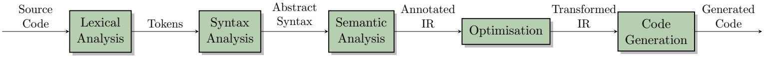
Lexical Analysis
Programming Languages

Sujit Kumar Chakrabarti

IIITB

Language Processing

Compilers



Lexical Analysis

Example – Natural Languages

Input: Ram killed Ravan

Lexical Analysis

Example – Natural Languages

Input:	Ram	killed	Ravan
Rule:	NOUN	VERB	NOUN

Lexical Analysis

Example – Natural Languages

Input:	Ram	killed	Ravan
Lexical Analysis Rule:	↓	↓	↓
	NOUN	VERB	NOUN

Lexical Analysis

Example – Programming Languages

`int` `main` `(` `)` `{` `printf` `(` `Hello World!` `)` `;` `}`

Terminology

- **Token classes/Tokens.** e.g. KW, ID, (, } etc.
- **Lexemes.** e.g. `"int"`, `"main"`, `"("`, `"}"` etc.

Lexical Analysis

Example – Programming Languages

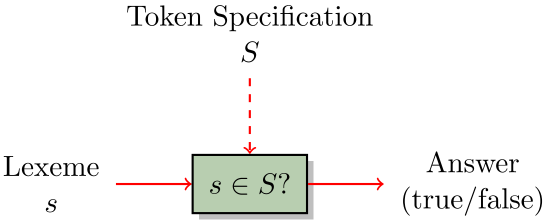
int	main	(	)	{	printf	(	Hello World!	)	;	}
KW int	ID main	(	)	{	ID printf	(	SL Hello World!	)	;	}

Terminology

- **Token classes/Tokens.** e.g. KW, ID, (, } etc.
- **Lexemes.** e.g. `"int"`, `"main"`, `"("`, `"}"` etc.

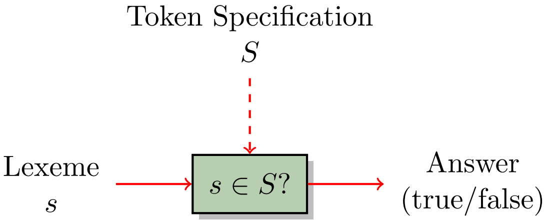
Specification of Token Classes

Example



Specification of Token Classes

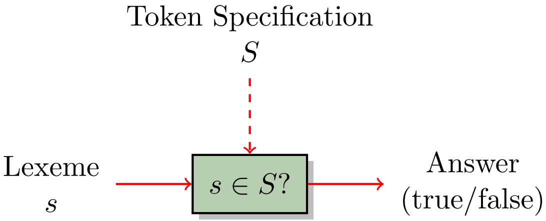
Example



Specification (S) of token classes

Specification of Token Classes

Example



Specification (S) of token classes – *regular expressions*

Regular Expressions

Example – identifier

- **English:**
  - Must comprise of only alphabetic character (α)

Regular Expressions

Example – identifier

- **English:**
  - Must comprise of only alphabetic character (α)
- **Regular Expression:** $\alpha+$

Regular Expressions

Example – identifier

- **English:**
  - Must start with an alphabetic character (α)
  - Subsequent letters can be alphabetic or numeric (N)

Regular Expressions

Example – identifier

- **English:**
  - Must start with an alphabetic character (α)
  - Subsequent letters can be alphabetic or numeric (N)
- **Regular Expression:** $\alpha(\alpha|N)*$

Regular Expressions

Rules of Construction

1. ϵ (empty symbol)
2. Let r and s be two regular expressions:

Expression	**Meaning**
$r\|s$	Choice
rs	Concatenation
$r\star$	Zero or more
$r+$	One or more
$r?$	Zero or one

Regular Expressions

Rules of Construction

1. ϵ (empty symbol)
2. Let r and s be two regular expressions:

Expression	Meaning	Example	Instance
$r\|s$	Choice	$\alpha\|N$	a, b, ..., 1, 2, ...
rs	Concatenation	rs	a1, b1, ..., a2, ...
$r\star$	Zero or more	$\alpha\star$	ϵ, a, ab, aaa, ...
$r+$	One or more	$N+$	1, 11, 12, ...
$r?$	Zero or one	$N?$	ϵ, 1, 2, ...

Regular Expressions

Example – Identifier

- **English:**
  1. Must start with an alphabetic character
  2. Subsequently, may have either an alphabetic character (α), a numeric character (N), separated by zero or more underscores ('_').

Regular Expressions

Example – Identifier

- **English:**
  1. Must start with an alphabetic character
  2. Subsequently, may have either an alphabetic character (α), a numeric character (N), separated by zero or more underscores ('_').
- **Regular Expression:** $\alpha\{[(\alpha|N)(\text{'\_'}\star)]\star(\alpha|N)\}?$

Regular Expressions

Activity

Regular expression for floating point numbers in C programming language

Implementation of Lexical Analysis

Finite State Automata

Example

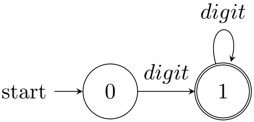
NUMBER:

Implementation of Lexical Analysis

Finite State Automata

Example

NUMBER:



return $NUMBER$

Regular Expressions

Regular Expressions and Finite State Automata

- Regular expressions are easy to use for specifying token classes but ...

Regular Expressions

Regular Expressions and Finite State Automata

- Regular expressions are easy to use for specifying token classes but ... hard to implement.

Regular Expressions

Regular Expressions and Finite State Automata

- Regular expressions are easy to use for specifying token classes but ... hard to implement.
- Finite state machines are easy to implement but ...

Regular Expressions

Regular Expressions and Finite State Automata

- Regular expressions are easy to use for specifying token classes but ... hard to implement.
- Finite state machines are easy to implement but ... what about their correspondence with token classes?

Regular Expressions

Regular Expressions and Finite State Automata

- Regular expressions are easy to use for specifying token classes but ... hard to implement.
- Finite state machines are easy to implement but ... what about their correspondence with token classes?
- Equivalent
- Let R be the set of all regular expressions, and let F be the set of all finite state automata.

$$\forall r \in R, \exists f \in F \text{ such that } L(r) = L(f)$$

$$\forall f \in F, \exists r \in R, \text{ such that } L(r) = L(f)$$

Regular Expressions

Regular Expressions and Finite State Automata

- Regular expressions are easy to use for specifying token classes but ... hard to implement.
- Finite state machines are easy to implement but ... what about their correspondence with token classes?
- Equivalent
- Let R be the set of all regular expressions, and let F be the set of all finite state automata.

$$\forall r \in R, \exists f \in F \text{ such that } L(r) = L(f)$$

$$\forall f \in F, \exists r \in R, \text{ such that } L(r) = L(f)$$

Lexical Analysis

Overview

- Finite state automata
- Implementation of finite state automata
- Implementation of lexical analysers: manual and automated generation