Regular Expressions: E_Σ

Some properties : Occasions for remarks and exercises

- Are * and . such that: $(e_1.e_2)^* = e_1^*. e_2^*$? - Give a proof of your claim.

- Is | *left distributive* over . ? - Prove your claim.

- Prove that * is *idempotent*.

- Why the knowledge of the operator properties is a relevant task? - Show a concrete situation.

Exercises - 1

- Are ***** and **.** such that: $(\mathbf{e_1.e_2})^* = \mathbf{e_1}^*.\, \mathbf{e_2}^*$? - Give a proof of your claim.
  - No
  - Proof. (by counterexample) Let e1=a, e2=b. Then we show that for $s=a^2b$: $s \in e1^*e2^*$ but $s \notin (e1e2)^*$.

    $e1^*e2^* = (\cup_{i\in\aleph} e1^i)X.(\cup_{i\in\aleph} e2^i) = (\cup_{i\in\aleph}\{a\}^i)X.(\cup_{i\in\aleph}\{b\}^i)$

    $= \cup_{i,j\in\aleph} (\{a\}^i X.\{b\}^j) = \cup_{i,j\in\aleph}\{a^i b^j\}$

    hence $s \in \cup_{i,j\in\aleph} \{a^i b^j\}$, for i=2, j=1

    $(e1e2)^* = \cup_{i\in\aleph}(\{a\}X.\{b\})^i = \cup_{i\in}\{ab\}^i$

    hence $s \notin \cup_{i\in\aleph}\{ab\}^i$, because each *a* must be always, followed by one *b*.

- Is **|** *left distributive* over **.** ? - Prove your claim.
  - No. We prove that ∃ e1,e2,e3 such that: (e1.e2)|e3 ≠ (e1|e3).(e2|e3)
  - Proof. ….

Exercises - 2

- Prove that * is *idempoten*

  proof: For all e, let Sem(e)=[e]. Then:

  $(e^*)^* = \cup_{i\in\aleph}(Sem(e^*))^i = \cup_{i\in\aleph} (\cup_{j\in\aleph} Sem(e)^j)^i$

  $= \cup_{i\in\aleph} (\cup_{j\in\aleph} [e]^j)^i = \cup_{i,j\in\aleph} [e]^{i*j}$

  since ($\forall$ i,j$\in\aleph$, i*j $\in\aleph$) and, ($\forall$n $\in\aleph$, $\exists$i,j $\in\aleph$ i*j=n), i*j computes only and all naturals when i,j are ranging on $\aleph$. Then,

  $\cup_{i,j\in\aleph} [e]^{i*j} = \cup_{i\in\aleph} [e]^i = e^*$

- Why the knowledge of the operator properties is a relevant task? - Show a concrete situation.
  - it helps in studying and simplifying problems and solutions that are using the operators
  - As a concrete situation consider: (e1*|e2*|e1*)* can be replaced by (e1|e2)

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

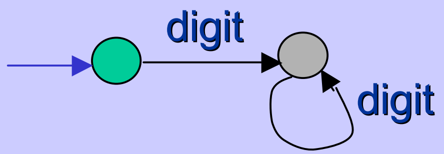
Regolare perchè: digit < S < F

Esercizio - 3

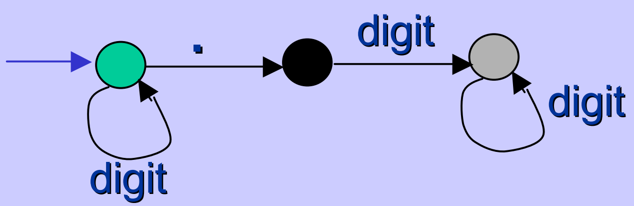
3. Si dia un automa per tale lessico

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

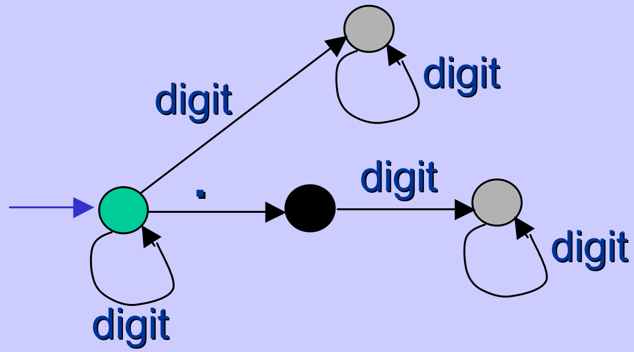
S = digit digit*



F = digit*. digit digit*



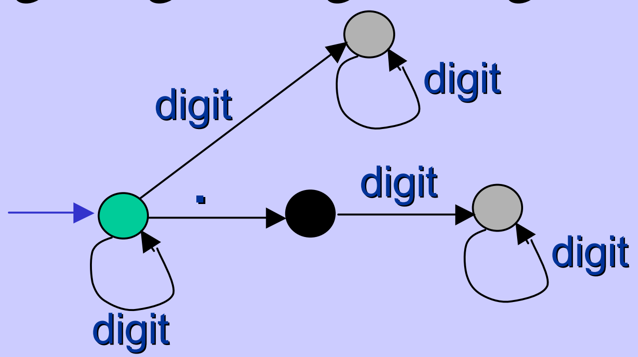
digit digit*| digit*. digit digit*



Esercizio - 4

4. Si dia un automa deterministico per tale lessico

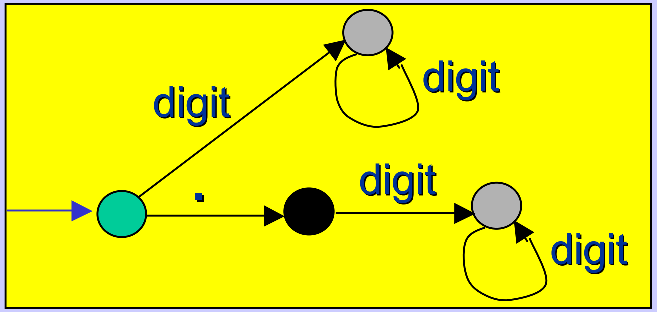
digit digit*| digit*. digit digit*



	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);}
```

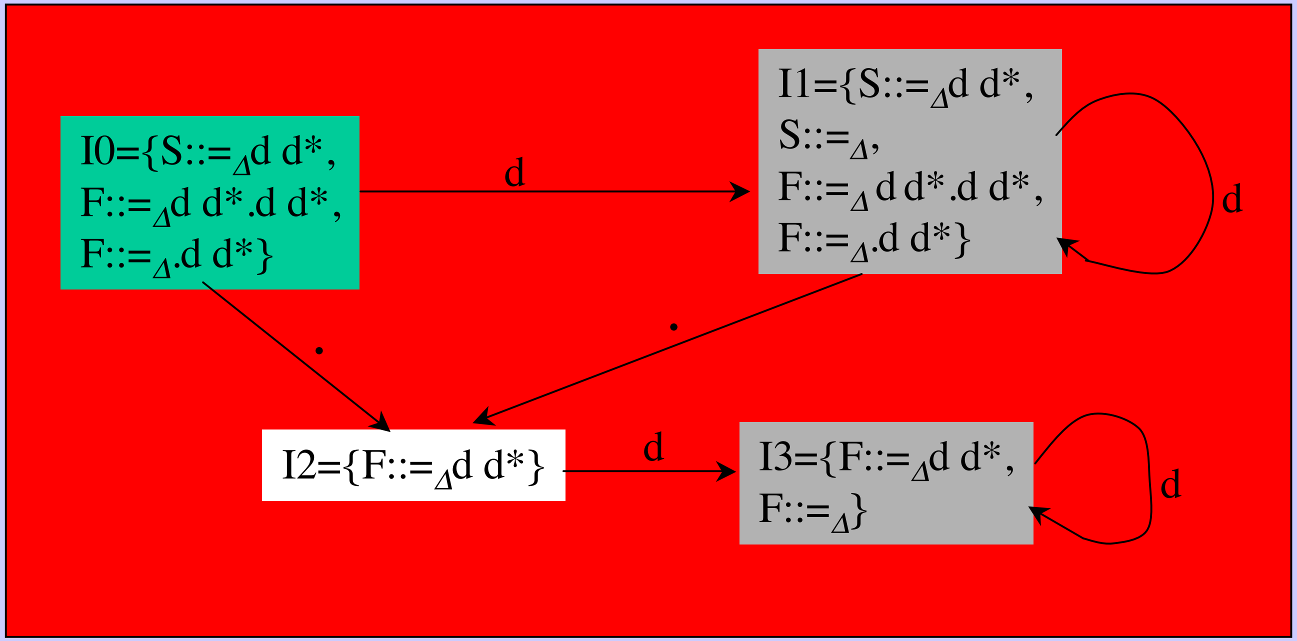
Tabella di analisi: **Star**

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

`

Automi di Items: un esempio

S::= digit digit*
F::= digit*.digit digit*



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