Formal Syntax of Programming Languages

Mooly Sagiv

Benefits of formal definitions

- Intellectual
- Better understanding
- Formal proofs
- Mechanical checks by computer
- Tool generations

What is a good formal definition?

- Natural
- Concise
- Easy to understand
- Permits effective mechanical reasoning

Syntax vs. Semantics

- The pattern of formation of sentences or phrases in a language
- Examples
  - Regular expressions
  - Context free grammars

- The study or science of meaning in language
- Examples
  - Interpreter
  - Compiler
  - Better mechanisms will be given in the course

Benefits of formal syntax for programming language

- Intellectual
- Simplicity
- Better understanding
  - Interaction between different parts
- Abstraction
  - Portability
- Tool generations
  - Parser

Recursive Syntax Definitions

- The syntax of programming languages is naturally defined recursively
- Valid program are represented as syntax trees
- Expressions
  - Every variable is an expression
  - If E1 and E2 are expressions and op is a binary operation then so is 'E_1 op E_2' is an expression
- Statements
  - If V is a variable and E is an expression then 'V := E' is a statement
  - If S_1 and S_2 are statements and E is an expression then
    - 'S_1 ; S_2' is a statement
    - 'if (E) S_1 else S_2' is a statement

C Example

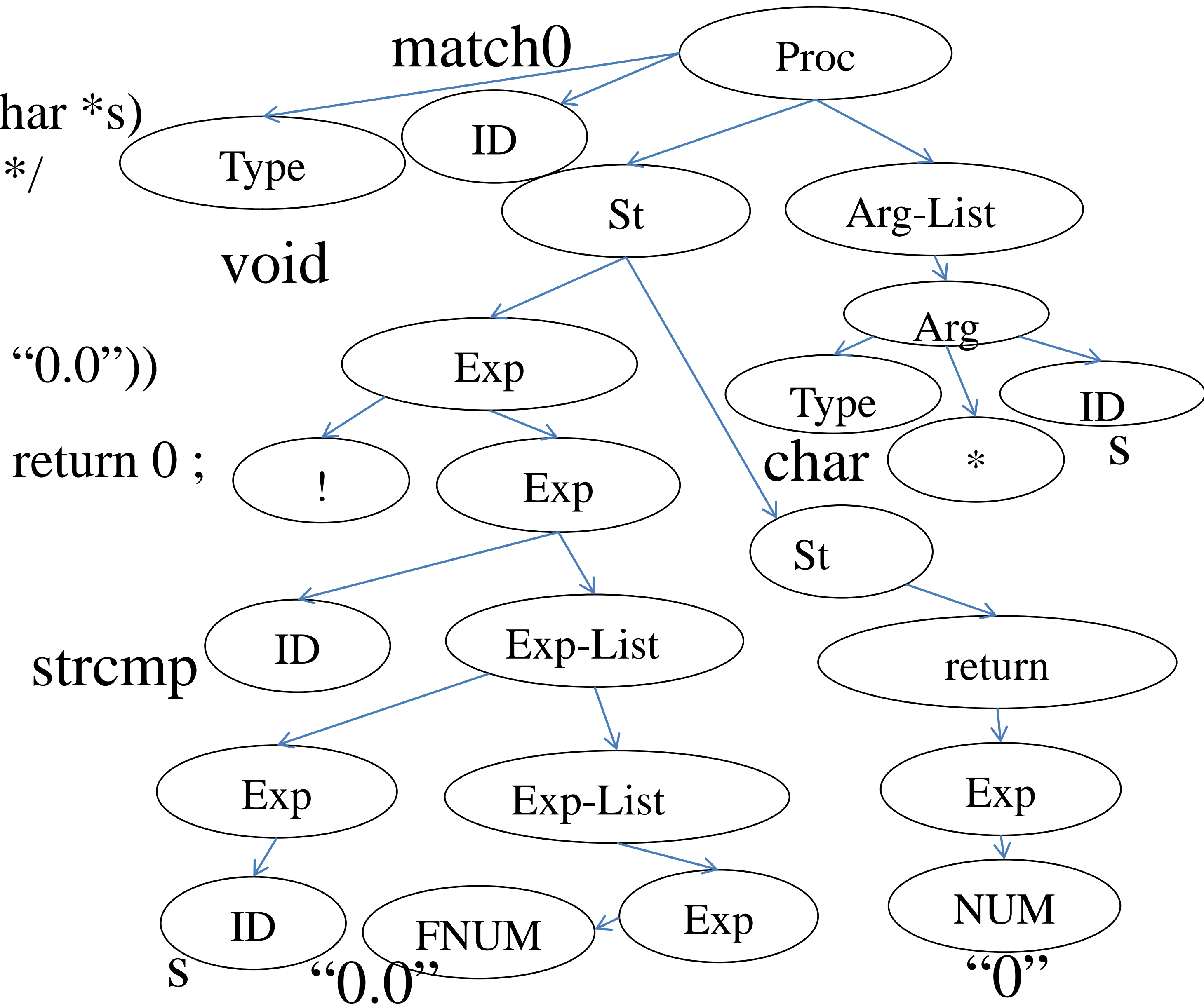
```
void match0(char *s)
 /* find a zero */

{

  if (!strcmp(s, “0.0”))

                    return 0 ;

}
```



Context Free Grammars

- Non-terminals
  - Start non-terminal
- Terminals (tokens)
- Context Free Rules

  <Non-Terminal> $\rightarrow$ Symbol Symbol … Symbol

Example Context Free Grammar

1 $S \rightarrow S \; ; \; S$

2 $S \rightarrow \text{id} := E$

3 $S \rightarrow \text{print} \; (L)$

4 $E \rightarrow \text{id}$

5 $E \rightarrow \text{num}$

6 $E \rightarrow E + E$

7 $E \rightarrow (S, E)$

8 $L \rightarrow E$

9 $L \rightarrow L, E$

Derivations

- Show that a sentence is in the grammar (valid program)
  - Start with the start symbol
  - Repeatedly replace one of the non-terminals by a right-hand side of a production
  - Stop when the sentence contains terminals only
- Rightmost derivation
- Leftmost derivation

Parse Trees

- The trace of a derivation
- Every internal node is labeled by a non-terminal
- Each symbol is connected to the deriving non-terminal

Example Parse Tree

S

S ; S

S ; id := E

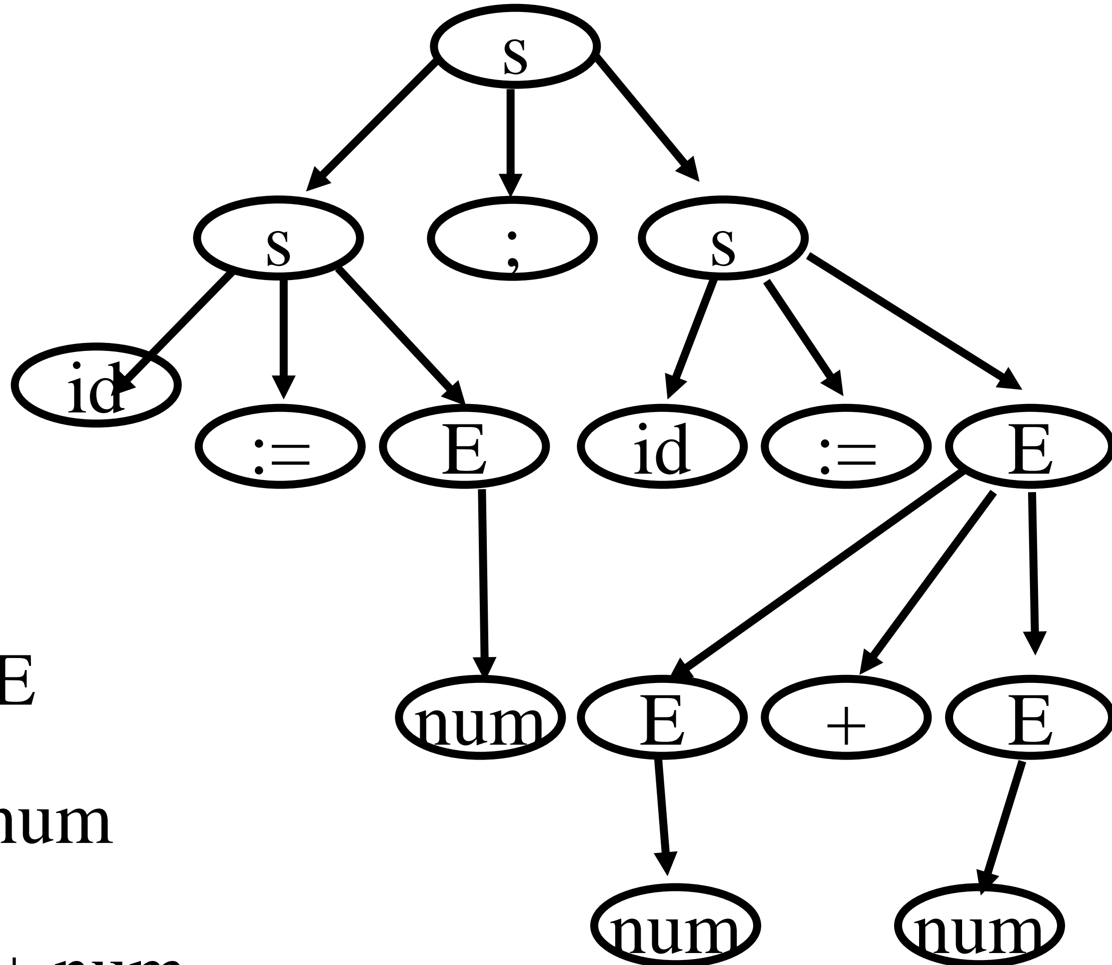
id := E ; id := E

id := num ; id := E

id := num ; id := E + E

id := num ; id := E + num

id := num ; id :=num + num



Ambiguous Grammars

- Two leftmost derivations
- Two rightmost derivations
- Two parse trees

A Grammar for Arithmetic Expressions

1 $E \rightarrow E + E$

2 $E \rightarrow E * E$

3 $E \rightarrow id$

4 $E \rightarrow (E)$

Drawbacks of Ambiguous Grammars

- Ambiguous semantics
- Parsing complexity
- May affect other phases

Non Ambiguous Grammar for Arithmetic Expressions

Ambiguous grammar

1 $E \rightarrow E + E$

2 $E \rightarrow E * E$

3 $E \rightarrow id$

4 $E \rightarrow (E)$

1 $E \rightarrow E + T$

2 $E \rightarrow T$

3 $T \rightarrow T * F$

4 $T \rightarrow F$

5 $F \rightarrow id$

6 $F \rightarrow (E)$

Non Ambiguous Grammars for Arithmetic Expressions

Ambiguous grammar

Ambiguous grammar		
1 $E \rightarrow E + E$	1 $E \rightarrow E + T$	1 $E \rightarrow E * T$
2 $E \rightarrow E * E$	2 $E \rightarrow T$	2 $E \rightarrow T$
3 $E \rightarrow id$	3 $T \rightarrow T * F$	3 $T \rightarrow F + T$
4 $E \rightarrow (E)$	4 $T \rightarrow F$	4 $T \rightarrow F$
	5 $F \rightarrow id$	5 $F \rightarrow id$
	6 $F \rightarrow (E)$	6 $F \rightarrow (E)$

Parser Generators

- Input: A context free grammar
- Output: A parser for this grammar
  - Reports syntax error
  - Generates syntax tree

Abstract Syntax

- An ambiguous grammar used to concisely define the hierarchy of sentences
- Define the structure of a tree
  Exp ::= Exp BinOp Exp | ID | Const

Context Sensitive Features

- Some PL features cannot be defined using context free grammars
  - Identifiers
  - Types
- Some are awkward to define

Summary

- Context free grammars are very useful
- Support in some PLs (Prolog)
- Good automatic tools (yacc, bison, Cup)
- Used in many contexts