

How to build Scanner for a Lexics L

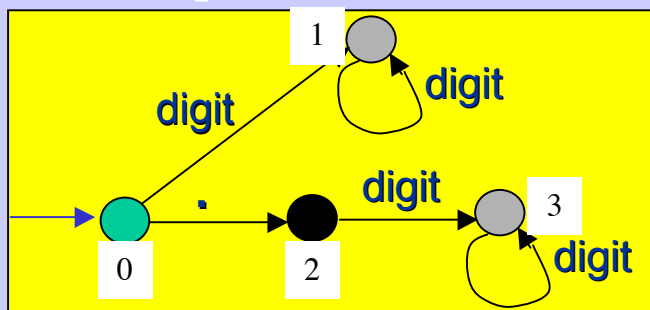
Other Techniques

- *Transition Diagrams*
- *Minimization*
- *Dotted Automata*

Transition Diagrams on $\Sigma - 1$

- Are used in *by hand* constructions, of recognizers.
- Lexics must be *quite simple* to guarantee correctness by eye.
- Consist of directed, labeled, possibly, annotated, **Graphs**,
- And in a **Codification Code** of the graph.

Graph



Codification Code

- To each *node* one *case statement* is generated
 - case 0: **if** (c==digit) state = 1;
 else if (c==dot) state = 2;
 else state = fail(0);
 - case 2: **if** (c==digit) state = 3;
 else state = fail(2);
 - case 1: **if** (c==digit) state = 1;
 else {retract(1);
 install-num();
 return(S);}
 - case 3: **if** (c==digit) state = 3;
 else {retract(1);
 install-num();
 return(S);}

Transition Diagrams on Σ - 2

- Diagrams/Codes can be composed to construct a scanner

```
int state = 0;  
int lexical-value;
```

```
token nexttoken()  
{  
  while(1) {  
    switch (state) {  
      case 0: ...  
      case 1: ...  
      case 2: ...  
      case 3: ...}  
    }  
  }  
}
```

A list of auxiliary structures and functions that have to be implemented

forward: input pointer

c: character to be read

nextchar(): go to next character

retract(k): go k characters back

state: last traversed state

Start: initial state

install-num(): update symbol table

fail(state)-recover(state): management of the failure conditions

digit, dot, ... : predicates for special characters

I/O: interface controller

...

Dotted Automata:

From Regular Expressions To DFA

- **Dotted Automata: How obtain Deterministic Automata directly from Regular Expressions**
- Let e be the input RE. It results an output DFA, A_e such that $L(e)=L(A_e)$
 - A_e **Initial State:** $I_0=C_s(\{\Delta e\})$,
 - Definition of the **Closure Function** C_s (by induction on e):
 - $C_s(S) = \bigcup_{\Delta\beta \in S} C(\Delta\beta)$
 - $C(\Delta\epsilon) = \{\Delta\}$;
 - $C(\Delta b\beta) = \{\Delta b\beta\}, \forall b \in \Sigma, \beta \in \Sigma^*$;
 - $C(\Delta e^*\beta) = C(\Delta e e^*\beta) \cup C(\Delta\beta)$;
 - $C(\Delta(e1|e2)\beta) = C(\Delta e1 \beta) \cup C(\Delta e2 \beta)$;
 - *additional cases can be added for optional additional RE operators*
 - Definition of the A_e 's *move*, denoted M , and of the remaining A_e states I_j :
 - $M(I_i, a) = I_j$ sse
 - $a \in \text{first}(I_i) = \{a \mid \Delta a\beta \in I_i, a \in \Sigma\}$
 - $I_j = C_s(\{\Delta\beta \mid \Delta a\beta \in I_i, a \in \Sigma\})$
 - **Final State:** $\{I_j \mid \Delta \in I_j\}$

- Induction is on the concatenation structure of e
- Δ is the dot symbol that is used to mark an item
- Δe is called dotted item
- Δ is the empty item

Apply it to expression

$e = ab^*b$

Compare the technique with the Thompson's one

Dotted Automata: From Regular Grammars To DFA

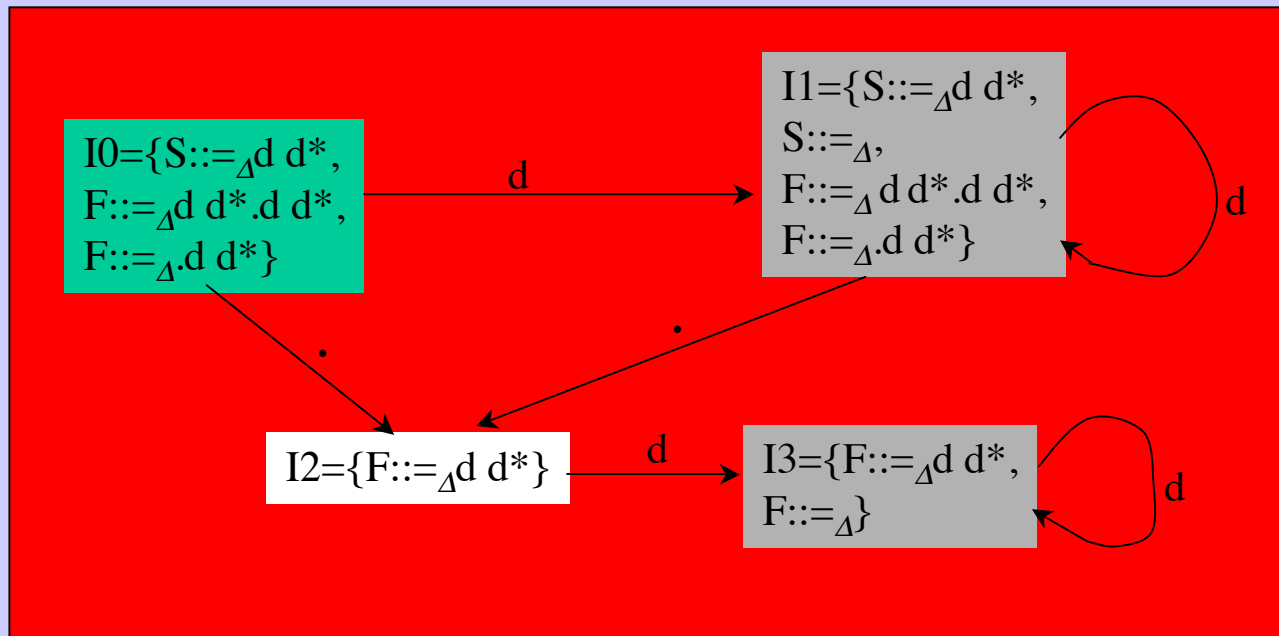
- **Dotted Automata: From RG to DFA**
- let $G = \langle S, \Sigma, s_0, P \rangle$ be Regular, and $P = \{s ::= e\}$. It return $A_G: L(G) = L(A_G)$.
 - **Initial State:** $I_0 = C(\{s ::=_{\Delta} e \mid s ::= e \in P, s \in S_0 \subseteq S\})$,
 S_0 contiene le categorie lessicali (token) di interesse
 - Definition of the **Closure Function** C_s (by induction on *productions*):
 - $C_s(\{s ::=_{\Delta} \beta \mid s \in S\}) = \bigcup_{s \in S} C(s ::=_{\Delta} \beta)$
 - $C(s ::=_{\Delta} \varepsilon) = \{s ::=_{\Delta} \varepsilon\}$;
 - $C(s ::=_{\Delta} b\beta) = \{s ::=_{\Delta} b\beta\}, \forall b \in \Sigma, \beta \in \Sigma^*$;
 - $C(s ::=_{\Delta} s'\beta) = C_s(\{s ::=_{\Delta} e\beta \mid s' ::= e \in P\}) \forall \beta \in \Sigma^*$;
 - $C(s ::=_{\Delta} e^*\beta) = C(s ::=_{\Delta} ee^*\beta) \cup C(s ::=_{\Delta} \beta)$;
 - $C(s ::=_{\Delta} (e1|e2)\beta) = C(s ::=_{\Delta} e1\beta) \cup C(s ::=_{\Delta} e2\beta)$;
 - *additional cases can be added for optional additional RE operators*
 - Definition of the A_G move, denoted M , and of the remaining A_G states I_j :
 - $M(I_i, a) = I_j$ sse
 - $a \in \text{first}(I_i) = \{a \mid s ::=_{\Delta} a\beta \in I_i, a \in \Sigma\}$
 - $I_j = C(\{s ::=_{\Delta} \beta \mid s ::=_{\Delta} a\beta \in I_i, a \in \Sigma\})$
 - **Final States:** $\{I_j \mid \exists sj \in S, sj ::=_{\Delta} \in I_j\}$

Apply it to Grammar

$S ::= A B$
$A ::= a$
$B ::= b^* b$

Automati di Items: un esempio

$S ::= \text{digit digit}^*$
 $F ::= \text{digit}^* . \text{digit digit}^*$



	digit	.
0	1	2
1	1	2
2	3	
3	3	

Minimal Automata

Equivalent States: \sim , \sim_{Π}

$$x \sim y \Leftrightarrow \forall \gamma \in \Sigma^*: (\gamma, x \Rightarrow^* \lambda, x' \Leftrightarrow \gamma, y \Rightarrow^* \lambda, y' \text{ s.t. } x' \sim y')$$

$$x \sim_{\Pi} y \Leftrightarrow \forall a \in \Sigma: (a, x \Rightarrow \lambda, x' \Leftrightarrow a, y \Rightarrow \lambda, y' \text{ s.t. } x' \sim_{\Pi} y')$$

Minimization Algorithm

Input: $A = \langle S, \Sigma, \text{move}, s, F \rangle$

Output: $M_A = \langle \Pi_S, \Sigma, \text{move}|_{\Pi_S}, s|_{\Pi_S}, F|_{\Pi_S} \rangle$

1. $\Pi = \{S \setminus F, F\}$;
2. $\Pi_v = \text{refine}_A(\Pi)$;
3. while $\Pi \neq \Pi_v$ do {
 1. $\Pi = \Pi_v$;
 2. $\Pi_v = \text{refine}_A(\Pi)$
4. $\Pi_S = \Pi$

Apply it to automaton A'

$\langle S = \{0,1,2\}, \Sigma = \{a,b\},$
 $\text{move} = \{ \langle \langle 0,a \rangle 1 \rangle, \langle \langle 1,b \rangle 2 \rangle, \langle \langle 2,b \rangle 2 \rangle \}$
 $s_0 = 0, F = \{2\} \rangle$

Operators: Let $\Pi = \{P_1, \dots, P_k\}$

$\text{refine}_A(\Pi) = \Pi_v = \{Q_1, \dots, Q_h\}$ s.t:

$\forall i \in [1..h]$:

- $\exists j \in [1..k]: Q_i = P_j$, or
- $\exists j \in [1..k], Q_i \subset P_j$ and
 $\forall x, y \in Q_i, x \sim_{\Pi_v} y$ and $\forall z \in P_j \setminus Q_i, x \not\sim_{\Pi_v} z$

Partition of S: $\Pi_S = \{P_1, \dots, P_k\}$

- copertura: $S = \bigcup_{j \in [1..k]} P_j$
- equivalenza: $P_i \cap_{i \neq j \in [1..k]} P_j = \emptyset$

Representatives $S|_{\Pi_S}$ on S: $\Pi_S = \{P_1, \dots, P_k\}$

- representative: $[P_i] \in P_i - \forall x \in P_i, [x] = [P_i]$
- $S|_{\Pi_S} = \{[x] | x \in S\}$
- $\text{move}|_{\Pi_S} = \{ \langle \langle [s], a \rangle, U|_{\Pi_S} \rangle | \langle \langle s, a \rangle U \rangle \in \text{move} \}$