Course 2

Chapter 1. Scanning

INPUT: source program
OUTPUT: PIF + ST

```
Algorithm Scanning v1
While (not(eof)) do
    detect(token);
    classify(token);
    codify(token);
End_while
```

Detect

Take notes!

I am a student.I   am Simona

- Separators => ***Remark 1)***

if (x==y) {x=y+2}

- Look-ahead => ***Remark 2)***

Classify

- Classes of tokens:
  - Identifiers
  - Constants
  - Reserved words (keywords)
  - Separators
  - Operators

- If a token can NOT be classified => LEXICAL ERROR

Codify

- May be codification table

OR

code for identifiers and constants

- Identifier, constant => Symbol Table (ST)
- PIF = Program Internal Form = array of pairs
- pairs (token, position in ST)

identifier, constant

Algorithm Scanning v2

```
While (not(eof)) do
     detect(token);
     if token is reserved word OR operator OR separator
           then genPIF(token, 0)
           else
          if token is identifier OR constant
                then index = pos(token, ST);
                      genPIF(token, index)
                else  message "Lexical error"
          endif
      endif
endwhile
```

a=a+b

FIP

(id,1)
(=,0)
(id,1)
(+,0)
(id,2)

ST

1 a
2 b

Remarks:

- genPIF = adds a pair (token, position) to PIF

- Pos(`token`,ST) – searches *`token`* in symbol table S*T*; if found then return position; if not found insert in SR and return position
- Order of classification (reserved word, then identifier)
- If-then-else imbricate => detect error if a token cannot be classified

Algorithm Scanning v2

```
While (not(eof)) do
    detect(token);
    if token is reserved word OR operator OR separator
        then genPIF(token, 0)
        else
        if token is identifier OR constant
            then index = pos(token, ST);
                genPIF(token_type, index)
            else  message "Lexical error"
        endif
    endif
endwhile
```

Remarks:

- Also comments are eliminated
- Most important operations: SEARCH and INSERT

Symbol Table

Definition *= contains all information collected during compiling regarding the* <u>*symbolic names*</u> *from the source program*

identifiers, constants, etc.

Variants:

- Unique symbol table – contains all symbolic names
- distinct symbol tables: IT (identifiers table) + CT (constants table)

ST organization

Remark: search and insert

1.	Unsorted table – in order of detection in source code	O(n)
2.	Sorted table: alphabetic (numeric)	O(lg n)
3.	Binary search tree (balanced)	O(lg n)
4.	Hash table	O(1)

Hash table

- K = set of keys (symbolic names)
- A = set of positions ($|A| = m$; m –prime number)

$h : K \rightarrow A$

$$h(k) = (\text{val}(k) \bmod m) + 1$$

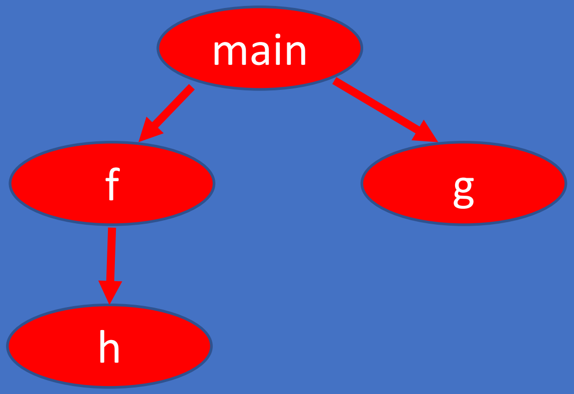
- Conflicts: $k_1 \neq k_2$, $h(k_1) = h(k_2)$

Toy hash function to use at lab:
Sum of ASCII codes of chars

Visibility domain (scope)

- Each scope – separate ST
- Structure -> inclusion tree

Hierachical structure of STs:



Example:

```
Int main(){
… int a;

void f()
  {float a;
     … int h() {…}
   }
 …
void g()
  {char a;
     …
  }
}
```

Formal Languages

- basic notions-

Examples of languages

- natural (ex. English, Romanian)
- programming (ex. C,C++, Java, Python)
- formal

A formal language is a set
Ex.:
$L = \{a^n b^n | \ n>0\}$ $L = \{ab, aabb, aaabbb, \ldots\}$
$L' = \{01^n | n>=0\}$ $L' = \{0, 01, 011, \ldots\}$

Example

a boy has a dog

S→PV
P→ *a* N
N→ *boy* or N→ *dog*
(N→*boy*|*dog*)
V → QC
Q → *has*
C → BN

B → *a*

- A→ α = **rule**
- S,P,V,N,Q,C,B = **nonterminal symbols**
- *a, boy,dog,has* = **terminal symbols**

Remarks

1. Sentence = word, sequence (contains only terminal symbols) ; denoted w.
2. S⇒PV⇒a NV⇒a NQC⇒a N has C - sentential form

   In general : $w=a_1a_2...a_n$

3. The rule guarantees syntactical correctness, but not the semantical correctness (*A dog has a boy*)

Grammar

- ***Definition***: A (formal) **grammar** is a 4-tuple: $G=(N,\Sigma,P,S)$ with the following meanings:
  - N – set of <u>nonterminal</u> symbols and $|N| < \infty$
  - Σ - set of <u>terminal</u> symbols (alphabet) and $|\Sigma|<\infty$
  - P – finite set of <u>productions</u> (rules), with the propriety:

    $P\subseteq(N\cup\Sigma)^* N(N\cup\Sigma)^* \mathbf{x} (N\cup\Sigma)^*$
  - $S\in N$ – <u>start symbol</u> /axiom

Remarks :

1. $(\alpha,\beta)\in P$ is a production denoted $\alpha\rightarrow\beta$
2. $N \cap \Sigma = \emptyset$

A^* = transitive and reflexive closure = {a,aa,aaa,...} $\{a^0\}$

A = {a}

A+ = {a,aa,aaa,...}

$X^0 =\varepsilon$

Binary relations defined on $(N \cup \Sigma)^*$

- ***Direct derivation***
  $\alpha \Rightarrow \beta$, $\alpha,\beta \in (N \cup \Sigma)^*$ ***if*** $\alpha = x1xy1$, $\beta = x1yy1$ ***and*** $x \rightarrow y \in P$
  (x is transformed in y)
- ***k derivation***

$\alpha \overset{k}{\Rightarrow} \beta$, $\alpha,\beta \in (N \cup \Sigma)^*$

sequence of k direct derivations $\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{k-1} \Rightarrow \beta$, $\alpha,\alpha_1,\alpha_2,\ldots\alpha_{k-1},\beta \in (N \cup \Sigma)^*$

- ***+ derivation***

$\alpha \overset{+}{\Rightarrow} \beta$ ***if*** $\exists\ k>0$ ***such that*** $\alpha \overset{k}{\Rightarrow} \beta$ (there exists at least one direct derivation)

- **** derivation***

$\alpha \overset{*}{\Rightarrow} \beta$ ***if*** $\exists\ k\geq 0$ ***such that*** $\alpha \overset{k}{\Rightarrow} \beta$ namely, $\alpha \overset{*}{\Rightarrow} \beta \Leftrightarrow \alpha \overset{+}{\Rightarrow} \beta$ ***OR*** $\alpha \overset{0}{\Rightarrow} \beta$ $(\alpha=\beta)$

Definition: ***Language generated*** by a grammar G=(N,Σ,P,S) is:

$$L(G)=\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

L1 = {a,b,aa}
L2 = {c,d,cd}
L1L2 = {ac,ad,acd,bc,bd,bcd,aac,aad,aacd}

Remarks:

1. $S \overset{*}{\Rightarrow} \alpha, \alpha \in (N \cup \Sigma)^*$ = sentential form
   $S \overset{*}{\Rightarrow} w, w \in \Sigma^*$ = word / sequence
2. Operations defined for languages (sets) :
   $L1 \cup L2$, $L1 \cap L2$, L1-L2 , $\overline{L}$ (complement) , $L^+=\bigcup_{k>0} L^k$, $L^*=\bigcup_{k\geq 0} L^k$
   Concatenation: $L=L_1L_2 = \{w_1w_2 \mid w_1 \in L_1 , w_2 \in L_2\}$
3. $|w|=0$ (empty word - denoted ε)

Definition: Two grammar G_1 and G_2 are equivalent if they generate the same language

$$L(G_1)=L(G_2)$$

Chomsky hierarchy(based on form $\alpha \to \beta \in P$)

- type 0 : no restriction
- type 1 : context dependent grammar ($x_1 A y_1 \to x_1 \gamma y_1$)
- type 2 : context free grammar ($A \to \alpha \in P$,where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$)
- type 3 : regular grammar ($A \to aB|a \in P$)

Remark :

type 3 $\subseteq$ type 2 $\subseteq$ type 1 $\subseteq$ type 0

Regular grammars

- $G = (N, \Sigma, P, S)$ right linear grammar if

  $\forall p \in P$: $A \rightarrow aB$ or $A \rightarrow b$, where $A,B \in N$ and $a,b \in \Sigma$

- $G = (N, \Sigma, P, S)$ regular grammar if
  - G is right linear grammar

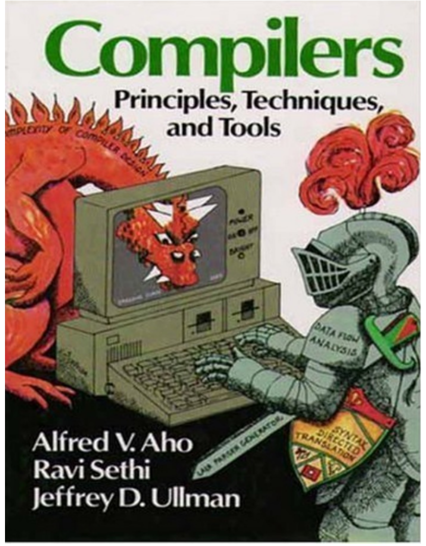
  and
  - $A \rightarrow \varepsilon \notin P$, with the exception that $S \rightarrow \varepsilon \in P$, in which case S does not appear in the rhs (right hand side) of any other production

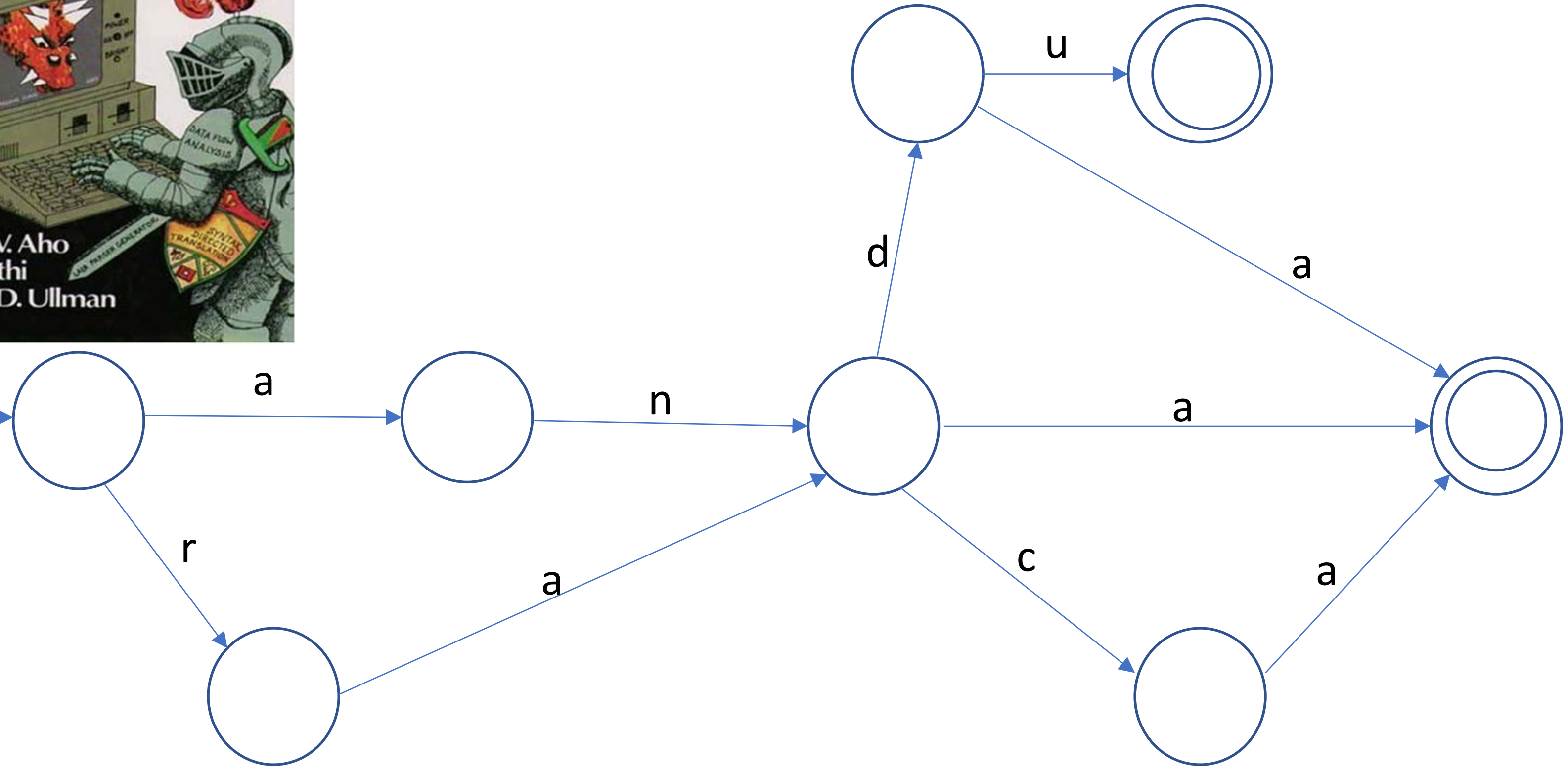
S->aA|ε; A-> a reg
S->aS|aA; A->bS|b reg
S->aA; A->aA|ε NOT reg
S->aA|ε; A->aS NOT reg

- $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$ - right linear language

Notations

- A,B,C,... – nonterminal symbols
- $S \in N$ – start symbol
- $a,b,c,... \in \Sigma$ – terminal symbol
- $\alpha,\beta,\gamma \in (N \cup \Sigma)^*$ - sentential forms
- ε – empty word
- $x,y,z,w \in \Sigma^*$ - words
- $X,Y,U,... \in (N \cup \Sigma)$ – grammar symbols (nonterminal or terminal)





Problem: The door to the tower is closed by the Red Dragon, using a complicated machinery. Prince Charming has managed to steal the plans and is asking for your help. Can you help him determining all the person names that can unlock the door