ΕΠΛ323 - Θεωρία και Πρακτική Μεταγλωττιστών

Lecture 3a

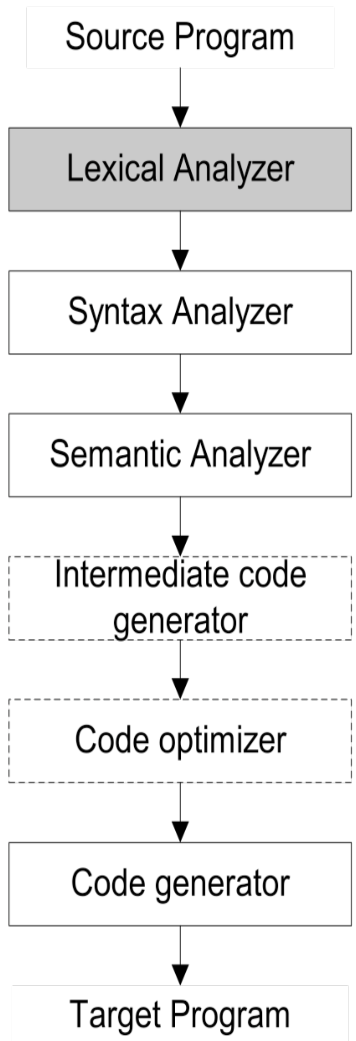
Lexical Analysis

Elias Athanasopoulos
eliasathan@cs.ucy.ac.cy

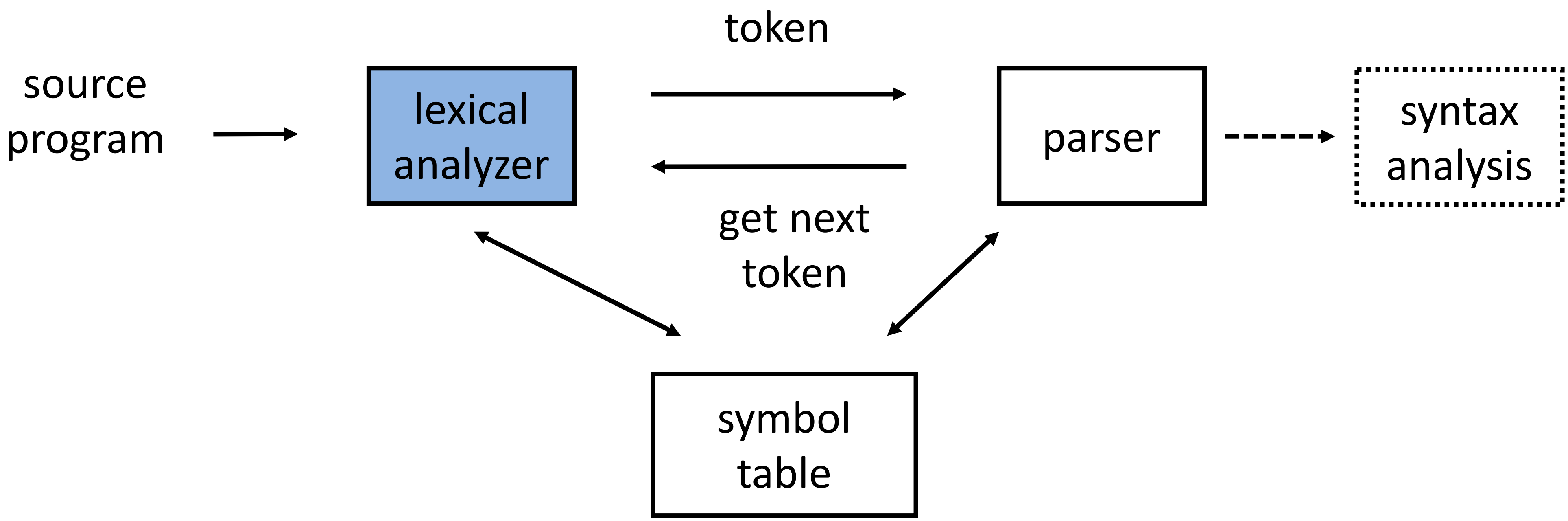
Lexical Analysis

Λεκτική Ανάλυση

- Definitions
  - Tokens, patterns, lexemes
- Regular Expressions
- Transition Diagrams
- Finite Automata
  - Non-deterministic (NFA)
  - Deterministic (DFA)



The Role of Lexical Analysis



Lexical Analysis Properties

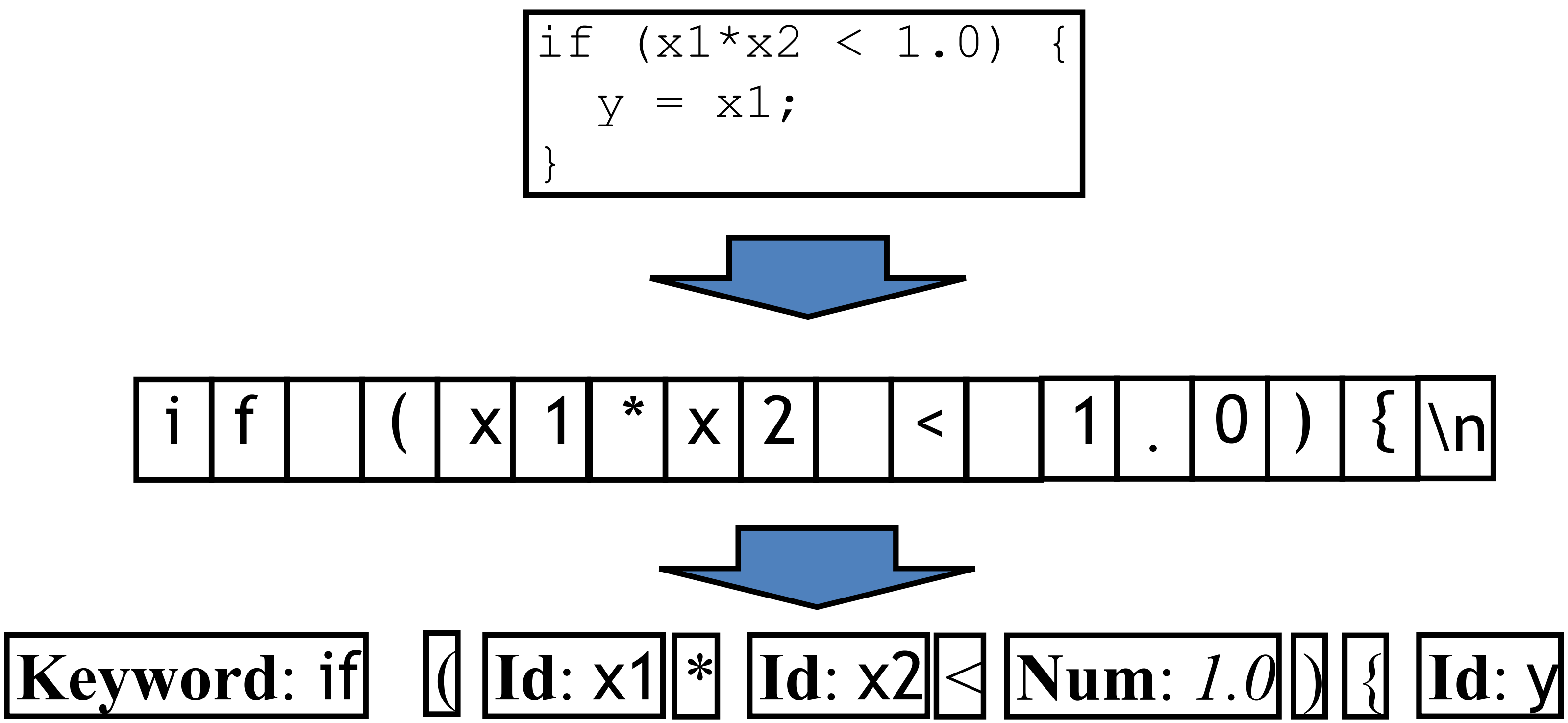
- First phase of the compiler
- Reads the input characters (source program)
  - Heavy I/O, many techniques for speeding up the process
  - De-beautifies the source (strips comments, white-space)
  - Keeps state for error-reporting (line numbers)
  - Sometimes implements the pre-processor
- Produces a sequence of tokens that the parser uses for *syntax analysis*
  - Separation of lexical-syntax analysis is mostly for a clean design

Lexical-Syntax Analysis Separation

- Simpler design
  - Syntax analysis without comments and white-space is simpler
- Efficiency
  - Specialized buffering for reading the source program
- Portability
  - Handling of special characters/alphabets is isolated

How it works?

- Convert source code stream to a series of **tokens**



Tokens

Διακριτικά

- Identifiers (*αναγνωριστικά*)
  - x, y11, elsex, _i00
- Keywords (*δεσμευμένες λέξεις*)
  - if, else, while, break
- Constants (*σταθερές*)
  - 2, 1000, -500, 5L, 2.0, 0.00020, .02, 1., 1e5
- Operators and symbols (*τελεστές ή σύμβολα*)
  - \+ * { } ++ < << [ ] >=
- Strings (*αλφαριθμητικά*):
  - “x”, “He said, \“Are you?\””
- Comments (*σχόλια*)
  - /** comment **/

Challenges

- Several different formats
  - `2.e0, 20.e-01, 2.0000`
- Context is significant
  - Lexical analyzer has a local view

```
if (x == f(x))
fi (x == f(x))
```

- Keyword-less languages (e.g., PL/I)
  - `IF THEN THEN THEN = ELSE; ELSE ELSE = THEN;`

Treating whitespace

- Whitespace is primarily added for readability of the source code
- In some languages whitespace is not significant and can make things complicated

```
DO 5 I = 1.25
(means DO5I = 1.25)
DO 5 I = 1,25
(means a loop from 1 to 25)
```

Tokens – Patterns – Lexemes

Δικριτικά – Πρότυπα – Λέξεις

- Tokens (*διακριτικά*)
  - Elements of the language (identifiers, keywords,etc.)
- Pattern (*πρότυπο*)
  - A rule that if applied to a set of strings (or text) generates the same token
- Lexeme (*λέξη*)
  - A sequence of characters in the source program that is matched by the pattern for a token

Example

```
const pi = 3.1456;
```

The substring **`pi`** is a lexeme for the token “identifier”

Examples of tokens

Token	Sample Lexemes	Pattern (informal)
const	const	const
if	if	if
relation	<, <=, =, <>, >, >=	< or <= or = or <> or > or >=
id	pi, count, D2	letter followed by letters or digits
num	3.141659, 0, 6.03E23	any numeric constant
literal	“core dumped”	any characters between “ and “ except “

Attributes for Tokens

`E = M *C ** 2`

<**id**, pointer to symbol-table entry for E>

<**assign_op**, >

<**id**, pointer to symbol-table enry for M>

<**mult_op**, >

<**id**, pointer to symbol-table entry for C>

<**exp_op**, >

<**num**, integer value 2>

How we match tokens?

SPECIFICATION OF TOKENS

Definitions

- Alphabet (*αλφάβητο*)
  - Finite set of symbols
  - E.g., {0,1} is the binary alphabet
- String (*συμβολοσειρά*)
  - Finite set of symbols drawn from the alphabet
  - ε is the empty string
  - |x| is the size of string, **banana** is a string of size 6
- Language (*γλώσσα*)
  - Any set of strings constructed using an alphabet
  - E.g., {ε}, ∅, {01, 00, 11, 10}

String operations

prefix of *s*	A string obtained by removing zero ore more trailing symbols of string *s*; e.g., **ban** is a prefix of **banana**
suffix of *s*	A string formed by deleting zero ore more of the leading symbols of *s*; e.g., **nana** is a suffix of **banana**
substring of *s*	A string obtained by deleting a prefix and a suffix form *s*; e.g., **nan** is a substring of **banana**
proper prefix, *suffix*, or *substring* of *s*	Any nonempty string *x* that is, respectively, a prefix, suffix, or substring of *s* such that $s \neq x$
subsequence of *s*	Any string formed by deleting ero ore more not necessarily contiguous symbols from *s*; e.g., **baaa** is a subsequence of **banana**

Operations on Languages

- Concatenation (*συνένωση ή παράθεση*)
- Union (*ένωση*)
- Closure (*κλείσιμο*)

Concatenation

Συνένωση

- Assume languages, L and M, their concatenation, L$\cap$M, or LM is
  - LM = { *st* | *s* $\in$ L and *t* $\in$ M }
  - s, t are **strings**

Example

L = {A, B, C, …, Z}

M = {0,1,2, …, 9}

LM = {A0, A1, …, B0, B1, …}

Exponentiation

Ύψωση σε δύναμη

- $L^0 = \{\varepsilon\}$

- $L^k = \{s_1\ s_2 \dots s_k \mid s_i$ is in $\in L$, $i=1,..,k\}$

Example

$L = \{A, B, C, \dots, Z\}$

$L^2 = \{AA, AB, \dots, BA, BB, \dots\}$

Union

Ένωση

- Assume languages L and M. Their union, $L \cup M$, is
  - $L \cup M = \{ s \mid s \in L \text{ or } s \in M \}$
  - s is **string**

Example

$L=\{A, B, C, \ldots, Z\}$

$M=\{0,1,2,\ldots,9\}$

$L \cup M = \{A, B, C, \ldots, Z, 0,1,2,\ldots,9\}$

Closure

Κλείσιμο

- Kleene closure of *L*
  - *L** denotes “zero ore more concatenations of” *L*

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

- Positive closure of L
  - *L+* denotes “one ore more concatenations of” *L*

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Examples

L = {A, B, ..., Z, a, b, ... z}, i.e., all letters

D = {0, 1, ..., 9}, i.e., all digits

1. **$L \cup D$** is the set of letters and digits
2. **LD** is the set of strings consisting of a letter followed by a digit
3. **L^4** is the set of all four-letter strings
4. **L^*** is the set of all strings of letters, including the empty string
5. **$L(L \cup D)^*$** is the set of all strings of letters and digits beginning with a letter
6. **D^+** is the set of all strings of one or more digits

Regular Expressions

Κανονικές Εκφράσεις

- In Pascal, an identifier is a letter followed by zero or more letters
  - I.e., it is a member of the set $\mathbf{L(L \cup D)^*}$
- We use *regular expressions* to define such sets
  - **letter (letter | digit) ***
- Each regular expression *r* over an alphabet denotes a language *L(r)*

Rules

1. ε is a regular expression that denotes {ε}, i.e., the set containing the empty string
2. If *a* is a symbol in alphabet Σ then *a* is a regular expression that denotes {*a*}
   - *a* is used for the symbol, the string and the regular expression
3. Suppose *r* and *s* are regular expressions denoting the language *L(r)* and *L(s)*
   - (r)|(s) is a regular expression denoting $L(r) \cup L(s)$
   - (r)(s) is a regular expression denoting $L(r) \cap L(s)$
   - $(r)^*$ is a regular expression denoting $L(r)^*$
   - (r) is a regular expression denoting L(r)

Operator precedence

Προτεραιότητες

1. The unary operator * has the highest precedence and is left associative
2. Concatenation has the second highest precedence and is left associative
3. | has the lowest precedence and is left associative

(a) | ((b)(c))* is equivalent to *a|b*c*

Regular Expressions Algebra

$r \vert s = s \vert r$	$\vert$ is commutative
$r \vert (s \vert t) = (r \vert s) \vert t$	$\vert$ is associative
$(rs)t = r(st)$	concatenation is associative
$r(s \vert t) = rs \vert rt$ $(s \vert t)r = sr \vert tr$	concatenation distributes over $\vert$
$\varepsilon r = r$ $r\varepsilon = r$	ε is the identify element of concatenation
$r^* = (r \vert \varepsilon)^*$	relation between * and ε
$r^{**} = r^*$	* is idempotent

Shorthands

- +: “one or more instances of” r^+ is equal to $(L(r))^+$
- ?: “zero or one instance of” r? equal to r|ε
- [a-z]: {a, b, …,z}, equal to a|b|c|d…|z
- [^a-z]: not in set {a, b, …,z}

Regular definitions

- A frequently used regular expression can be named for delivering additional regular expressions

Pascal Identifiers (e.g., x1, y, velocity100, etc.)

```
letter → A | B | . . . | Z | a | b | . . . | z
digit  → 0 | 1 | . . . | 9
id     → letter (letter|digit)*
```

Example 1

- Unsigned numbers in Pascal
  - 5280, 39.37, 6.336E4, 1.894E-4

```
   digit → 0 | 1 | . . . | 9
  digits → digit digit*
opt_frac → . digits | ε
 opt_exp → (E(+|-|ε)digits)|ε
     num → digits opt_frac opt_exp
```

Example 2

- Unsigned numbers in Pascal
  - 5280, 39.37, 6.336E4, 1.894E-4

```
    digit → 0 | 1 | . . . | 9
   digits → digit+
 opt_frac → (. digits)?
  opt_exp → (E(+|-)?digits)?
      num → digits opt_frac opt_exp
```