Lexical Analysis

Scanning of the characters and **Checking of the language words and symbols** used in the syntactic structures
(identifiers, numerals, keywords, separators, delimiters, ...)

The **Lexical Languages** are, in general, quite simple.

- They are:

  Regular Languages: $\mathfrak{R}$
- They can be defined using:

  Regular Expressions

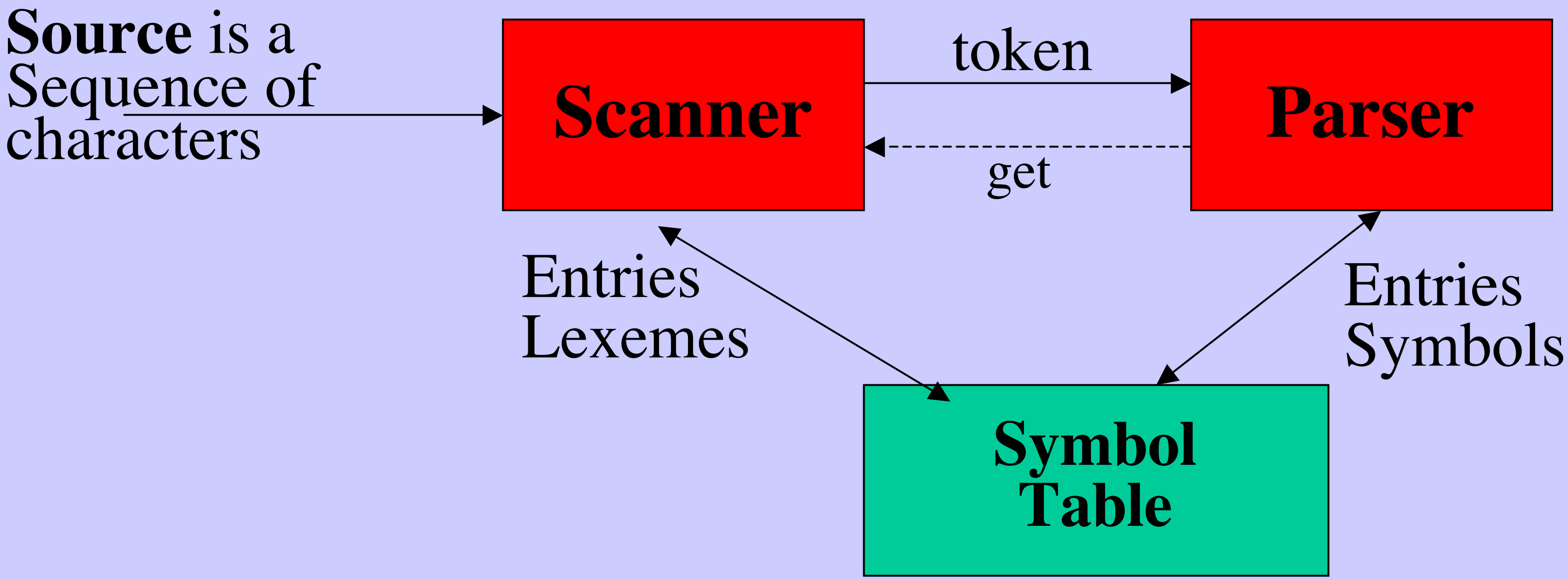
  Regular (or Linear) Grammars

  Transition Diagrams
- They can be analysed using:

  N/D Automata

Where and How in a Compiler:

One Pass Structure - First Phase



Lexical Analysis (**Scanner**) is driven from
Syntactic Analysis (**Parser**) which is asking for next token

Definitions: Token, Lexeme, Pattern

Token = a Lexical *Category*

lexeme = a *Value* of a Category

pattern = Lexical *rules* defining Tokens and Lexemes

Exemple

source: const pigreco = 3.1416
scanning: **const id rel num**

Lexeme:

Where source symbols are stored ?

- Source Symbols are maintained in Information Colums

- Scanner does not generate token but: **<token**,attribute>

  <const, **>** **<id**, k **>** **<rel**, **=** **>** **<num**, k+1 **>**

	lexeme	
k	pigreco	
k+1	3.1416	

Information Colums

SymbolTable

Patterns:

How Tokens are *unambiguously* defined ?

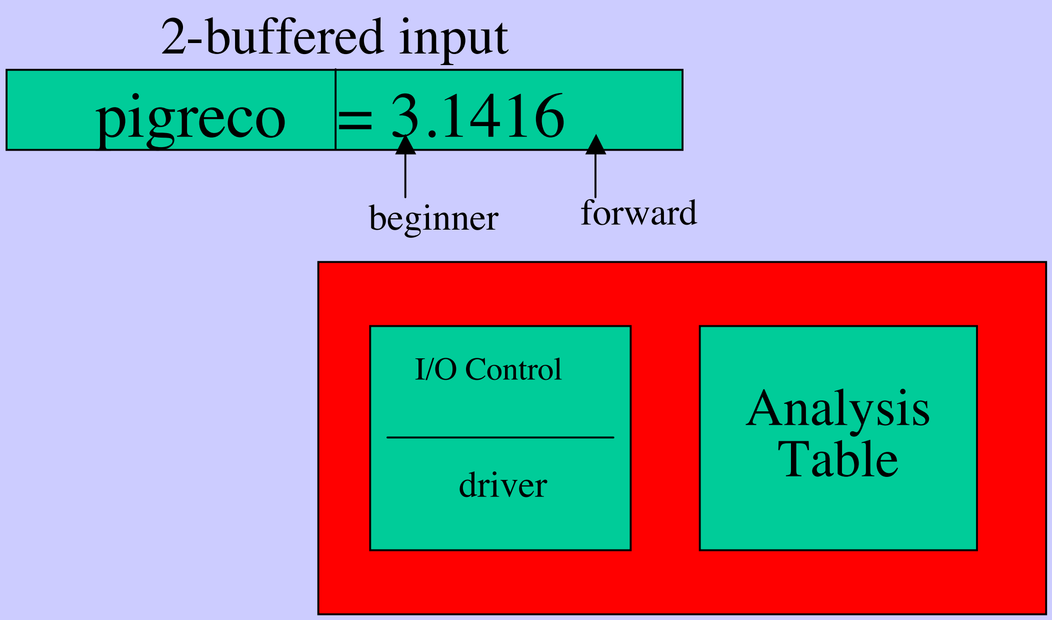
- **Patterns** may be defined using many different Formalisms:
  - Regular Expressions
  - Regular Grammars
  - Transition Diagrams

- Though equivalent, such Formalisms:
  - Have different expressiveness.
  - Hence the use of some is easier, clearer, and neater than that of the other.
  - Have different working frameworks.
  - Hence, the mechanization of the underlying analysis process is different for the different formalisms

Scanner Generators

- **Today scanners are automatically generated using Regular Grammars, Regular Expressions, DFAutomata, in this order;**

- **Meaning preserving transformations:**
  - **allow to pass from one formulation to the next one, and**
  - **result into the structure below:**



Regular Grammars

(but before) Regular Expressions: E_Σ

The Syntax

Regular Expressions $\mathbf{E}_\Sigma$ on an alphabet Σ are:

The minimum set recursively defined by:

1. $\boldsymbol{\varepsilon} \in \mathbf{E}_\Sigma$
2. $\mathbf{a} \in E_\Sigma$, $\quad \forall a \in \Sigma$
3. $\mathbf{e_1 . e_2} \in \mathbf{E}_\Sigma$, $\quad \forall e_1, e_2 \in E_\Sigma$
4. $\mathbf{e_1 | e_2} \in \mathbf{E}_\Sigma$, $\quad \forall e_1, e_2 \in E_\Sigma$
5. $\mathbf{e^*} \in \mathbf{E}_\Sigma$, $\quad \forall e \in E_\Sigma$
6. $\mathbf{e^i} \in \mathbf{E}_\Sigma$, $\quad \forall e \in E_\Sigma$

Regular Expressions:
The Meaning

The meaning of E_Σ is $\mathfrak{R} \subseteq 2^{\Sigma^*}$

$\Sigma = \{a,b\}$, $\Sigma^i = \Sigma \times . \Sigma^{i-1}$, $\Sigma^0 = \{\lambda\}$, $\Sigma^2 = \{aa, ab, bb, ba\}$

Exponentiation $\Sigma^* = \cup_{i \in N} \Sigma^i = \{\lambda, a, b, aa, ab, bb, ba, aaa, \ldots\}$

Power Set $2^{\Sigma^*} = \{u \mid u \subseteq \Sigma^*\} = \{\{\lambda\},\{a\},\{b\},\ldots,\{\lambda,a\},\ldots,\{\lambda,a,b,ab\},\ldots,\}$

$[\_]: E_\Sigma \to \mathfrak{R}$

1. $[\varepsilon] = \{\lambda\}$ - singleton with nullary string
2. $[a] = \{a\}$ - singleton with 1 char string
3. $[e_1.e_2] = \{uv \mid u \in [e1], v \in [e2]\} = [e1] \times . [e2]$ - juxtaposition of the product set
4. $[e_1 | e_2] = \{u \mid u \in [e1] \cup [e2]\}$ - set union
5. $[e^*] = \{u \mid u \in \cup_{i \in N} [e]^i\}$ - exponentiation set
6. $[e^i] = \{u \mid u \in [e]^i\}$ *shortening* - i^{th}-power

Regular Expressions:
Examples

0 | 1 | … | 9

[0 | 1 | … | 9] = {0,1,…,9}

(0 | 1 | … | 9)*

[(0 | 1 | … | 9)*] = {0,1,…,9, 00, 01,…, 3808, ...}

(1 | … | 9).(0 | 1 | … | 9)*

[(1 | … | 9).(0 | 1 | … | 9)*] = {1,…,9, 10, 11,…, 3808, ...}

Usually, dot notation in concatenation operator is omitte. Then:

(1 | … | 9)(0 | 1 | … | 9)* is written instead of

Grammars on Σ

$\mathbf{G = < V, \Sigma, s \in V, P >}$

- **V** *non-terminal set* (Tokens and auxiliarys categories)
- $\mathbf{\Sigma}$ *terminal set*
- **s** *initial symbol* (of V)
- $\mathbf{P} \subseteq V \times E_{\Sigma \cup V}$ *production set*

Grammars

When it is a Regular Grammar

G is Regular, if in addition:

- V has a complete ordering < such that:
- $\forall$ v::=e $\in$ P, $\quad \mathbf{e} \in \mathbf{E}_{\Sigma \cup \{\mathbf{v}' < \mathbf{v}\}}$

Grammar

Example: Relations between grammars, expressions

The use of a regular grammar G in a lexic for numbers

num::= simple | fract | exp
simple::= digit digit*
fract::= simple.simple
exp::= fract E simple |
 fract E (+|-) simple
digit::= 0 | 1 | … | 9

How proving that G is regular?:

- Look for a complete ordering < on V
- Prove that < satisfies the second property

Only grammar productions are shown:

- What is Σ ?
- And, the other components of G ?
- Easily inferable: V (look left), s (look top)

How proving that G defines the language we are looking for?:

- Another Story Begins.

Grammars - Meaning L(G): Equations on Languages

Meaning associates to:

- **Each** non-terminal, V, **One** language, $L(V) \in \mathbf{2}^{\Sigma^*}$.
- **Each** production, **v::=e**, **One** equation, $\mathbf{L(v)} = [\mathbf{e}]$

see below on how do it

$e \in \mathbf{E}_\Sigma$ ======> $[\mathbf{e}] \in \mathbf{2}^{\Sigma^*}$

v::=e ======> $\mathbf{L(v)} = [e] \in (V \times \mathbf{2}^{\Sigma^*})$

$\{v_1::=e_1, \ldots, v_k::=e_k\}$ ======>

$\{\underline{L}(v_1), \ldots, \underline{L}(v_k) \mid \forall i, \underline{L}(v_i) \in \mathbf{2}^{\Sigma^*}$ *and* $\underline{L}(v_i) \equiv [[\underline{L}(vj)/vj \mid vj<vi]e_i]\}$

$G = \langle V, \Sigma, s \in V, P \rangle$ ======> $\underline{L}(G) = \underline{L}(s)$

Meaning of (Regular) Grammars: An Example of Computation

n::= s | f | e
s::= d d*
f::= s.s
e::= f E s |
      f E (+|-) s
d::= 0 | 1 | … | 9

L(0|1|…|9) = {0,…,9} $\in (V \times \mathbf{2}^{\Sigma^*})$

{$\underline{L}$(d)={0,…,9},
 $\underline{L}$(s)=L([{0,…,9}/d] d d*)
   ={0,…,9}{0,…,9}* ={0,…,9}$^+$
 $\underline{L}$(e)= …
   ={u.vEw | u,v,w ∈ {0,…,9}$^+$}

L(n) = {0,…,9}$^+$ ∪
   {u.v | u,v ∈ {0,…,9}$^+$} ∪
   {u.vEw | u,v,w ∈ {0,…,9}$^+$}
…

Regular Grammars: Meaning

Compare the formulation in the previous slide,

$\{v_1::=e_1,\ldots, v_k::=e_k\}$ =======>

$\{\underline{\mathbf{L}}(v_1),\ldots,\underline{\mathbf{L}}(v_k) \mid \forall i, \underline{\mathbf{L}}(v_i) \in \mathbf{2}^{\Sigma^*}$ *and* $\underline{\mathbf{L}}(v_i) \equiv [\![[\underline{\mathbf{L}}(vj)/vj \mid vj<vi]e_i]\!]\}$

with the the one below.

$\{v_1::=e_1,\ldots, v_k::=e_k\}$ =======>

$\{\underline{\mathbf{L}}(v_1),\ldots,\underline{\mathbf{L}}(v_k) \mid \forall i, \underline{\mathbf{L}}(v_i) \in \mathbf{2}^{\Sigma^*}$ *and* $\underline{\mathbf{L}}(v_i) \equiv [\![[e_i^{1},\ldots,e_i^{ni}/v_i^{1},\ldots,v_i^{ni}]e_i]\!]\}$

where: 1) $\{v_i^{1},\ldots,v_i^{ni}\}=\{v_j \in V \mid v_j<v_i\}$;

2) $v_i^{1}<\ldots<v_i^{ni}$;

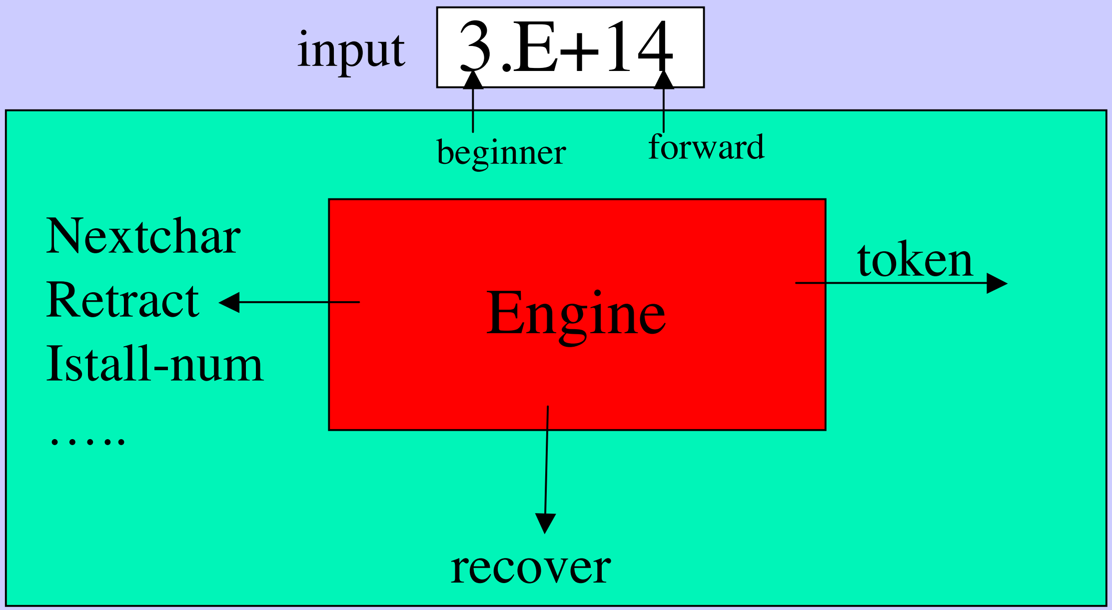
3) $[e_i^{1},\ldots,e_i^{ni}/v_i^{1},\ldots,v_i^{ni}]e_i = [e_i^{1},\ldots,e_i^{ni-1}/v_i^{1},\ldots,v_i^{ni-1}]([e_i^{ni}/ v_i^{ni}]e_i)$

Comment the differences.

Justify why the last one is more correct than the other.

How to Recognize the Lexic defined by a Grammar

n::= s | f | e
s::= d d*
f::= s.s
e::= f E s |
      f E (+|-) s
d::= 0 | 1 | … | 9



Scanners do it