
\begin{array}{lcll}
S & \longmapsto & E + S & \mid \; E \\
E & \longmapsto & \text{number} & \mid \; (\, S \,)
\end{array}
$$

e.g.: (1 + 2 + (3 + 4)) + 5

- Note the vertical bar '|' is shorthand for multiple productions:

$$
\left.
\begin{array}{l}
S \longmapsto E + S \\
S \longmapsto E \\
E \longmapsto \text{number} \\
E \longmapsto (S)
\end{array}
\right\}
$$

4 productions
2 nonterminals: S, E
4 terminals: (,), +, number
Start symbol: S

Derivations in CFGs

- Example: derive (1 + 2 + (3 + 4)) + 5
- $\underline{\mathbf{S}} \longmapsto \underline{\mathbf{E}} + S$

$\longmapsto (\underline{\mathbf{S}}) + S$

$\longmapsto (\underline{\mathbf{E}} + S) + S$

$\longmapsto (1 + \underline{\mathbf{S}}) + S$

$\longmapsto (1 + \underline{\mathbf{E}} + S) + S$

$\longmapsto (1 + 2 + \underline{\mathbf{S}}) + S$

$\longmapsto (1 + 2 + \underline{\mathbf{E}}) + S$

$\longmapsto (1 + 2 + (\underline{\mathbf{S}})) + S$

$\longmapsto (1 + 2 + (\underline{\mathbf{E}} + S)) + S$

$\longmapsto (1 + 2 + (3 + \underline{\mathbf{S}})) + S$

$\longmapsto (1 + 2 + (3 + \underline{\mathbf{E}})) + S$

$\longmapsto (1 + 2 + (3 + 4)) + \underline{\mathbf{S}}$

$\longmapsto (1 + 2 + (3 + 4)) + \underline{\mathbf{E}}$

$\longmapsto (1 + 2 + (3 + 4)) + 5$

$S \longmapsto E + S \mid E$

$E \longmapsto \text{number} \mid (\ S\ )$

For arbitrary strings α, β, γ and production rule $A \longmapsto \beta$
a single step of the derivation is:

$$\alpha A \gamma \longmapsto \alpha\beta\gamma$$

(*substitute* β for an occurrence of A)

In general, there are many possible derivations for a given string

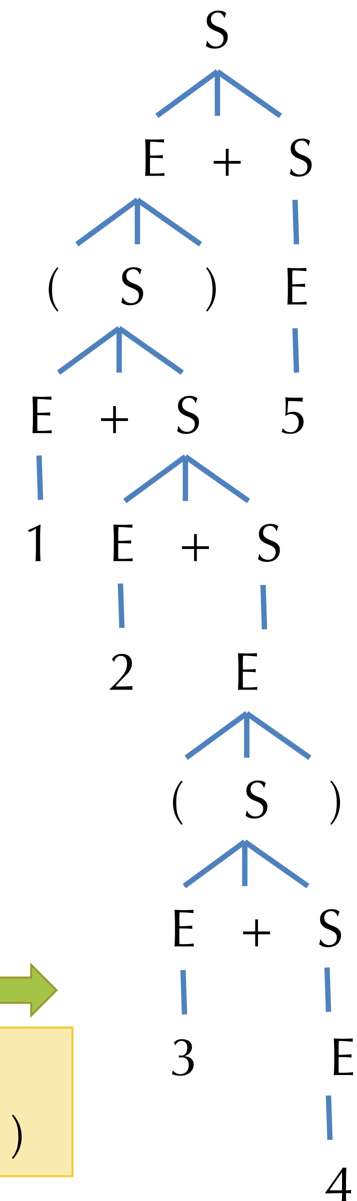
Note: Underline indicates symbol being expanded.

From Derivations to Parse Trees

- Tree representation of the derivation
- Leaves of the tree are terminals
  - In-order traversal yields the input sequence of tokens
- Internal nodes: nonterminals
- No information about the *order* of the derivation steps

- (1 + 2 + (3 + 4)) + 5

$$S \longmapsto E + S \mid E$$
$$E \longmapsto \text{number} \mid (S)$$

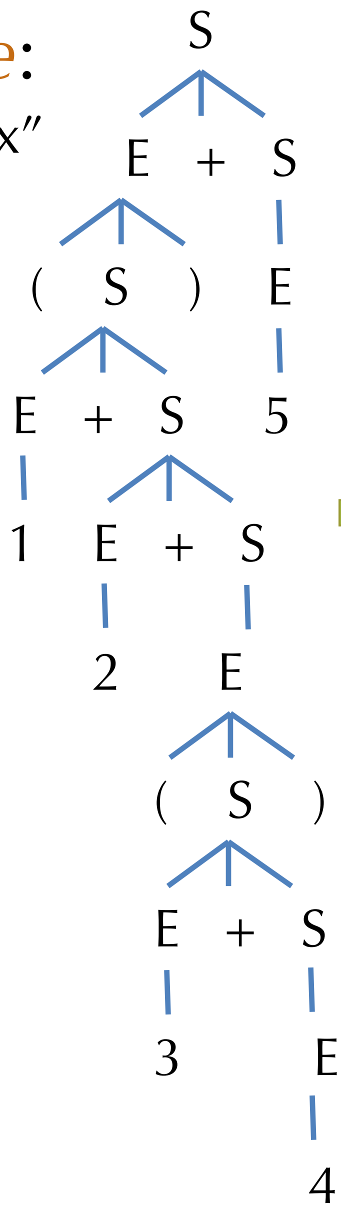


Parse Tree

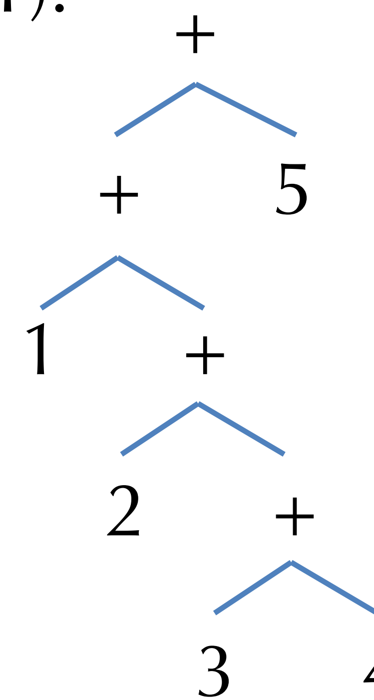
From Parse Trees to Abstract Syntax

- *Parse tree*:

“concrete syntax”



- *Abstract syntax tree* (AST):



- Hides, or *abstracts*, unneeded information.

Derivation Orders

- Productions of the grammar can be applied in any order.
- There are two standard orders:
  - *Leftmost derivation*: Find the left-most nonterminal and apply a production to it.
  - *Rightmost derivation*: Find the right-most nonterminal and apply a production there.

- Idea: These are ***search strategies*** for finding a parse tree
  - Both strategies (and any other) yield the same parse tree!
  - Parse tree doesn't contain the information about what order the productions were applied.

Example: Left- and rightmost derivations

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

- Leftmost derivation:
- $\underline{\mathbf{S}} \longmapsto \underline{\mathbf{E}} + S$
  $\longmapsto (\underline{\mathbf{S}}) + S$
  $\longmapsto (\underline{\mathbf{E}} + S) + S$
  $\longmapsto (1 + \underline{\mathbf{S}}) + S$
  $\longmapsto (1 + \underline{\mathbf{E}} + S) + S$
  $\longmapsto (1 + 2 + \underline{\mathbf{S}}) + S$
  $\longmapsto (1 + 2 + \underline{\mathbf{E}}) + S$
  $\longmapsto (1 + 2 + (\underline{\mathbf{S}})) + S$
  $\longmapsto (1 + 2 + (\underline{\mathbf{E}} + S)) + S$
  $\longmapsto (1 + 2 + (3 + \underline{\mathbf{S}})) + S$
  $\longmapsto (1 + 2 + (3 + \underline{\mathbf{E}})) + S$
  $\longmapsto (1 + 2 + (3 + 4)) + \underline{\mathbf{S}}$
  $\longmapsto (1 + 2 + (3 + 4)) + \underline{\mathbf{E}}$
  $\longmapsto (1 + 2 + (3 + 4)) + 5$

Rightmost derivation:

$\underline{\mathbf{S}} \longmapsto E + \underline{\mathbf{S}}$
$\longmapsto E + \underline{\mathbf{E}}$
$\longmapsto \underline{\mathbf{E}} + 5$
$\longmapsto (\underline{\mathbf{S}}) + 5$
$\longmapsto (E + \underline{\mathbf{S}}) + 5$
$\longmapsto (E + E + \underline{\mathbf{S}}) + 5$
$\longmapsto (E + E + \underline{\mathbf{E}}) + 5$
$\longmapsto (E + E + (\underline{\mathbf{S}})) + 5$
$\longmapsto (E + E + (E + \underline{\mathbf{S}})) + 5$
$\longmapsto (E + E + (E + \underline{\mathbf{E}})) + 5$
$\longmapsto (E + E + (\underline{\mathbf{E}} + 4)) + 5$
$\longmapsto (E + \underline{\mathbf{E}} + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{E}} + 2 + (3 + 4)) + 5$
$\longmapsto (1 + 2 + (3 + 4)) + 5$

Loops and Termination

- Some care is needed when defining CFGs
- Consider:

$$\begin{aligned} S &\longmapsto E \\ E &\longmapsto S \end{aligned}$$

  - This grammar has nonterminal definitions that are "nonproductive". (i.e. they don't mention any terminal symbols)
  - There is no finite derivation starting from S, so the language is empty.

- Consider: $S \longmapsto (\ S\ )$

  - This grammar is productive, but again there is no finite derivation starting from S, so the language is empty

- Easily generalize these examples to a "cycle" of many nonterminals, which can be harder to find in a large grammar

- Upshot: be aware of "vacuously empty" CFG grammars.
  - Every nonterminal should eventually rewrite to an alternative that contains only terminal symbols.

Associativity, ambiguity, and precedence.

GRAMMARS FOR PROGRAMMING LANGUAGES

Associativity

Consider the input: $1 + 2 + 3$

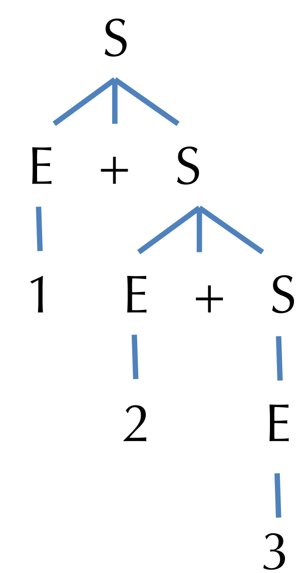
$$S \longmapsto E + S \mid E$$
$$E \longmapsto \text{number} \mid (S)$$

Leftmost derivation:

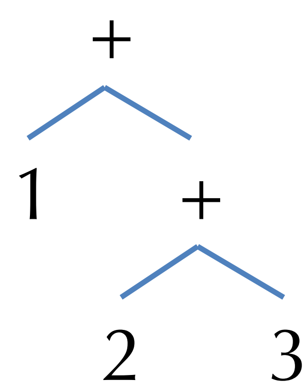
$$\begin{aligned} \underline{\mathbf{S}} &\longmapsto \underline{\mathbf{E}} + S \\ &\longmapsto 1 + \underline{\mathbf{S}} \\ &\longmapsto 1 + \underline{\mathbf{E}} + S \\ &\longmapsto 1 + 2 + \underline{\mathbf{S}} \\ &\longmapsto 1 + 2 + \underline{\mathbf{E}} \\ &\longmapsto 1 + 2 + 3 \end{aligned}$$

Rightmost derivation:

$$\begin{aligned} \underline{\mathbf{S}} &\longmapsto E + \underline{\mathbf{S}} \\ &\longmapsto E + E + \underline{\mathbf{S}} \\ &\longmapsto E + E + \underline{\mathbf{E}} \\ &\longmapsto E + \underline{\mathbf{E}} + 3 \\ &\longmapsto \underline{\mathbf{E}} + 2 + 3 \\ &\longmapsto 1 + 2 + 3 \end{aligned}$$



Parse Tree



AST

Associativity

- This grammar makes '+' *right associative*…
  - i.e., the abstract syntax tree is the same for both 1 + 2 + 3 and 1 + (2 + 3)
- Note that the grammar is *right recursive*…

$$S \longmapsto E + S \mid E$$
$$E \longmapsto \text{number} \mid (S)$$

S refers to itself on the right of +

- How would you make '+' left associative?
- What are the trees for "1 + 2 + 3"?

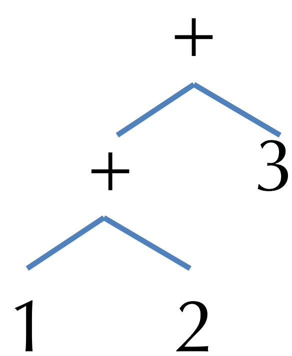
Ambiguity

- Consider this grammar:

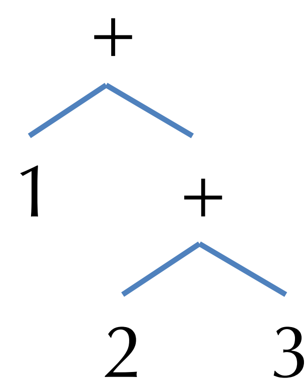
$$S \mapsto S + S \mid (\ S\ ) \mid \text{number}$$

- Claim: it accepts the *same* set of strings as the previous one.
- What's the difference?
- Consider these *two* leftmost derivations:
  - $\underline{\mathbf{S}} \mapsto \underline{\mathbf{S}} + S \mapsto 1 + \underline{\mathbf{S}} \mapsto 1 + \underline{\mathbf{S}} + S \mapsto 1 + 2 + \underline{\mathbf{S}} \mapsto 1 + 2 + 3$
  - $\underline{\mathbf{S}} \mapsto \underline{\mathbf{S}} + S \mapsto \underline{\mathbf{S}} + S + S \mapsto 1 + \underline{\mathbf{S}} + S \mapsto 1 + 2 + \underline{\mathbf{S}} \mapsto 1 + 2 + 3$

- One derivation gives left associativity, the other gives right associativity to '+'
  - Which is which?



AST 1



AST 2

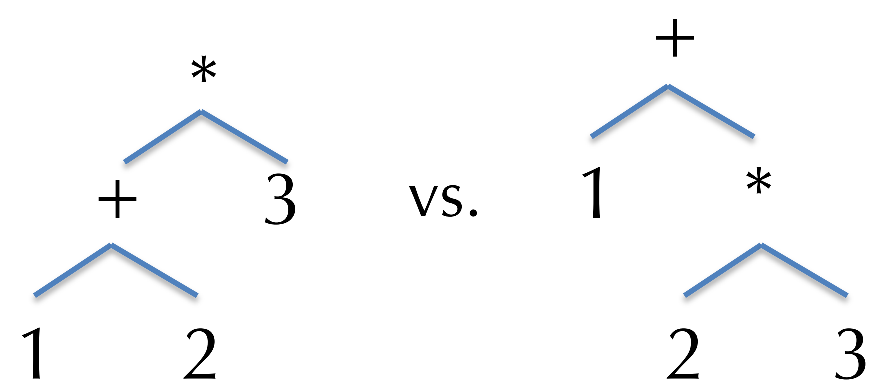
Why do we care about ambiguity?

- The '+' operation is associative, so it doesn't matter which tree we pick. Mathematically, $x + (y + z) = (x + y) + z$
  - But, some operations aren't associative. Examples?
  - Some operations are only left (or right) associative. Examples?

- Moreover, if there are multiple operations, ambiguity in the grammar leads to ambiguity in their *precedence*
- Consider:

$$S \longmapsto S + S \;|\; S * S \;|\; (S) \;|\; \text{number}$$

- Input: 1 + 2 * 3
  - One parse = (1 + 2) * 3 = 9
  - The other = 1 + (2 * 3) = 7



Eliminating Ambiguity

- We can often eliminate ambiguity by adding nonterminals and allowing recursion only on the left (or right) .
- Higher-precedence operators go *farther* from the start symbol.
- Example:

$$S \mapsto S + S \mid S * S \mid (S) \mid \text{number}$$

- To disambiguate:
  - Decide (following math) to make '*' higher precedence than '+'
  - Make '+' left associative
  - Make '*' right associative
- Note:
  - S_2 corresponds to 'atomic' expressions

$$\begin{aligned} S_0 &\mapsto S_0 + S_1 \mid S_1 \\ S_1 &\mapsto S_2 * S_1 \mid S_2 \\ S_2 &\mapsto \text{number} \mid (S_0) \end{aligned}$$

Context Free Grammars: Summary

- Context-free grammars allow concise specifications of programming languages.
  - An unambiguous CFG specifies how to parse: convert a token stream to a (parse tree)
  - Ambiguity can (often) be removed by encoding precedence and associativity in the grammar.

- Even with an unambiguous CFG, there may be more than one derivation
  - Though all derivations correspond to the same abstract syntax tree.

- Still to come: finding a derivation
  - But first: menhir

parser.mly, lexer.mll, range.ml, ast.ml, main.ml

DEMO: BOOLEAN LOGIC