

Syntax

Prof. Evan Chang
Meeting 3, CSCI 3155, Fall 2009



Announcements

- Assignment 1 due Thu Sep 3 at 11:55pm
 - Find a partner, Post on general forum
- Want to give everyone a chance to reflect on questions in class
 - Think for a moment, then raise your hand
 - I'll look for a few hands before calling someone

2

Review

- Languages are adopted to fill a void and may have little to do with "quality"
- No universally accepted metrics for good design
- There are some criteria and characteristics that help us evaluate languages
- Table 1.1 (in Sebesta) provides a starting point

3

Syntax

What is the purpose of a language specification?

- Who uses a language specification?
 - language implementers
 - language users
 - language designers

5

What do we need to describe in a language specification?

- Two things
 - Syntax — forms of expr, stmts, program
 - semantics — meaning of exprs, stmts, progr
 - "how things evaluate"

6

What is the difference between syntax and semantics?

Example: =
 ↗ assignment
 ↘ equality

- Can a given syntax have two different semantics (in different languages)?

7

Today: Syntax

- We need a way to unambiguously describe the syntax of a language
- Only one part of understanding a language (what is the other big part?)

↘ semantics

8

Skill: Standard notation for syntax

The screenshot shows the Python 2.6.2 documentation for section 5.3.4. Calls. It explains that a call to a callable object (like a function) can have a series of arguments. It lists the syntax for arguments: positional arguments, keyword arguments, and a mix of both. It also notes that a trailing comma is allowed after positional arguments but does not affect semantics. The code snippet shows a function call with positional arguments, keyword arguments, and a mix of both.

```

call ::= primary "(" [expression_list] ")"
expression_list ::= expression ("," keyword_argument)*
keyword_argument ::= keyword "=" expression
expression ::= expression "[" expression "]" | expression "." expression | "+" expression | "-" expression | "-">
positional_argument ::= expression "(" "," expression ")"
keyword_argument ::= keyword "=" ["keyword_argument"]
keyword ::= identifier "=" expression
    
```

10

Language Terminology

- A language is a set of strings ^{of characters in some alphabet (Σ)}
- A sentence is a string in a language
- A (context-free) grammar describes (context-free) languages
- BNF (Backus-Naur Form) is a notation for describing languages (context-free)

Language Terminology

- A language is a set of strings
- A sentence is a string of a language
- A (context-free) grammar describes (context-free) languages
- BNF (Backus-Naur Form) is a notation for (context-free) languages

11

One-Slide Summary

- A context-free grammar (=BNF) is a notation for specifying formal languages. They contain terminals, non-terminals, and productions.
- A derivation exhibits a string recognized by a grammar
 - A parser takes a sequence of tokens as input and produces a parse tree or derivation (if the input is valid).

12

Lexical vs. Syntactic *characters*

<ifthenelse> → 'i' 'f' <expr> 't' 'h' 'e' 'n' ...

characters

<ifthenelse> → <if_token> <expr> <then_token>

<ifthenelse> → if <expr> then

13

Lexical vs. Syntactic *'a' 'b' 'c'*

- A **lexeme** is a seq of chars into syntactic unit
- lowest-level unit
- A **token** is category of lexemes

14

Lexical vs. Syntactic

- A **lexeme** is a sequence of characters that form a syntactic unit
- A **token** is a category of lexemes

15

Lexical vs. Syntactic

- A **lexer** (i.e., lexical analyzer) groups characters into **lexemes** and classifies them into **tokens**
- A **parser** (i.e., syntactic analyzer) recognizes strings of **tokens**

16

Lexical vs. Syntactic

- A **lexer** (i.e., lexical analyzer) groups characters into **lexemes** and classifies them into **tokens**
- A **parser** (i.e., syntactic analyzer) recognizes strings of **lexemes**

17

Lexical vs. Syntactic (Aside)

- A **lexer** (i.e., lexical analyzer) groups characters into **lexemes** and classifies them into **tokens**
 - regular languages / regular expressions useful here
- A **parser** (i.e., syntactic analyzer) recognizes strings of **lexemes**
 - context-free languages / grammars useful here
- Topic for a formal languages course

18

Recognizers and Generators

- A grammar can be viewed as either
 - **generating** a the set of strings in a language L or
 - **recognizing** whether a given string s is in a language L

19

Example Grammar (Book Notation)

$\langle \text{stmt} \rangle \rightarrow \langle \text{assign} \rangle \mid \langle \text{return} \rangle$ *production*
 $\langle \text{assign} \rangle \rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$
 $\langle \text{return} \rangle \rightarrow \text{return} \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle \rightarrow \langle \text{id} \rangle + \langle \text{id} \rangle \mid \langle \text{id} \rangle \mid \langle \text{number} \rangle$
 $\langle \text{id} \rangle \rightarrow a \mid b \mid \dots \mid z$ *terminals*
 $\langle \text{number} \rangle \rightarrow 0 \mid 1 \mid \dots \mid 9$ *non-terminal*

- Which are **terminals**, **non-terminals**, and **productions**?

20

Example Grammar (Common Notation)

$\text{stmt} ::= \text{assign} \mid \text{return}$
 $\text{assign} ::= \text{id} = \text{expr}$
 $\text{return} ::= \text{return expr}$
 $\text{expr} ::= \text{id} + \text{id} \mid \text{id} \mid \text{number}$
 $\text{id} ::= a \mid b \mid \dots \mid z$
 $\text{number} ::= 0 \mid 1 \mid \dots \mid 9$

Handwritten examples:
 $a =$
 $\text{return } 0$
 $\text{return } a + b$
 $b = 5 + a$

- What are some example sentences?

21

Example Grammar: Terminology

$\text{stmt} ::= \text{assign} \mid \text{return}$
 $\text{assign} ::= \text{id} = \text{expr}$
 $\text{return} ::= \text{return expr}$
 $\text{expr} ::= \text{id} + \text{id} \mid \text{id} \mid \text{number}$
 $\text{id} ::= a \mid b \mid \dots \mid z$
 $\text{number} ::= 0 \mid 1 \mid \dots \mid 9$

- Which are **terminals**, **non-terminals**, and **productions**?

22

Example Grammar: Sentences

$\text{stmt} ::= \text{assign} \mid \text{return}$
 $\text{assign} ::= \text{id} = \text{expr}$
 $\text{return} ::= \text{return expr}$
 $\text{expr} ::= \text{id} + \text{id} \mid \text{id} \mid \text{number}$
 $\text{id} ::= a \mid b \mid \dots \mid z$
 $\text{number} ::= 0 \mid 1 \mid \dots \mid 9$

- What are some example sentences?

23

In the language?

- $a + b = c + d$ *No*
- $a = b + c + d$ *No*
- $a = b + 1$ *No*
- $a = 1 + b$ *No*
- $\text{myvar} = \text{myvar1} + \text{myvar2}$ *No*

$\text{stmt} ::= \text{assign} \mid \text{return}$
 $\text{assign} ::= \text{id} = \text{expr}$
 $\text{return} ::= \text{return expr}$
 $\text{expr} ::= \text{id} + \text{id} \mid \text{id} \mid \text{number}$
 $\text{id} ::= a \mid b \mid \dots \mid z$
 $\text{number} ::= 0 \mid 1 \mid \dots \mid 9$

24

How could we generalize? To include 2

1. $a + b = c + d$ X
2. $a = b + c + d$ YES
3. $a = b + 1$ YES
4. $a = 1 + b$ NO
5. $myvar = myvar1 + myvar2$ NO

```

stmt ::= assign | return
assign ::= id = expr
return ::= return expr
expr ::= id + id | id | number
id ::= a | b | ... | z
number ::= 0 | 1 | ... | 9
    
```

25

How could we generalize? To include 5

1. $a + b = c + d$
2. $a = b + c + d$
3. $myvar = myvar1 + myvar2$
4. $a = b + 1$
- 5. $a = 1 + b$

Derivation for 4

```

stmt => assign
    => id = expr
    => id = id + expr
    => id = id + number
    => id = id + 1
    => id = b + 1
    => a = b + 1
    
```

```

expr ::= id + expr
      | id
      | number + expr
    
```

```

stmt ::= assign | return
assign ::= id = expr
return ::= return expr
expr ::= id + id | id | number
id ::= a | b | ... | z
number ::= 0 | 1 | ... | 9
    
```

26

More examples ((), (), (), X

- Write a grammar that generates matching open and close braces ε 3

$matchingbr ::= \{ matchingbr \} | \epsilon | matchingbr matchingbr$

$try1 ::= \{ \} | \{ try1 \} | \{ \} try1 | try1 \{ \}$

$(()) ((())) ac$ $try2 ::= \{ \} | \{ try2 \} | try2 try2$
 $| \epsilon$

27

More examples

- Write a grammar that generates strings with an odd number of 'a's

$\langle odda \rangle ::= a \langle odda \rangle a a$

$\langle odda2 \rangle ::= a | a \langle odda2 \rangle a$

28

For Next Time

- Reading
- Online discussion forum
- Homework assignment 1

29