Syntax and Grammars

1

The Study of Languages

- Syntax: the *form* and *compositional structure* of the language
  - *Revolutionary new ideas appear rarely.*
  - *Colorless green ideas sleep furiously.*
- Semantics: the *meaning* and *interpretation* of the language
  - *Time flies like an arrow.*
  - *Fruit flies like a banana.*
- Pragmatics: the *use* of the language

2

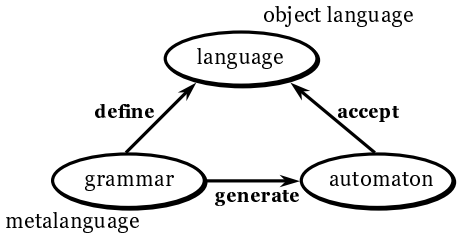
Metalanguages

- *Metalanguage*: a language used to *define* another language

  We will use various *formal languages* to define the *syntax* and *semantics* of programming languages

3

Formal Languages



4

Languages

- *Alphabet* - any *finite* set of *symbols*
  - {0, 1}: *binary alphabet*
- *String* - a *finite* sequence of symbols from the alphabet
  - 1011: *a string of length 4*
  - ε : *the empty string*
- *Language* - *any* set of strings on an alphabet
  - {00, 01, 10, 11}: *the set of strings of length 2*
  - ∅ : *the empty set*

5

Grammars

- A *grammar* is a 4-tuple <*T*, *N*, *S*, *P*>
- *T* is the set of *terminal symbols* or the *alphabet*
- *N* is the set of *nonterminal symbols* denoting *sets of strings*
- *S* is a *distinguished nonterminal symbol* denoting the *language*, called the *start symbol*
- *P* is the set of *productions* specifying *how to form strings* using symbols in *T* and *N*

6

Derivations

- A *production* has the form $\alpha \rightarrow \beta$ where α and β are strings of terminals and nonterminals ($\alpha \neq \varepsilon$)
- We say $\gamma\ \alpha\ \delta \Rightarrow \gamma\ \beta\ \delta$ (derives in *one* step) whenever $\alpha \rightarrow \beta$ is a production
- We use $\alpha \Rightarrow^* \beta$ to denote deriving in *zero* or *more* steps
- A string ω of terminals is in the language, called a *sentence*, if $S \Rightarrow^* \omega$

7

An Example

- Productions:
  expr → *expr op expr*
  expr → **(**' *expr* '**)**'
  expr → '**-**' *expr*
  expr → **id**
  op → '+' | '**-**' | '*' | '/'
- Terminals: **id**, '+', '**-**', '*', '/', '**(**', '**)**'
- Nonterminals: *expr*, *op*
- The start symbol: *expr*

8

An Example

expr → *expr op expr*
→ **id** *op expr*
→ **id** '+' *expr*
→ **id** '+' '**(**' *expr* '**)**'
→ **id** '+' '**(**' *expr op expr* '**)**'
→ **id** '+' '**(**' **id** *op expr* '**)**'
→ **id** '+' '**(**' **id** '*' *expr* '**)**'
→ **id** '+' '**(**' **id** '*' **id** '**)**'

9

Chomsky Hierarchy

0	unrestricted	Turing machines	attribute grammars	static semantics
1	context sensitive	linear bounded automata		
2	context free	pushdown automata	Backus-Naur forms	syntax
3	regular	finite automata	regular expressions	lexicon

10

Regular Grammars

- A grammar $G = <T, N, S, P>$ is *right-linear* if all productions are of the form
  $A \rightarrow x\,B$ or $A \rightarrow x$
  where $A, B \in N$, and $x \in T^*$
- A grammar is *left-linear* if all productions are of the form
  $A \rightarrow B\,x$ or $A \rightarrow x$
- A *regular grammar* is one that is either *right-linear* or *left-linear*

11

Regular Expressions

- ε is a RE denoting $\{\varepsilon\}$
- If $a \in$ alphabet, then a is a RE denoting $\{a\}$
- Suppose r and s are RE denoting L(r) and L(s)
  - (r) | (s) is a RE denoting L(r) ∪ L(s)
  - (r) (s) is a RE denoting L(r)L(s)
  - (r)* is a RE denoting (L(r))*
  - (r) is a RE denoting L(r)

12

Examples

- $a|b$ $\{a, b\}$
- $(a|b)(a|b)$ $\{aa, ab, ba, bb\}$
- a^* $\{\varepsilon, a, aa, aaa, \ldots\}$
- $(a|b)^*$ the set of all strings of a's and b's
- $a|a^*b$ the set containing the string a and all strings consisting of zero or more a's followed by a b

13

Notational Shorthand

- One or more instances
  $(r)^+$ denoting $(L(r))^+$
  $r^* = r^+ \mid \varepsilon$
  $r^+ = r\, r^*$
- Zero or one instance
  r? = r | ε
- Character classes
  [abc] = a | b | c
  [a-z] = a | b | ... | z

14

Examples

- letter [A-Za-z]
- digit [0-9]
- id {letter}({letter}|{digit})*
- number
  {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?

15

Context-Free Grammars

- A grammar $G = <T, N, S, P>$ is *context-free* if all productions are of the form
  $A \rightarrow x$
  where $A \in N$, and $x \in (T \cup N)^*$

16

Backus-Naur Forms (BNF)

<expr> ::= <expr> <op> <expr>
<expr> ::= '**(**' <expr> '**)**'
<expr> ::= '**-**' <expr>
<expr> ::= '**id**'
<op> ::= '+' | '**-**' | '*' | '/'

17

Extended BNF (EBNF)

- Expressions can be enclosed within parentheses, namely,
  $E = (\, E \,)$
- $\{x\}$ denotes an arbitrarily long sequence of x, namely,
  $E = E\, x \mid \varepsilon$
- $[x]$ denotes x or nothing, namely,
  $E = x \mid \varepsilon$

18

EBNF

<EBNF> ::= { <production> }
<production> ::= <nonterm> '::=' <expr>
<expr> ::= <term> { '|' <term> }
<term> ::= <factor> { <factor> }
<factor> ::= <nonterm> | <terminal>
| '(' <expr> ')' | '[' <expr> ']' | '{' <expr> '}'
<nonterm> ::= '<' 'letter' { 'letter' | 'digit' } '>'
<terminal> ::= ' ' ' { 'character' } ' ' '

19

Context-Sensitive Grammars

- A grammar $G = <T, N, S, P>$ is *context-sensitive* if all productions are of the form
  $x \rightarrow y$
  where $x, y \in (T \cup N)^+$ and
  $|x| \leq |y|$

20

Unrestricted Grammars

- A grammar $G = <T, N, S, P>$ is *unrestricted* if all productions are of the form
  $x \rightarrow y$
  where $x \in (T \cup N)^+$ and $y \in (T \cup N)^*$

21

Attribute Grammars

- An *attribute grammar* is a *context-free grammar* with associated *attributes* and *semantic rules*
- Each *grammar symbol* is associated with a set of *attributes*
- Each *production* is associated with a set of *semantic rules* for computing attributes

22

An Example

Production	*Semantic Rules*
L → E '\n'	print(E.val)
E → E_1 '+' T	E.val := E_1.val + T.val
E → T	E.val := T.val
T → T_1 '*' F	T.val := T_1.val * F.val
T → F	T.val := F.val
F → '(' E ')'	F.val := E.val
F → **digit**	F.val := **digit**.val

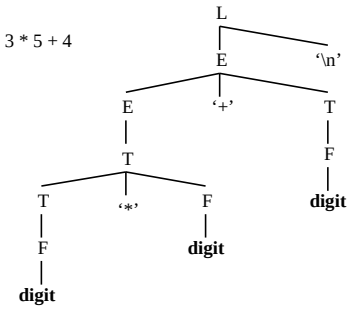
23

Parse Trees

- A *parse tree* is a graphical representation for a derivation that filters out the order of choosing nonterminals for rewriting

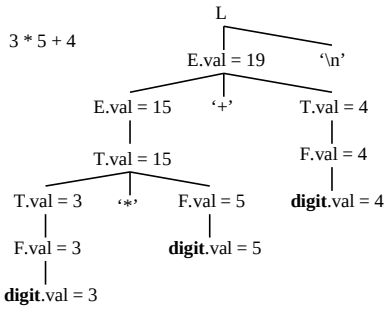
24

An Example



25

Decorated Parse Trees



26

Attributes

- An attribute of a node (grammar symbol) in the parse tree is *synthesized* if its value is computed from that of its *children*
- An attribute of a node in the parse tree is *inherited* if its value is computed from that of its *parent* or *siblings*

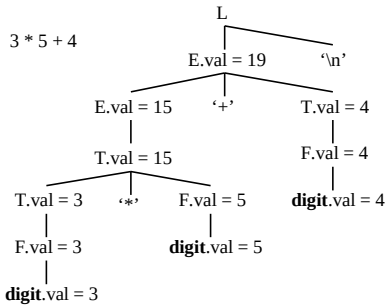
27

Synthesized Attributes

Production	***Semantic Rules***
L → E '\n'	print(E.val)
E → E_1 '+' T	E.val := E_1.val + T.val
E → T	E.val := T.val
T → T_1 '*' F	T.val := T_1.val * F.val
T → F	T.val := F.val
F → '(' E ')'	F.val := E.val
F → **digit**	F.val := **digit**.val

28

Synthesized Attributes



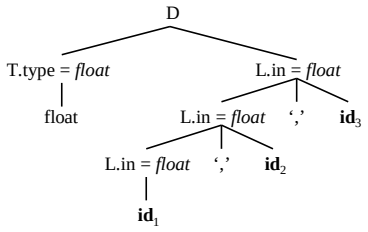
29

Inherited Attributes

Production	***Semantic Rules***
D → T L	L.in := T.type
T → **int**	T.type := *integer*
T → **float**	T.type := *float*
L → L_1 ',' **id**	L_1.in := L.in Addtype(**id**.entry, L.in)
L → **id**	Addtype(**id**.entry, L.in)

30

Inherited Attributes



31

Post Systems

- Post systems are *deductive formal systems*
- Post systems have the same power as *unrestricted grammars*
- Grammars are useful in *describing* certain aspects of programming languages
- Deductive systems are useful in various capabilities in *reasoning* about programming languages

32

Post Systems

- A Post system is a 3-tuple <*S*, *V*, *R*>
- *S* is a finite set of *signs* or the *alphabet*
- *V* is a finite set of *variables*
- *R* is a finite set of *inference rules*

33

Inference Rules

- A *term* is a string of signs and variables
- A *word* is a string of signs
- An inference rule is a *figure* of the form

$$\frac{t_1 \; t_2 \ldots t_n}{t}$$

where t, t_1, t_2, … , t_n are all *terms*

- The t_i are called *premises* and the t *conclusion* of the inference rule

34

Axioms

- An *axiom* is an inference rule without premises
- An *instance* of an inference rule is obtained from an inference rule by substituting *strings of signs* for all *variables*, the same string being substituted for all occurrences of one and the same variable

35

Proofs

- An *instance* of an *axiom* is a proof of the *conclusion*
- If Pr_1, Pr_2, …, Pr_n are proofs of a_1, a_2, …, a_n, respectively, and

$$\frac{a_1 \; a_2 \ldots a_n}{a}$$

is an *instance* of an *inference rule*, then

$$\frac{Pr_1 \; Pr_2 \ldots Pr_n}{a}$$

is a proof of a

36

Languages

- A word is *provable* in a Post system if we can find a proof of it
- The *language* derived by a Post system is the *set of words* provable from the system

37

An Example

- The set of *signs* is {*N*, |}
- The set of *variables* is {*x*}
- The set of *inference rules* is {

$$\frac{}{N} \quad , \quad \frac{Nx}{Nx|}$$

}

38

An Example

- *N* is a word provable by the system and its proof is

$$\frac{}{N}$$

- *N*| is a word provable by the system and its proof is

$$\frac{\dfrac{}{N}}{N|}$$

- This system derives words of the form $N|_{39}..|$

Another Example

- The set of *signs* is {*N*, |, +, =}
- The set of *variables* is {*x*, *y*, *z*}
- The set of *inference rules* is {

$$\frac{}{N} \quad , \quad \frac{Nx}{Nx|} \quad , \quad \frac{Ny}{+\,y = y} \quad , \quad \frac{x + y = z}{x| + y = z|}$$

}

40

Another Example

- The word || + || = |||| (2 + 2 = 4) is provable by the system and its proof is

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{N}}{N|}}{N||}}{+ \,||\, = \,||}}{|\, + \,||\, = \,|||}}{||\, + \,||\, = \,||||}$$

41

Normal Forms

- The inference rules in the previous examples have the following properties
- *Each inference rule has at most one premise*
- *A variable occurs at most once in a premise*
- These restrictions, however, do not diminish the power of Post systems

42

Propositional Logic

- The set of *signs* is {*P*, |, *N*, *C*, *F*, *Th*}
- The set of *variables* is {*x*, *y*, *z*}
- There are two *inference rules* for propositions

$$\frac{}{P} \quad , \quad \frac{Px}{Px|}$$

43

Propositional Logic

- There are three *inference rules* for formulas

$$\frac{Px}{FPx} \quad , \quad \frac{Fx}{FNx} \quad , \quad \frac{Fx \quad Fy}{FCxy}$$

- There are four *inference rules* for theorems

$$\frac{Fx}{ThCCNxxx} \quad , \quad \frac{Fx \quad Fy}{ThCxCNxy} \quad ,$$

$$\frac{Fx \quad Fy \quad Fz}{ThCCxyCCyzCxz} \quad , \quad \frac{ThCxy \quad Thx}{Thy}$$

44

An Example

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P}}{FP}\dfrac{\dfrac{}{P}}{FP}}{FNP\ FP}\dfrac{\dfrac{}{P}}{FP}\dfrac{\dfrac{}{P}}{P|}}{FCNPP\quad FP\quad FP|}\dfrac{\dfrac{}{P}}{FP}}{ThCCCNPPPCCPP|CCNPPP|\quad ThCCNPPP}$$

$$ThCCPP|CCNPPP|$$

45

Semantics

- An *assignment* σ is a function from propositions to the set $\{\bot, \top\} = Bool$
- If $\sigma(i) = \top$, then P_i is a *true* proposition
- Σ is the set of *all* possible assignments
- Each proposition is associated a *nonnegative number*
  1. $\mathcal{N}【P】 = 0$
  2. $\mathcal{N}【Px|】 = \mathcal{N}【Px】 + 1$

46

Semantics

- The *semantic* is given by a function $\mathcal{M}$ from Σ to *Bool*:
  1. $\mathcal{M}【FPx】\ \sigma = \sigma(\mathcal{N}【Px】)$
  2. $\mathcal{M}【FNx】\ \sigma = \top$, iff $\mathcal{M}【Fx】\ \sigma = \bot$
  3. $\mathcal{M}【FCxy】\ \sigma = \top$,
     iff $\mathcal{M}【Fx】\ \sigma = \bot$ or $\mathcal{M}【Fy】\ \sigma = \top$
- A formula *Fx* is *valid* if $\mathcal{M}【Fy】\ \sigma = \top$ for all assignments σ

47

Theorem

- The term *Thx* is *derivable* in the Post system for propositional calculus if and only if the formula *Fx* is *valid*

proof:
The three axioms are valid, and modus ponens preserves validity, so all derivable formulas are valid

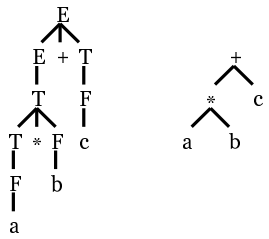
48

Abstract Syntax

- The *abstract syntax* of a language identifiers the *meaningful* components of each constructs in the language
- The *concrete syntax* adds some *syntactic sugars* to improve *user-friendliness*

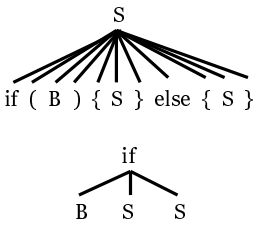
49

An Example



50

An Example



51

Constructors and Selectors

- Functions used to build constructs are called *constructors*
  - *assignment*(*I*, *E*)
  - *while*(*B*, *S*)
  - *if*(*B*, S_1, S_2)
- Functions used to take constructs apart are called *selectors*
  - *ident*, *rhs*, *whileCond*, *whileBody*, *ifCond*, *thenStmt*, *elseStmt*

52