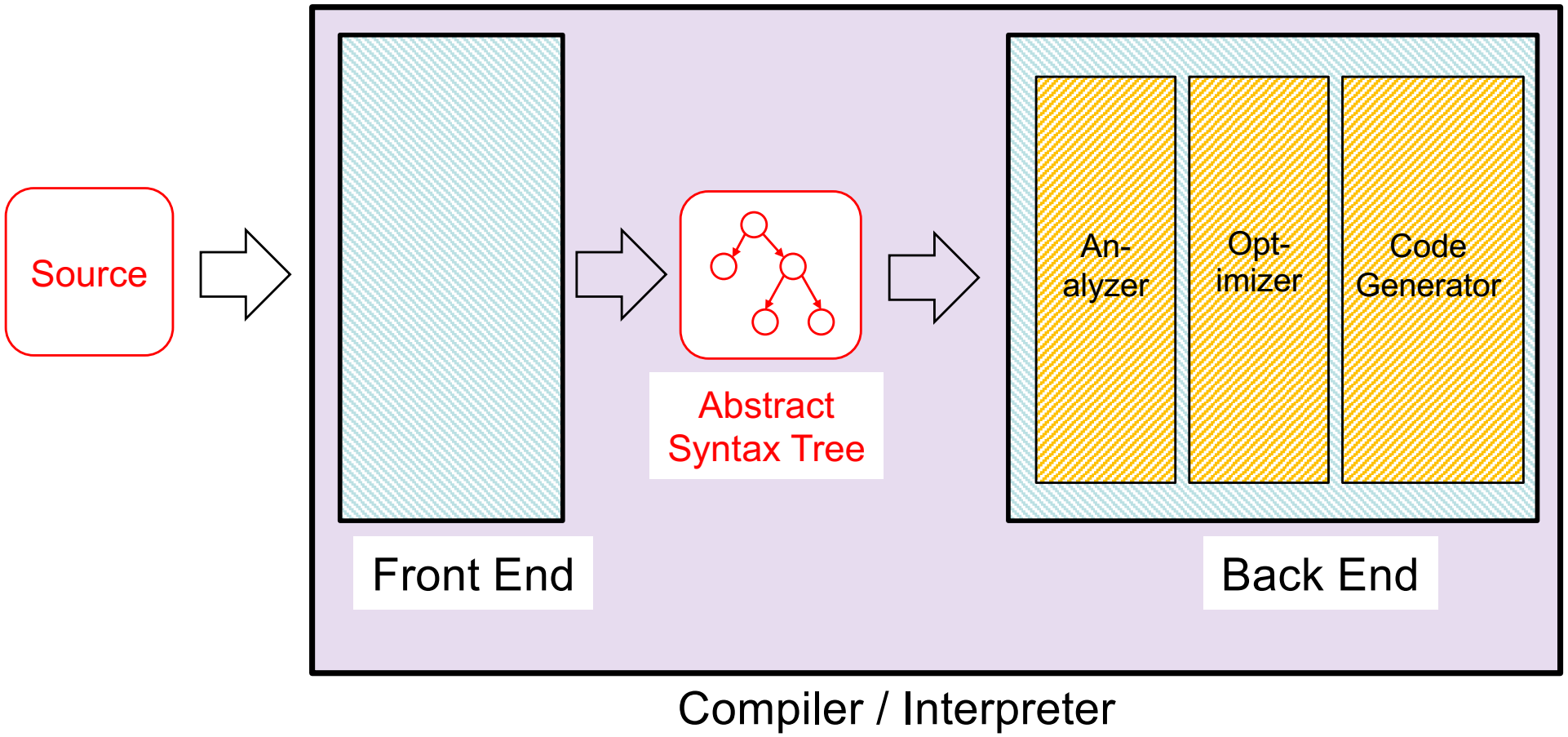
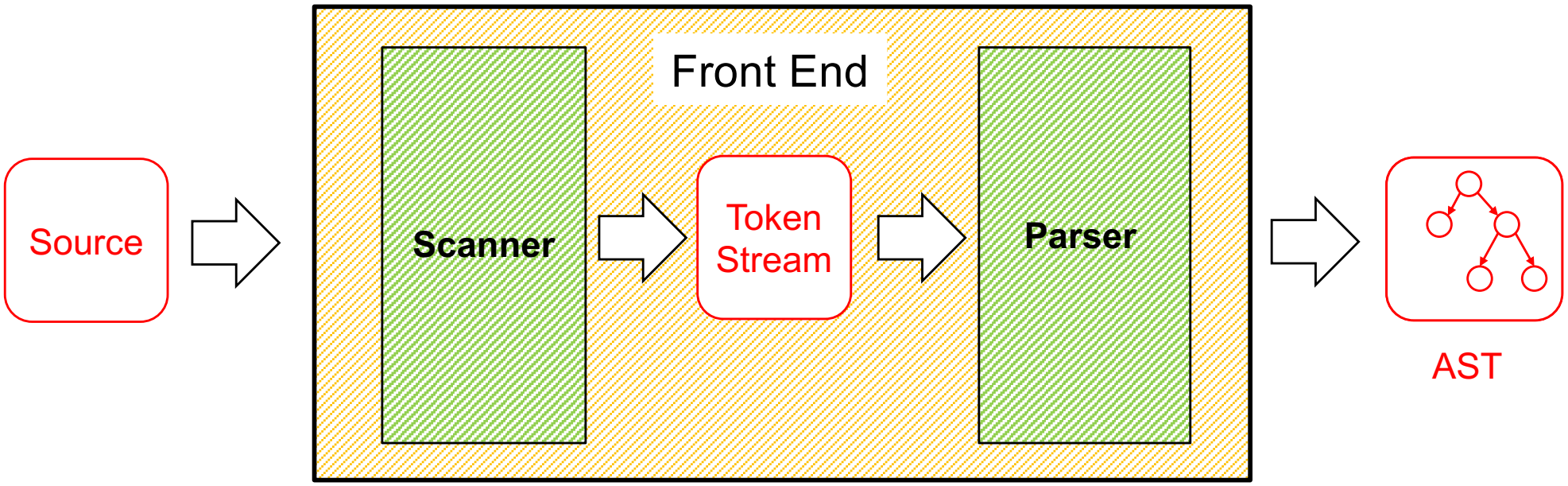
CMSC 330: Organization of Programming Languages

Context Free Grammars

Architecture of Compilers, Interpreters



Front End – Scanner and Parser



- Scanner / lexer converts program source into tokens (keywords, variable names, operators, numbers, etc.) using regular expressions
- Parser converts tokens into an AST (abstract syntax tree) using context free grammars

Context-Free Grammar (CFG)

- A way of describing sets of strings (= languages)
  - The notation L(G) denotes the language of strings defined by grammar G

- Example grammar G is $S \rightarrow 0S \mid 1S \mid \varepsilon$

  which says that string s' $\in$ L(G) iff
  - s' = ε, or $\exists$s $\in$ L(G) such that s' = 0s, or s' = 1s

- Grammar is same as regular expression (0|1)*
  - Generates / accepts the same set of strings

CFGs Are Expressive

- CFGs subsume REs, DFAs, NFAs
  - There is a CFG that generates any regular language
  - But: REs are often better notation for those languages

- And CFGs can define languages regexps cannot
  - S → (S) | ε // represents balanced pairs of ( )'s

- As a result, CFGs often used as the basis of parsers for programming languages

Parsing with CFGs

- CFGs formally define languages, but they do not define an *algorithm* for accepting strings
- Several styles of algorithm; each works only for less expressive forms of CFG
  - LL(k) parsing ← We will discuss this next lecture
  - LR(k) parsing
  - LALR(k) parsing
  - SLR(k) parsing
- Tools exist for building parsers from grammars
  - JavaCC, Yacc, etc.

Formal Definition: Context-Free Grammar

- A CFG G is a 4-tuple (Σ, N, P, S)
  - Σ – alphabet (finite set of symbols, or terminals)
    - Often written in lowercase
  - N – a finite, nonempty set of nonterminal symbols
    - Often written in UPPERCASE
    - It must be that N ∩ Σ = ∅
  - P – a set of productions of the form $N \rightarrow (\Sigma|N)^*$
    - Informally: the nonterminal can be replaced by the string of zero or more terminals / nonterminals to the right of the →
    - Can think of productions as rewriting rules (more later)
  - S ∈ N – the start symbol

Notational Shortcuts

```
S → aBc  // S is start symbol
A → aA
  |  b              // A → b
  |                 // A → ε
```

- A production is of the form
  - left-hand side (LHS) → right hand side (RHS)
- If not specified
  - Assume LHS of first production is the start symbol
- Productions with the same LHS
  - Are usually combined with |
- If a production has an empty RHS
  - It means the RHS is ε

Backus-Naur Form

- Context-free grammar production rules are also called Backus-Naur Form or BNF
  - Designed by John Backus and Peter Naur
    - Chair and Secretary of the Algol committee in the early 1960s. Used this notation to describe Algol in 1962

- A production A → B c D is written in BNF as <A> ::= <B> c <D>
  - Non-terminals written with angle brackets and uses ::= instead of →
  - Often see hybrids that use ::= instead of → but drop the angle brackets on non-terminals

Generating Strings

- We can think of a grammar as generating strings by rewriting
- Example grammar G

  $S \rightarrow 0S \mid 1S \mid \varepsilon$
- Generate string 011 from G as follows:

$S \Rightarrow 0S$ // using $S \rightarrow 0S$

$\Rightarrow 01S$ // using $S \rightarrow 1S$

$\Rightarrow 011S$ // using $S \rightarrow 1S$

$\Rightarrow 011$ // using $S \rightarrow \varepsilon$

Accepting Strings (Informally)

- Checking if s ∈ L(G) is called acceptance
  - Algorithm: Find a rewriting starting from G's start symbol that yields s
  - A rewriting is some sequence of productions (rewrites) applied starting at the start symbol
    - 011 ∈ L(G) according to the previous rewriting

- Terminology
  - Such a sequence of rewrites is a derivation or parse
  - Discovering the derivation is called parsing