

Syntax and Semantics of Prog. Languages

Syntax:

Describes what a legal program looks like

Semantics:

Describes what a correct (legal) program means

A formal language is a (possibly infinite) set of sentences (finite sequences of symbols) over a finite alphabet Σ of (terminal) symbols.

Example:

Syntax and Semantics of Prog. Languages

The syntax of programming languages is often defined in two layers: *tokens* and *sentences*.

- *tokens* – basic units of the language

Question: How to spell a token (word)?

Answer: regular expressions

- *sentences* – legal combination of tokens in the language

Question:

How to build correct sentences with tokens?

Answer: (context-free) grammars (CFG)

- E.g., Backus-Naur form (BNF) is a formalism used to express the syntax of programming languages.

Formalisms for Lexical and Syntactic Analysis

1. Lexical Analysis: Converts source code into sequence of tokens.
2. Syntax Analysis: Structures tokens into parse tree.

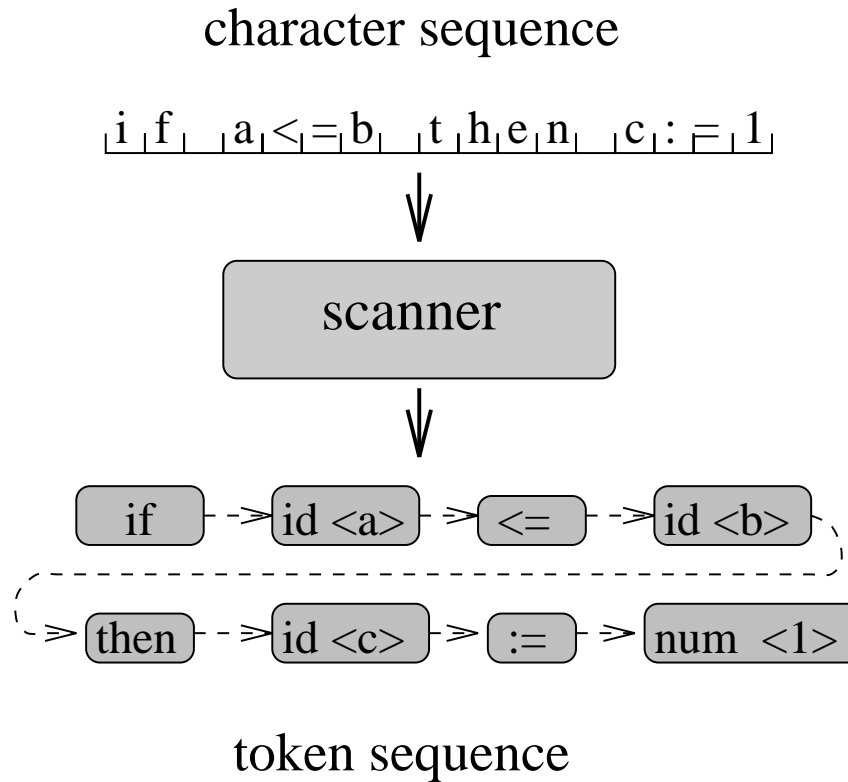
Two issues in **Formal Languages**:

- Language Specification → formalism to describe what a valid program (sentence) looks like.
- Language Recognition → formalism to describe a machine and an algorithm that can verify that a program is valid or not.

For (2), we use **context-free grammars** to specify programming languages. Note: recognition, i.e., parsing algorithms using PDAs (push-down automata) will be covered in **CS415**).

For (1), we use **regular grammars/expressions** for specification and **finite (state) automata** for recognition.

Lexical Analysis (Lexical Syntax)



Tokens (Terminal Symbols of CFG, Words of Lang.)

- Smallest “atomic” units of syntax
- Used to build all the other constructs
- Example, Pascal:

keywords: program begin if then ...

= * / - < > = <= >= <>

() [] ; := . , ...

number (Example: 3.14 28 ...)

identifier (Example: b square addEntry ...)

Lexical Analysis (cont.)

Identifiers

- Names of variables, etc.
- Sequence of terminals of restricted form;
Example, Pascal: **A31**, but not **1A3**
- Upper/lower case sensitive?

Keywords

- Special identifiers which represent tokens in the language
- May be reserved (*reserved words*) or not
 - E.g., Pascal: “**if**” is reserved.
 - E.g., FORTRAN: “**if**” is not reserved.

Delimiters – When does character string for token end?

- Example: identifiers are longest possible character sequence that does not include a delimiter
- Few delimiters in Fortran (not even ‘`_`’)
 - `DO I = 1.5` same as `DOI=1.5`
- Most languages have more delimiters such as ‘`_`’, new line, keywords, . . .

Regular Expressions

A syntax (notation) to specify regular languages.

RE r

Language $L(r)$

a $\{a\}$

ϵ $\{\epsilon\}$

$r \mid s$ $L(r) \cup L(s)$

rs $\{rs \mid r \in L(r), s \in L(s)\}$

r^+ $L(r) \cup L(rr) \cup L(rrr) \cup \dots$
(any number of r 's concatenated)

r^* $\{\epsilon\} \cup L(r) \cup L(rr) \cup L(rrr) \cup \dots$
($r^* = r^+ \mid \epsilon$)

(s) $L(s)$

(all left-assoc. in order of increasing prec.)

\Rightarrow **Note:** Inductive definition!

Examples of Expressions

RE

Language

$a|bc$

$(a|b)c$

$a\epsilon$

$a^*|b$

ab^*

$ab^*|c^+$

$(a|b)^*$

$(0|1)^*1$

Regular Expressions for Programming Languages

Let letter stand for $A \mid B \mid C \mid \dots \mid Z$

Let digit stand for $0 \mid 1 \mid 2 \mid \dots \mid 9$

identifier:

integer constant:

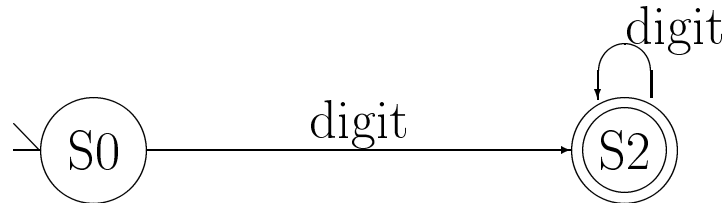
real constant:

Recognizers for Regular Expressions

Example 1: integer constant

RE: digit^+

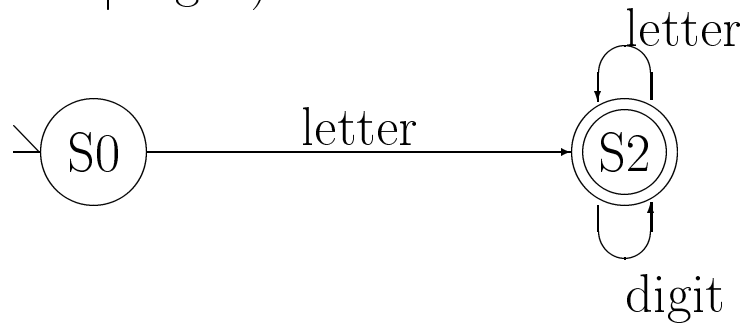
FSA:



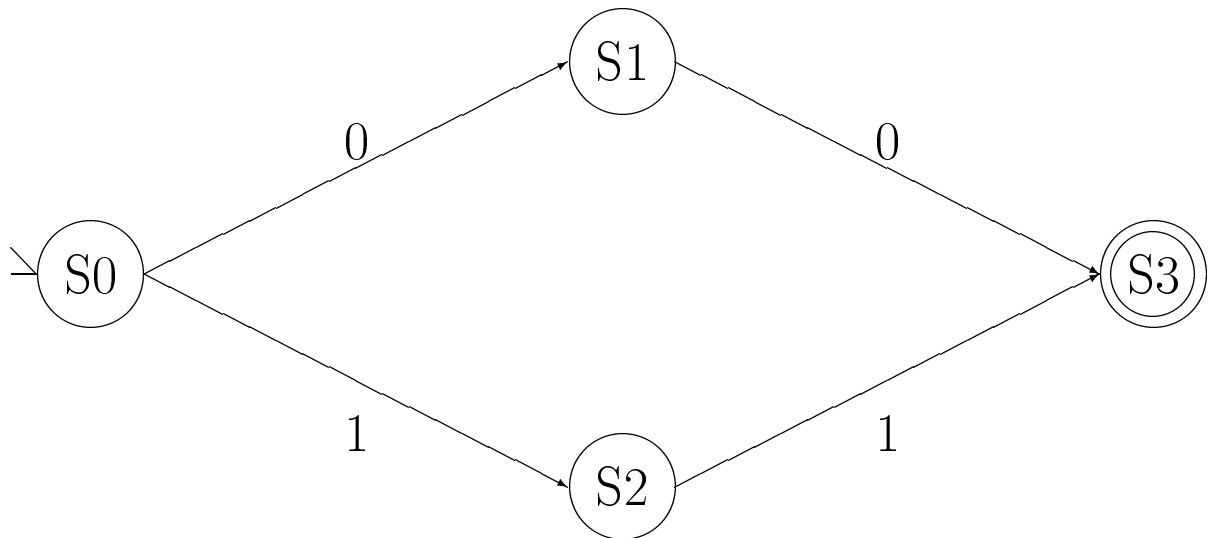
Example 2: identifier

RE: $\text{letter} (\text{letter} \mid \text{digit})^*$

FSA:



Finite State Automata



A Finite-State Automaton is a quadruple:

$\langle S, s, F, T \rangle$

- S is a set of *states*, e.g., $\{S0, S1, S2, S3\}$
- s is the *start state*, e.g., $S0$
- F is a set of *final states*, e.g., $\{S3\}$
- T is a set of *labeled transitions*, of the form
 $(state, input) \mapsto state$
[i.e., $S \times \Sigma \rightarrow S$]

Finite State Automata

Transitions can be represented using a transition table:

		0	1	Input
State	S0	S1	S2	
	S1	S3	-	
	S2	-	S3	

An FSA *accepts* or *recognizes* an input string iff there is some path from its start state to a final state such that the labels on the path are that string.

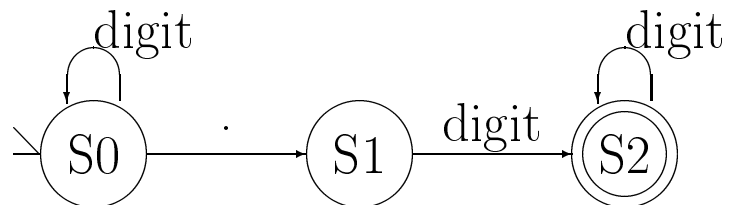
Lack of entry in the table (or no arc for a given character) indicates an error—reject.

Regular Expressions and FSAs

Example 3: Real constant

RE: $\text{digit}^*.\text{digit}^+$

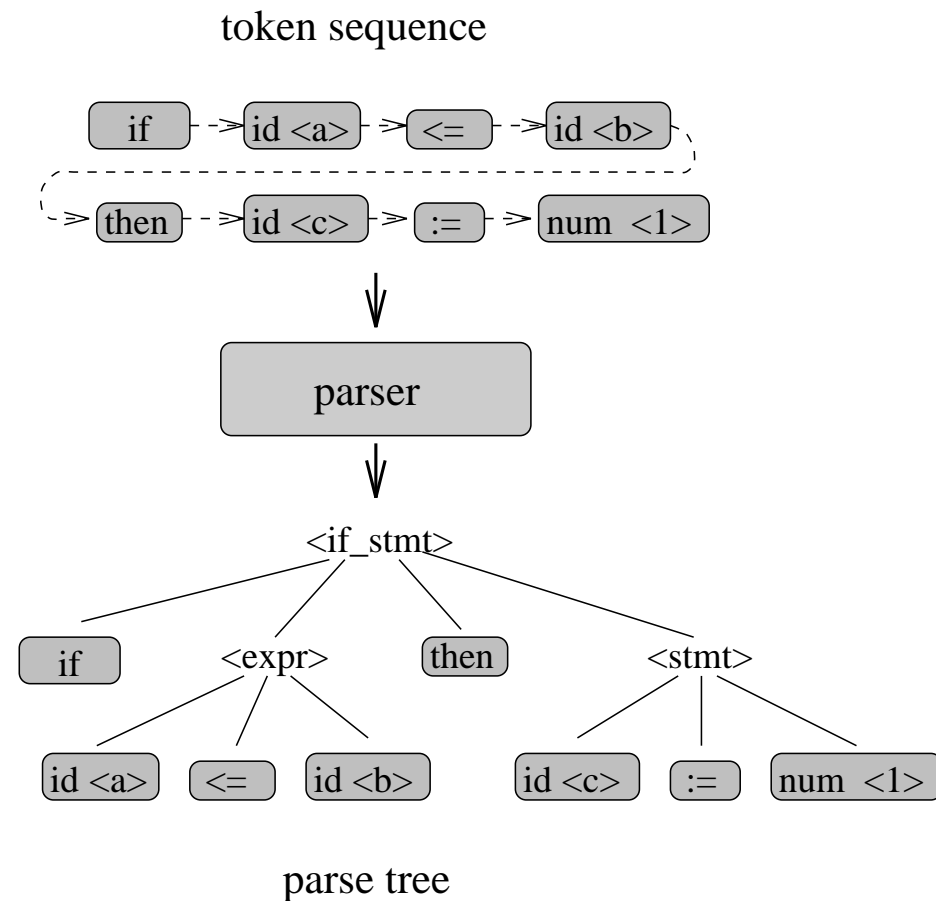
FSA:



Transition Table:

States	Inputs	
	<i>digit</i>	.
<i>S0</i>	<i>S0</i>	<i>S1</i>
<i>S1</i>	<i>S2</i>	—
S2	<i>S2</i>	—

Syntax Analysis



BNF (Backus-Naur Form): A formal notation for describing syntax— how components can be combined to form a valid program.

- To specify which programs are legal
- To describe the structure of programs (*parse tree*)
- BNF is a way of writing context free grammars (CFGs)

Context Free Grammars (CFGs)

- A formalism for describing languages
- CFGs are a quadruple $\langle T, N, P, S \rangle$:
 1. A set T of terminal symbols
 2. A set N of nonterminal symbols
 3. A set P production rules
 4. A special start symbol S
- BNF is a notation for describing CFGs.

A partial example:

...

$\langle \text{if-stmt} \rangle ::= \mathbf{if} \langle \text{expr} \rangle \mathbf{then} \langle \text{stmt} \rangle$

$\langle \text{expr} \rangle ::= \mathbf{id} \langle = \mathbf{id} \rangle$

$\langle \text{stmt} \rangle ::= \mathbf{id} := \mathbf{num}$

Elements of BNF Syntax

Terminal Symbol: **Symbol-In-Boldface**

Non-Terminal Symbol: *Symbol-In-Angle-Brackets*

Production Rule:

Non-Terminal ::= Sequence of Symbols

or

Non-Terminal ::= Sequence | Sequence | ...

Alternative Symbol: |

Empty String: ϵ

How a BNF Grammar Describes a Language

- A *sentence* is a sequence of terminal symbols (tokens)
- A *language* is a set of (acceptable) sentences
- The language $L(G)$ of a BNF grammar G is the set of sentences generated using the grammar:
 - Begin with start symbol.
 - Iteratively replace non-terminals with terminals according to rules.

Simple Grammar (\mathcal{G})

Terminals letters, digits, $:=$

Nonterminals $\langle \text{letter} \rangle$ $\langle \text{digit} \rangle$ $\langle \text{identifier} \rangle$
 $\langle \text{stmt} \rangle$

Productions

1. $\langle \text{letter} \rangle ::= A \mid B \mid C \mid \dots \mid Z$
2. $\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$
3. $\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle \mid$
 $\langle \text{identifier} \rangle \langle \text{letter} \rangle \mid$
 $\langle \text{identifier} \rangle \langle \text{digit} \rangle$
4. $\langle \text{stmt} \rangle ::= \langle \text{identifier} \rangle := 0$

Start Symbol $\langle \text{stmt} \rangle$

Derivation in a Grammar (\mathcal{G})

Is $X_2 := 0 \in L(\mathcal{G})$, i.e., can $X_2 := 0$ be derived in G ?

Terminology: leftmost (or canonical) derivation.

Next Lecture

Things to do:

- read Sethi, Ch 1 and Chs 2.1- 2.4
read Louden, Ch. 4 (4.1-4.5, 4.7), ASU Ch 2.2, 3.3, 3.4
- First homework will be posted by Wednesday

Recitations HAVE STARTED!