

Lecture 6

How to Describe Tokens?

Regular Languages are the most popular for specifying tokens because

- These are based on simple and useful theory,
- Are easy to understand and
- Efficient implementations exist for generating lexical analysers based on such languages.

Languages

Let Σ be a set of characters. Σ is called the *alphabet*. A *language* over Σ is set of strings of characters drawn from Σ . Here are some examples of languages:

- Alphabet = English characters
Language = English sentences
- Alphabet = ASCII
Language = C++, Java, C# programs

Languages are sets of strings (finite sequence of characters). We need some notation for specifying which sets we want. For lexical analysis we care about *regular languages*. Regular languages can be described using *regular expressions*. Each regular expression is a notation for a regular language (a set of words). If **A** is a regular expression, we write **L(A)** to refer to language denoted by **A**.

Regular Expression

A *regular expression (RE)* is defined inductively

- a** ordinary character from Σ
- e** the empty string
- R|S** either R or S
- RS** R followed by S (concatenation)
- R*** concatenation of R zero or more times ($R^* = \epsilon | R | RR | RRR \dots$)

Regular expression extensions are used as convenient notation of complex RE:

- R?** $\epsilon | R$ (zero or one R)
- R⁺** RR^* (one or more R)
- (R)** R (grouping)
- [abc]** $a|b|c$ (any of listed)
- [a-z]** $a|b|\dots|z$ (range)
- [^ab]** $c|d|\dots$ (anything but 'a','b')

Here are some Regular Expressions and the strings of the language denoted by the RE.

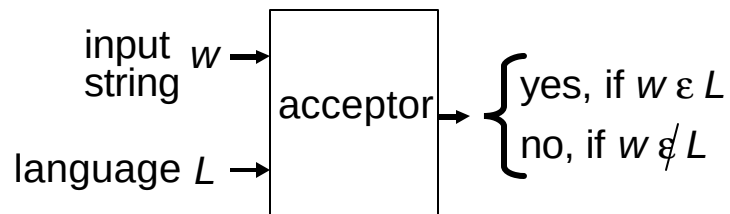
<u>RE</u>	<u>Strings in $L(R)$</u>
a	"a"
ab	"ab"
a b	"a" "b"
(ab)*	"", "ab" "abab" ...
(a ε)b	"ab" "b"

Here are examples of common tokens found in programming languages.

digit	'0' '1' '2' '3' '4' '5' '6' '7' '8' '9'
integer	digit digit*
identifier	[a-zA-Z_][a-zA-Z0-9_]*

Finite Automaton

We need mechanism to determine if an input string w belongs to $L(R)$, the language denoted by regular expression R . Such a mechanism is called an *acceptor*.

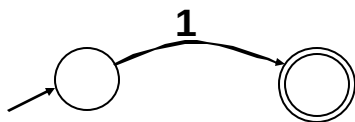


The acceptor is based on Finite Automata (FA). A *Finite Automaton* consists of

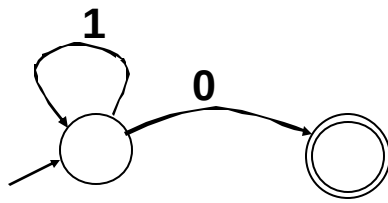
- An input alphabet Σ
- A set of states
- A start (initial) state
- A set of transitions
- A set of accepting (final) states

A finite automaton *accepts* a string if we can follow transitions labeled with characters in the string from start state to some accepting state. Here are some examples of FA.

- A FA that accepts only "1"



- A FA that accepts any number of 1's followed by a single 0



- A FA that accepts ab^*a ($\Sigma: \{a,b\}$)

