A Driver for NFA (DFA):
How to remove Backtracking - 1

$$f(\gamma) = \begin{cases} \textit{accept} \ \textbf{if} \ \text{move1}^*(\gamma) \cap F \neq \{\} \\ \textit{noaccept} \ \textbf{otherwise} \end{cases}$$

move

	a	b	ε
0	{1}		
1		{1,2}	
2			

move1

	a	b	ε
0	{1}		
1		{1,2}	
2			
{1,2}		{1,2}	

How to remove Backtracking

Function *move1**: $\Sigma^* \to S$

How to computes transitions using Set of States instead of single States

Let $move1(S,c) = Clos(\cup_{s \in Clos(S)} \{move(s,c)\})$

remember: $Clos(S) = S \cup Clos(\cup_{s \in S}\ move(s,\varepsilon))$

Then: $\text{move1*}(c) = move1(\{s_0\},c)$

$\text{move1*}(c_1,\ldots,c_{n-1},c_n) = move1(\text{move1*}(c_1,\ldots,c_{n-1}),cn)$

Then, Decision Function f can be reformulated in

$$\text{sem}(A)(\gamma) = f(\gamma) = \begin{cases} \textit{accept} & \textbf{if}\ \text{move1*}(\gamma) \in F \\ \textit{noaccept} & \textbf{if}\ \text{move1*}(\gamma) \notin F \end{cases}$$

How to remove Backtracking

A linear Driver for NFA/DFA

The implementation of ***move1**** leads to a new linear, deterministic, recogniser

```
Answer move1star()
        {S= Clos({s0})};
         nextchar(c);
         while(c≠eof) and (S≠∅) {
                S=move1(S,c);
                nextchar(c)};
         if (S∩F) ≠ ∅  return 'accept';
         return 'noaccept';
        }
```

Linear *(but step move1 requires exponential time)*

Apply it to automaton A

<**S**= {0,1,2}, **Σ**= {a,b},
move= {<<0,a>{1}>,
<<1,b>{1,2}>}
s_0=0, **F**= {2} >

when scanning: abb$

A Converter from NFA to DFA

based on *compiling move1* into a *Transition Table*

Preliminaries Definitions:

Structures and Notational Conventions

Star: Set -> (Σ -> Set) is a function representing tables

- with **Set**-indexed rows and **Σ**-indexed columns
- **Star[s]** - is the **s**-indexed row of table **Star**
- **Star[s]c** - is the value at the row **s** and column **c**
- **Star** does not contain any **ε**-indexed column

A Basic Operation on Tables

merge(move,S) = merge-row({move[s] | s∈S})

$\forall_{c\in\Sigma}$ merge-row($\{R_1,\ldots,R_k\}$) c = $(\cup_{1\leq i\leq k} Clos(R_i c))$

	digit	.	ε
0	1		1
1	3	2	{1,3}
2		2	
3	0	4	
4	4	5	
5	6		
6	6		

move

merge-row({0,1,3})= | {0,1,3} | {2,4} |

A Converter from NFA to DFA

based on *compiling move1* into a *Transition Table*

Implementation: A Routine

```
Table movestar() //DFA Transition Table.
{EntryStar = ∅;
 List = Clos({s0});
 while (List ≠ emptylist) {
        S = firstout(List);
        if (S ∉ EntryStar) {
               add(S,EntryStar);
               Star[S]= merge(move,S);
               List=List+{Star[S]c | c∈Σ};
               }
        }
 return Star;
}
```

EntryStar = list of the state sets already cosidered for the row indices of Star
List = list of the reached state sets to be considered for inclusion in the row indices of Star
Firstout = selection and removal of the first element
Add = adds a state set to the current list EntryStar

A suitable renaming, of the resulting Table states, should be always used

Applying it to automaton A

<**S**= {0,1,2}, **Σ**= {a,b},
move= {<<0,a>{1}>,
<<1,b>{1,2}>}
s_0=0, **F**= {2} >

Results automaton A'

<**S**= {0,1,2}, **Σ**= {a,b},
move= {<<0,a>1>,<<1,b>2>,
<<2,b>2>}
s_0=0, **F**= {2} >
Renaming {1,2} with 2

Exercise 2

A Driver for DFA

```
Answer StarDriver()
        {s= s0;
         nextchar(c);
         while(c≠eof) and (s≠{}) {
                   s=Star[s]c;
                  nextchar(c);}
         if ((s∉F) or (c≠eof)) return 'noaccept'
        return 'accept';
        }
```

linear

All steps run in constant time)

Apply it to automaton B

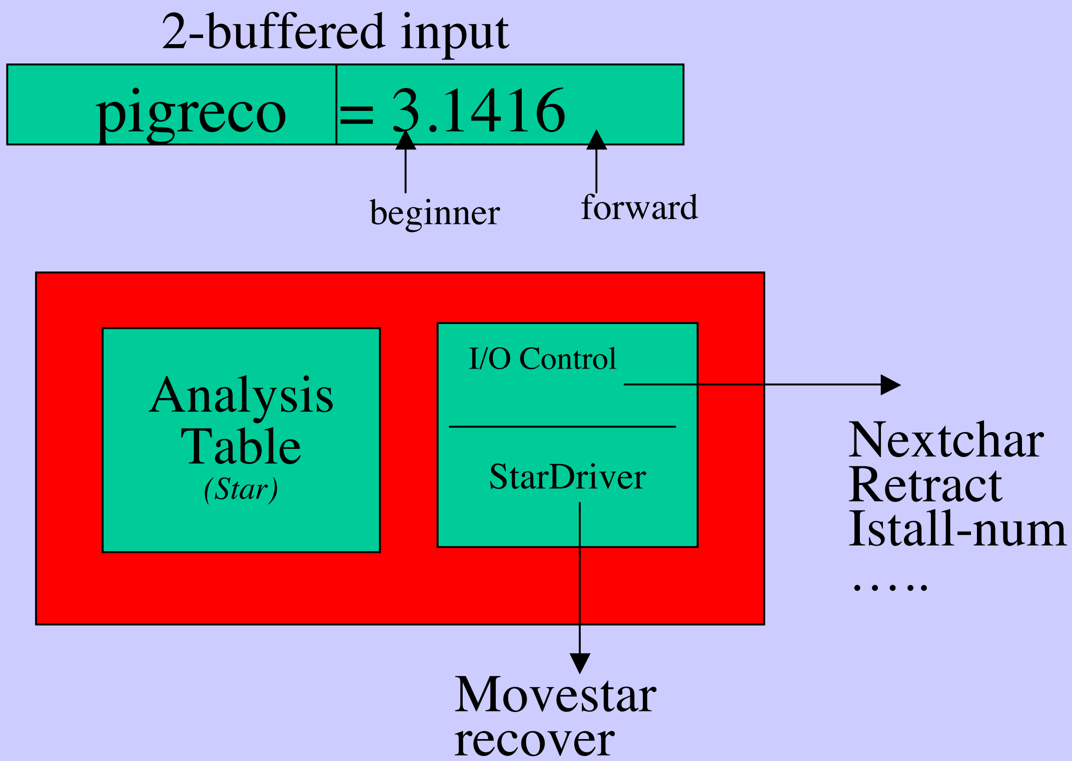
<**S**= {0,1,2}, **Σ**= {a,b},
move= {<<0,a>1>,<<1,b>2>,
<<2,b>2>}
s_0=0, **F**= {2} >

when scanning: abb$

How to build Scanner for a Lexics L

Use the following steps in order to obtain the structure on the right side:

1. Define a Regular Grammar G for L;
2. Check L(G)=L holds;
3. Define an Automaton A for G (using a safe transformational technique)
4. Convert A into a deterministic B having a Transition Table T.
5. Use the following setting:
   Analysis Table = T



How to do step 3? Suitably, extending Thompson's transformation to work on grammars, for instance, or using other techniques (dotted automata)

Conclusions

- *Lexic* is a Regular Language
- *Regular Languages* is a subset $\Re \subseteq 2^{\Sigma^*}$ of languages on Σ:
  - let $F = \{I \subseteq \Sigma^* \mid \#I < n$, for some natural n$\}$ be the set of all finite languages on Σ

  $\Re$ can be obtained from F by finite *unions* and *concatenations*, and *Kleene's star*

  $$\Re = F_{\{\cup, X_{\bullet}, *\}}$$

- *FSA* furnishes linear analyzers for $\Re$

Basic Properties (1)

Let L1 and L2 be Regular.

Union: is L1∪L2 Regular? yes

Product: is L1×L2 Regular? yes

Intersection: is L1∩L2 Regular? yes

Complement: is C(L1) Regular? yes

Decidability: Exists g computable such that g(L)=yes iff L is Regular?

NO

Basic Properties (2)

Let **L** be Regular, and
$A=\langle S,\Sigma,m,s_0,F\rangle$ be L(A)=L

Th.(iteration: --- Pumping Lemma ---∀ L, ∃ #S∈ℵ)
If x∈L, |x|>#S,
then: ∃u,w,v such that 0<|w|≤#S, x=u.w.v,
$u.w^k.v$ ∈L, *for each natural k*

Corollary: --- ∀ L, ∃ #S
If x∈L, |x|>#S,
then: ∃u,w,v such 0<|uw|≤#S, x=u.w.v,
$u.w^k.v$ ∈L, *for each natural k*

Apply it to prove that
$L = \{a^n b^n \mid n \text{ natural}\}$
is not Regular

Exercises

Exercise: Application of the Iteration Theorem

Problem. Prove that the language $L = \{a^n b^n \mid n \in \aleph, a,b \in \Sigma\}$, for given Σ, is not a regular language

Proof. (by contradiction, using pumping lemma)

- Assume L be regular.
- Then, the pumping lemma applies to L. Let m be the characteristic constant #S, of L, mentioned in the lemma.
- Then, let $x = a^m b^m$:
  - $x \in L$, since $m \in \aleph$
  - $|x| = 2m > m$, since $m > 0$
  - hence, $\exists$ u,w, v:
    - $x = uwv$
    - $|uw| \leq m$ & $|w| \neq 0$:
    - hence, $\exists$ m1,m2,m3:
      - $m2 \neq 0$ & $m = m1 + m2 + m3$ & $w = a^{m2}$ & $x = a^{m1} a^{m2} a^{m3} b^m$
      - hence:
        - $a^{m1} a^{m3} b^m \in L$ according Lemma since $k * m2 = 0$ for $k = 0$
        - $a^{m1} a^{m3} b^m \notin L$ by definition of L since $m1 + m3 \neq m1 + m2 + m3$ when $m2 \neq 0$

Exercises

Consider the lexics L of the numerals for integer and fixed-point numbers in decimal notation and arbitrary number of digits.

1. Give separate regular expressions for: integers, S, fixed point integers, F, the union of the two.
2. Give a grammar for lexic L.
3. Give an automaton for L.
3. Give a deterministic automaton for L.
4. Give deterministic recognizer Y for L
5. Modify Y for recognizing word sequences of L, separated by any character not in {0,..,9,'.'}. The new recognizer generates a sequence of <pi,li> where pi=position, li=length of the i-th recognized word. (words not in L are ignored)
7. Modify Y so that it recognizes the first, longer (7.1- shorter) word of the lexics that occur in a string on an alphabet containing {0,..,9,'.'}.

Esercizio - 1

1. Si diano le espressioni regolari per: i numerali per interi, S, i numerali per quelli in virgola fissa, F, l'unione dei due.

S = digit digit*
F = digit*.digit digit*
Digit = 0|1|…|9

Esercizio - 2

2. Si dia una grammatica regolare per tale lessico

S = digit digit*
F = digit*.digit digit*
Digit = 0|1|…|9

Regolare perchè: digit < S,F

oppure

F = digit*.S
S = digit digit*
Digit = 0|1|…|9

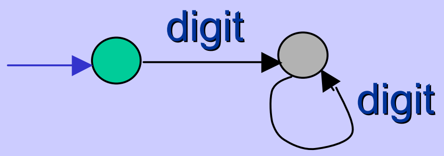
Regolare perchè: digit < S < F

Esercizio - 3

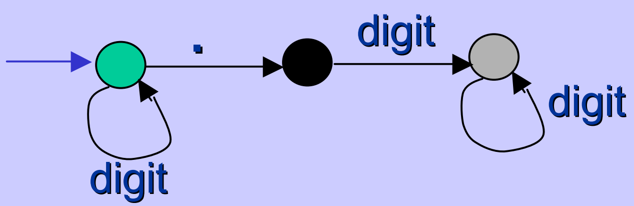
3. Si dia un automa per tale lessico

Osservazione. Possiamo semplificare la costruzione usando *digit* come fosse un carattere

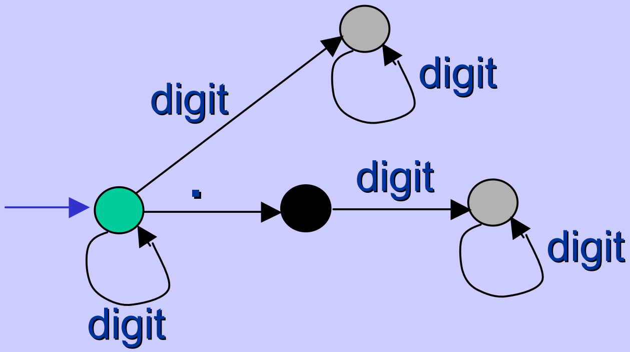
S = digit digit*



F = digit*. digit digit*



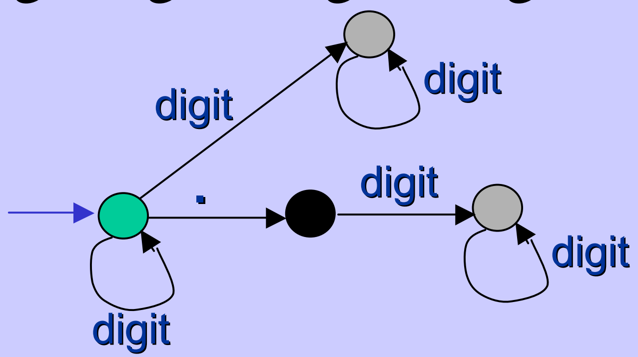
digit digit*| digit*. digit digit*



Esercizio - 4

4. Si dia un automa deterministico per tale lessico

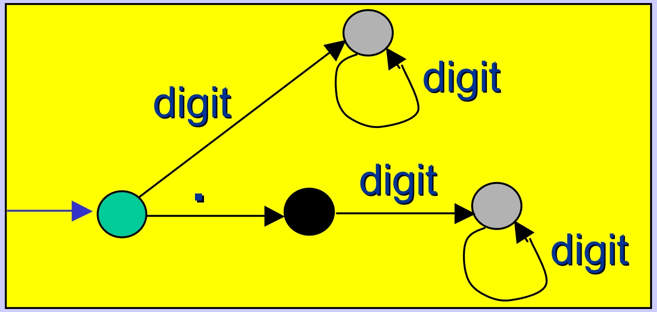
digit digit*| digit*. digit digit*



	digit	.
0	{0,1}	{2}
1	{1}	
2	{3}	
3	{3}	

Movestar

	digit	.
{0}	{0,1}	{2}
{0,1}	{0,1}	{2}
{2}	{3}	
{3}	{3}	



Esercizio - 5

5. Si dia un riconoscitore deterministico per tale lessico

```
Answer StarDriver()
{s= s0;
 nextchar(c);
 while(c≠eof) and (s≠{}) {
          s=Star[s]c;
          nextchar(c);}
 if ((s∉F) or (c≠eof)) answer='noaccept'
 else answer='accept';
 return (answer);}
```

Tabella di analisi: **Star**

	digit	.
{0}	{0,1}	{2}
{0,1}	{0,1}	{2}
{2}	{3}	
{3}	{3}	

Esercizio - 6

6. Si modifichi tale riconoscitore affinchè sia in grado di riconoscere sequenze di parole di tale lessico separate da caratteri non appartenenti a {0,..,9,’.’}

```
Answer StarDriver()
{input=0;
  while(nextchar(c) != eof){
     length=0; cur=input;
     s= s0;
     while(c ∈Σ && s≠{}) {
              s=Star[s]c;
              nextchar(c); //incr. anche input
              length++;}
     if (s∈F) answer= answer  ++ <cur,length>;}
return (answer);}
```

Tabella di analisi: **Star**

	digit	.
{0}	{0,1}	{2}
{0,1}	{0,1}	{2}
{2}	{3}	
{3}	{3}	