CSCI 742 - Compiler Construction

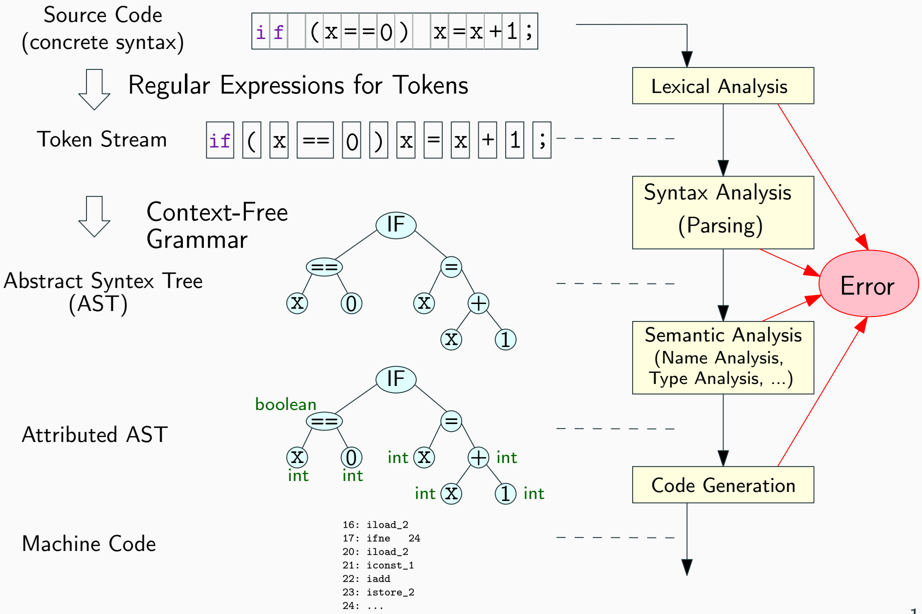
Lecture 3

Introduction to Regular Expressions

Instructor: Hossein Hojjat

January 22, 2018

Compiler Phases



Lexical Analysis

- **Goal:** Partition input string into meaningful elements called tokens
- Token is a syntactic category:
  - In English: verbs, nouns, pronouns, adverbs, adjectives , ...
  - In programming language: identifier, integer, keyword, semicolon, ...

Input:

i	**f**	**(**	**x**	**=**	**=**	**0**	**)**		**x**	**=**	**x**	**+**	**1**	**;**

Output:

IF , LPAREN , ID(x) , EQUALS , INTLIT(0) , RPAREN , ID(x) , EQSIGN , ID(x) , PLUS , INTLIT(1) , SEMICOLON

Lexical Analysis

- A lexical analyzer ("lexer" or "scanner") has the following tasks:

1) Recognize substrings corresponding to tokens
2) Return tokens with their categories

- There are finitely many token categories
  - Identifier
  - LPAREN
  - RPAREN
  - COLON
  - ... (many, but finitely many)
- There is unbounded number of instances of token classes like Identifier

Lexical Analysis

- Output of lexical analysis is a stream of tokens
  which is input to parser
- Parser relies on token category
  - For example, it treats identifiers and keywords differently
- We use token categories when writing grammars for parsing
- **Regular languages** can be used to describe valid tokens of almost every programming language

Languages

- Alphabet Σ: Finite set of elements
  - For lexer: Characters
  - For parser: Token classes
- Words (strings): Sequence of elements from the alphabet Σ
  - Special case: empty word ϵ
- $\Sigma*$: Set of all words over Σ
- Language over Σ: a subset of $\Sigma*$

Languages Example

- $\Sigma = \{a, b\}$
- $\Sigma* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \cdots\}$

Examples of two languages, subsets of $\Sigma*$:

- $L_1 = \{a, bb, ab\}$ (finite language, three words)
- $L_2 = \{ab, abab, ababab, \cdots\}$ = $\{(ab)^n | n \geq 1\}$ (infinite language)

Operation on Languages

Operation	Definition
union of L_1 and L_2 written $L_1 \cup L_2$	$L_1 \cup L_2 = \{s \mid s \in L_1 \vee s \in L_2\}$
concatenation of L_1 and L_2 written $L_1.L_2$	$L_1.L_2 = \{st \mid s \in L_1 \wedge t \in L_2\}$
Kleene closure of L written $L*$	$L* = \bigcup_{i=0}^{\infty} L^i$
positive closure of L written $L+$	$L+ = \bigcup_{i=1}^{\infty} L^i$

- L^i is recursively defined

  $L^0 = \{\epsilon\}$ (the language consisting only of the empty string)

  $L^1 = L$

  $L^{i+1} = \{wv : w \in L^i \wedge v \in L\}$ for each $i > 0$

Star Operation: Example

- $L = \{a, ab\}$
- $L.L = \{aa, aab, aba, abab\}$
- $L* = \{\epsilon, a, ab, aa, aab, aba, abab, aaa, ...\}$
- $= \{w \mid$ immediately before each b there is a $\}$

Star Operation: Example

- Star allows us to define infinite languages starting from finite ones
- We can use it to describe some of those infinite but reasonable languages

Star Operation: Example

- Star allows us to define infinite languages starting from finite ones
- We can use it to describe some of those infinite but reasonable languages

- When is $L*$ finite?

Star Operation: Example

- Star allows us to define infinite languages starting from finite ones
- We can use it to describe some of those infinite but reasonable languages

- When is $L*$ finite?
- Only in these two cases:
- $\emptyset^* = \{\epsilon\}$ (because $\emptyset^0 = \{\epsilon\}$)
- $\{\epsilon\}^* = \{\epsilon\}$

Properties of Words

- Let $w_i \in \Sigma*$ be a word
- Concatenation is associative:

$$(w_1.w_2).w_3 = w_1.(w_2.w_3)$$

- Empty word ϵ is left and right identity:

$$w.\epsilon = w$$

$$\epsilon.w = w$$

- Cancellation property

- If $w_1.w_3 = w_1.w_2$ then $w_3 = w_2$
- If $w_3.w_1 = w_2.w_1$ then $w_3 = w_2$

- There are many other properties, many easily provable from definition of operations

Properties of Words

Length of a word

- $|\epsilon| = 0$
- $|c| = 1 \qquad$ if $c \in \Sigma$
- $|w_1.w_2| = |w_1| + |w_2| \qquad w_i \in \Sigma*$

Reverse of a word

- $\epsilon^{-1} = \epsilon$
- $c^{-1} = c \qquad$ if $c \in \Sigma$
- $(w_1.w_2)^{-1} = w_2^{-1}.w_1^{-1}$

Fact about Indexing Concatenation

- Concatenation of w and v has these letters:

$$w_{(0)} \cdots w_{(|w|-1)} . v_{(0)} \cdots v_{(|v|-1)}$$

- Thus, for every i where $0 \leq i \leq |w| + |v| - 1$

$$\begin{aligned} (wv)_{(i)} &= w_{(i)}, & \text{if } i < |w| \\ (wv)_{(i)} &= v_{(i-|w|)}, & \text{if } i \geq |w| \end{aligned}$$

Regular Expressions

- Notations to describe regular languages
  - Regular expressions (RE)
  - Regular grammars

- Regular expression over alphabet Σ:
  1. ϵ is a RE denoting the set $\{\epsilon\}$
  2. if $a \in \Sigma$, then a is a RE denoting $\{a\}$
  3. if r and s are REs, denoting $L(r)$ and $L(s)$, then:
  - $r \mid s$ is a RE denoting $L(r) \cup L(s)$
  - $r \;.\; s$ is a RE denoting $L(r).L(s)$
  - $r*$ is a RE denoting $L(r)*$

- Precedence: Closure then Concatenation then Alternation

Regular Expressions

- Regular expressions are just a notation for some particular operations on languages

letter (letter | digit)*

- Denotes the set

letter (letter ∪ digit)*

- Any finite language $\{w_1, \cdots, w_n\}$ can be described using regular expression

$$w_1 \mid \cdots \mid w_n$$

Regular Expressions

Some RE operators can be defined in terms of previous ones

- $[a..z] = a|b|\cdots|z$ (use ASCII ordering)
- $e?$ (optional expression) $= e \mid \epsilon$
- $e+$ (repeat at least once)
- $!e$ (complement) $= \Sigma * \setminus e$
- $e_1 \& e_2$ (intersection) $= !(!e_1 \mid !e_2)$

Exercise

Find a regular expression that generates all alternating sequences of 0 and 1 with arbitrary length (including lengths zero, one, two, ...).

For example, the alternating sequences of length one are 0 and 1, length two are 01 and 10, length three are 010 and 101. Note that no two adjacent character can be the same in an alternating sequence.