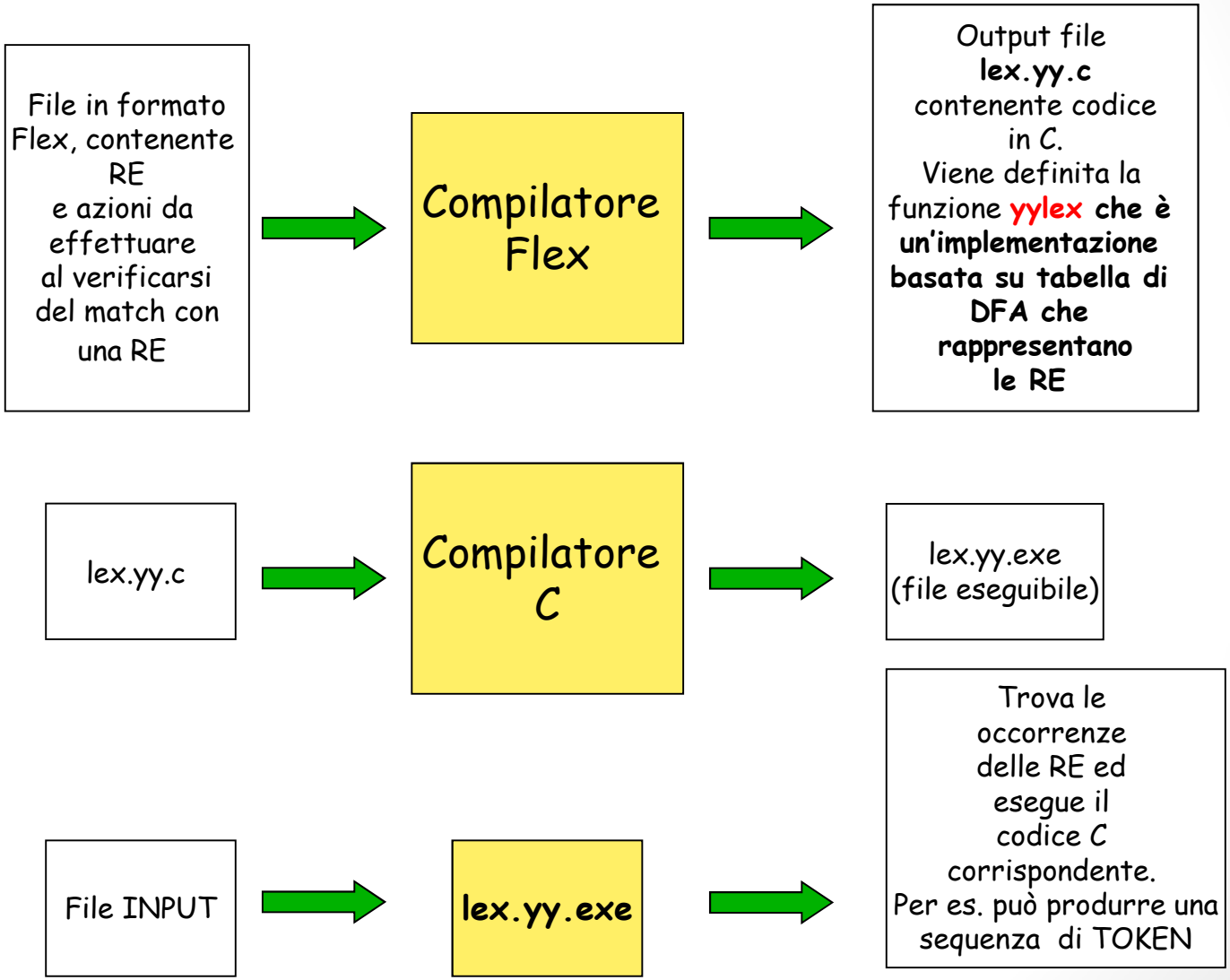
Generatori automatici di scanner: Flex

Lunedì 8 Ottobre

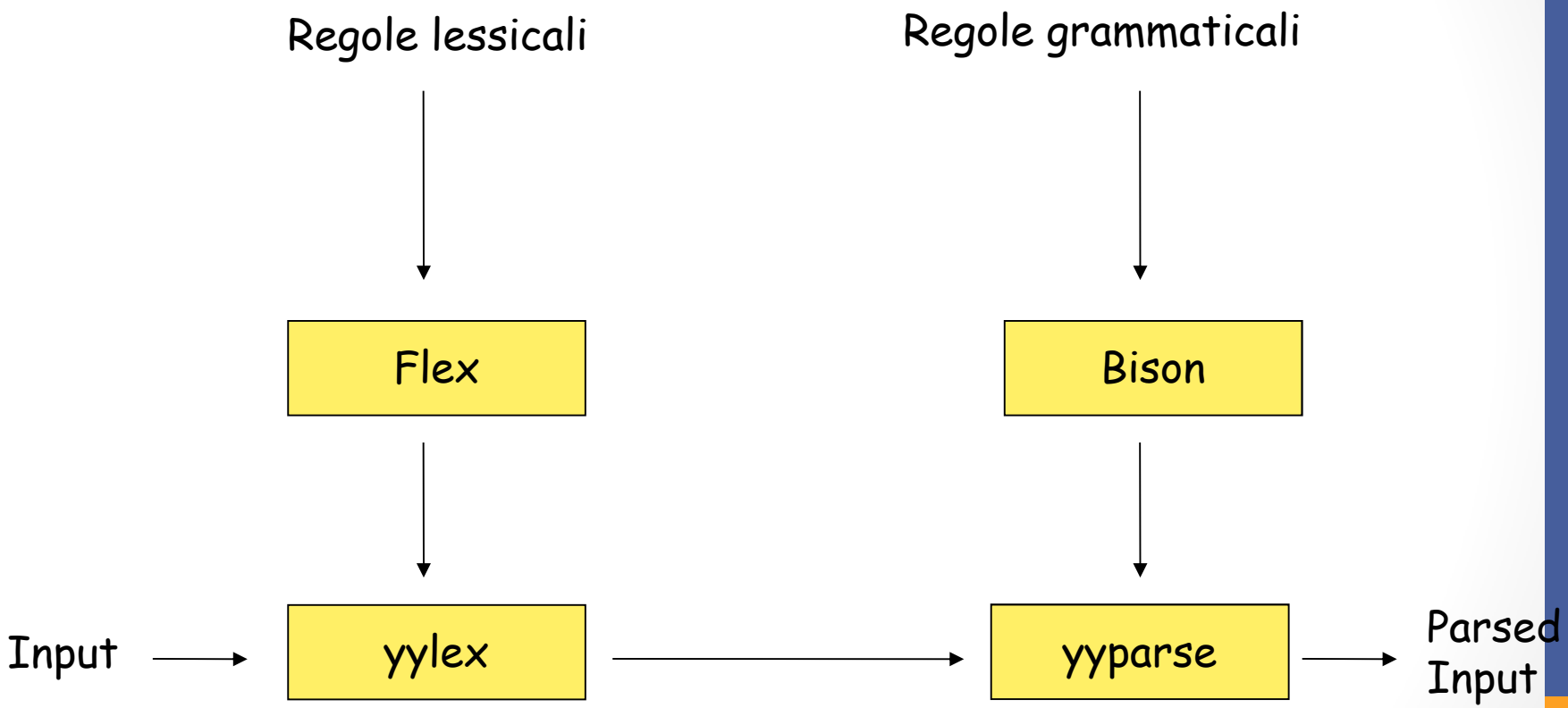
Generatori di scanner

- Un generatore automatico di scanner prende in input un file che specifica il lessico di un certo linguaggio:
  - solitamente nella forma di espressioni regolari
  - …e includendo altre funzioni ausiliarie, definizioni di token, …
- …e produce in output un codice (scritto in un certo linguaggio) che implementa il ruolo dello scanner
- Esitono molti generatori automatici:
  - Lex, Flex, ScanGen, …
    - generano un codice in C
  - JLex, Sable, Cup
    - generano un codice in Java
  - Lex fu il primo generatore di scanner. Esso fu inventato da Mike Lesk e Eric Shmidt (AT&T Bell Lab) nel 1975.
  - Esistono tanti software alternativi al Lex. Uno dei più conosciuti ed usati è Flex – Fast Lexical analyser generator (introdotta da Vern Paxson intorno al 1987 per risolvere problemi di efficienza).
  - Flex è un free software. Pur non essendo un software GNU, il GNU Project ne distribuisce un manuale.
    - http://flex.sourceforge.net/ (linux)
    - http://gnuwin32.sourceforge.net/packages/flex.htm (windows)

Uso di Flex


File in formato Flex, contenente RE e azioni da effettuare al verificarsi del match con una RE
Compilatore Flex
Output file lex.yy.c contenente codice in C. Viene definita la funzione yylex che è un'implementazione basata su tabella di DFA che rappresentano le RE
lex.yy.c
Compilatore C
lex.yy.exe (file eseguibile)
File INPUT
lex.yy.exe
Trova le occorrenze delle RE ed esegue il codice C corrispondente. Per es. può produrre una sequenza di TOKEN

Uso di Flex e Bison



Scrivere un programma in Flex

Un file in formato Flex (o Lex) consiste di 3 sezioni, separate da %%:

DEFINIZIONI
che contiene:

- definizioni di variabili o definizioni regolari
- segmento di codice C, indentato oppure delimitato da %{ e %}, che deve comparire nel file di output

%%

REGOLE
che contiene una sequenza di regole contenenti:

- i pattern espressi mediante RE
- codice C da eseguire in corrispondenza del match con un certo pattern

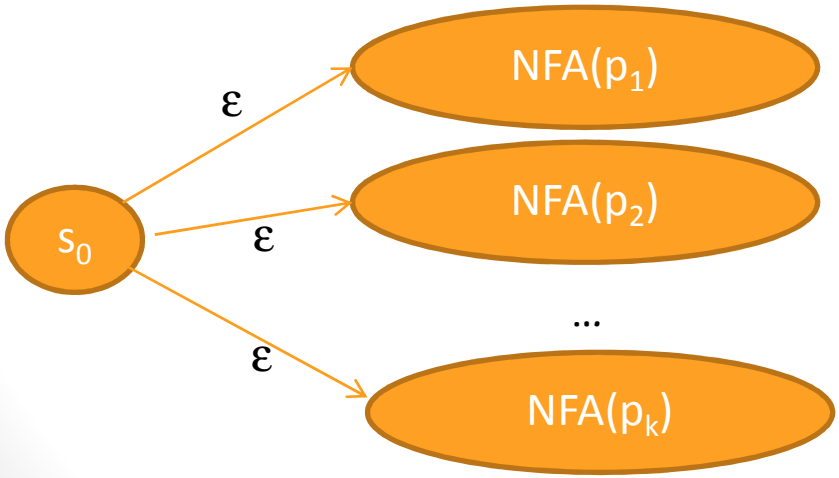
%%

FUNZIONI AUSILIARIE (opzionale)
che contiene codice C che deve essere copiato nel file di output

Anche nella sezione REGOLE è possibile inserire codice tra %{ e %}. Ciò deve avvenire prima della prima regola.
Può servire per esempio per dichiarare variabili locali usate nella routine di scanning.

Come funziona Flex?

Si descrivono tutte le espressioni regolari che definiscono i token
Converte le espressioni regolari in automi non deterministici
Combina tutti i NFAs in un unico NFA



Come funziona Flex?

Converte il NFA in un automa deterministico (DFA).

Ottimizza l'automa deterministico

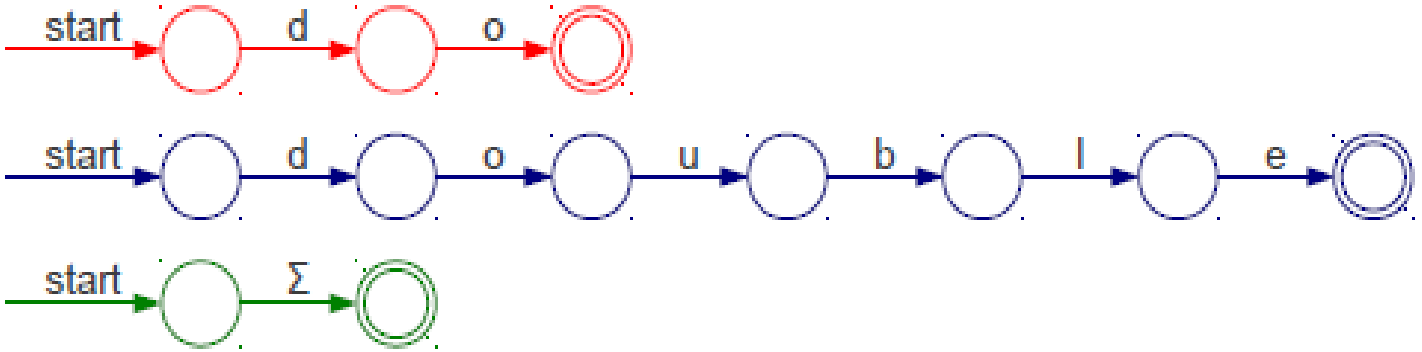
Produce codice per simulare l'azione del DFA

Come costruire il DFA?
E' sufficiente una subset construction?
Come si realizza il riconoscimento del longest best match o la regola di priorità?

Partiamo dagli NFA…

Più in dettaglio…

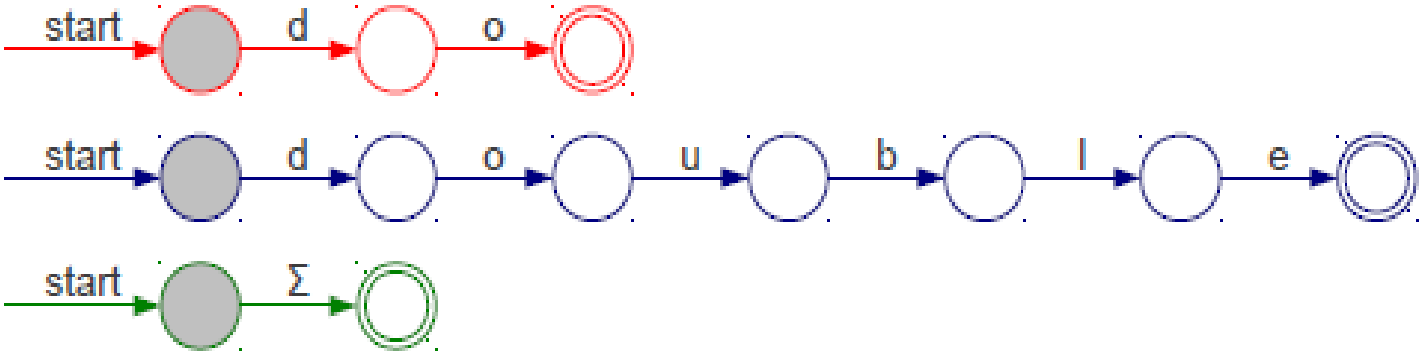
T_do	do
T_double	double
T_nonso	[a-zA-Z]

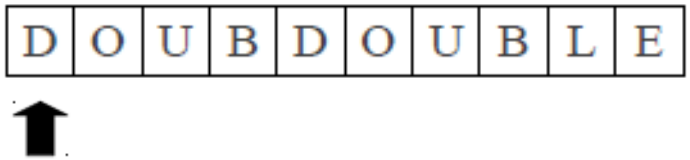


D	O	U	B	D	O	U	B	L	E

Più in dettaglio…

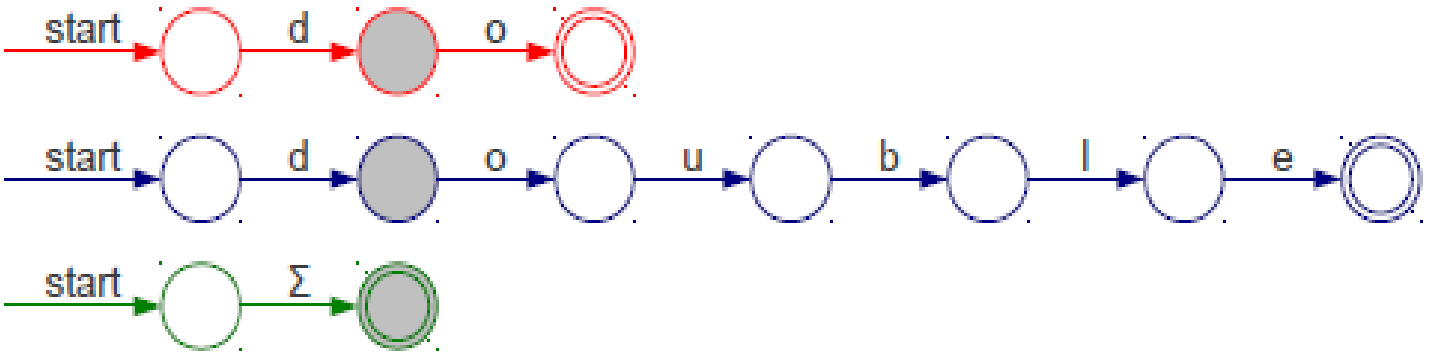
T_do	do
T_double	double
T_nonso	[a-zA-Z]

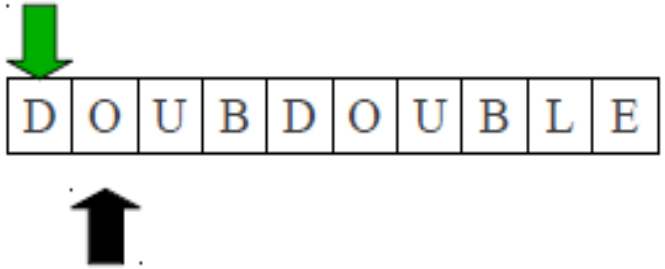




Più in dettaglio…

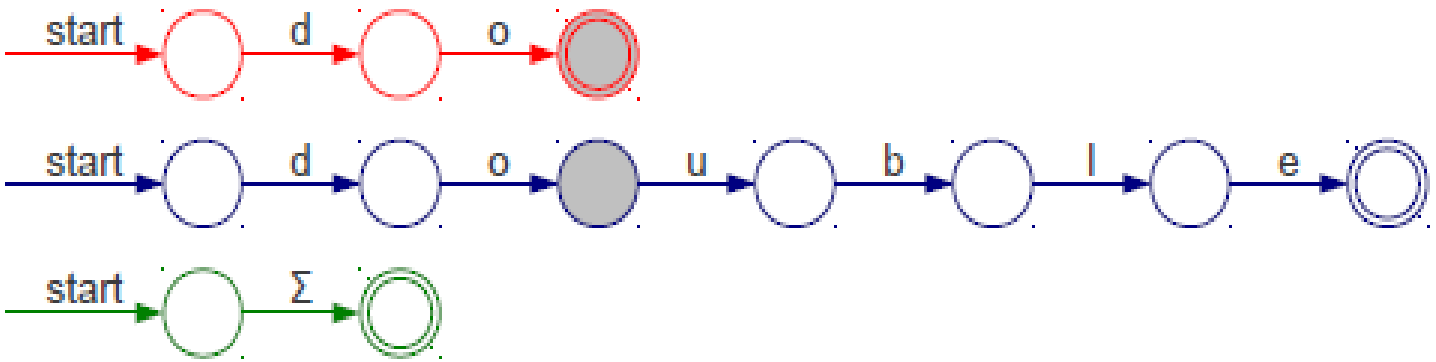
T_do	do
T_double	double
T_nonso	[a-zA-Z]

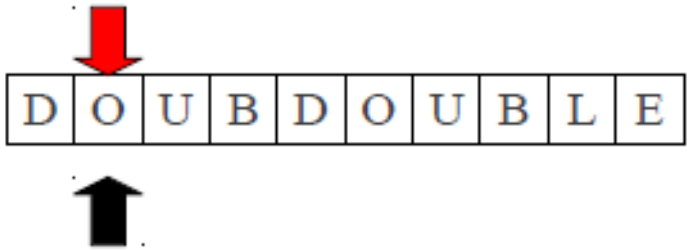




Più in dettaglio…

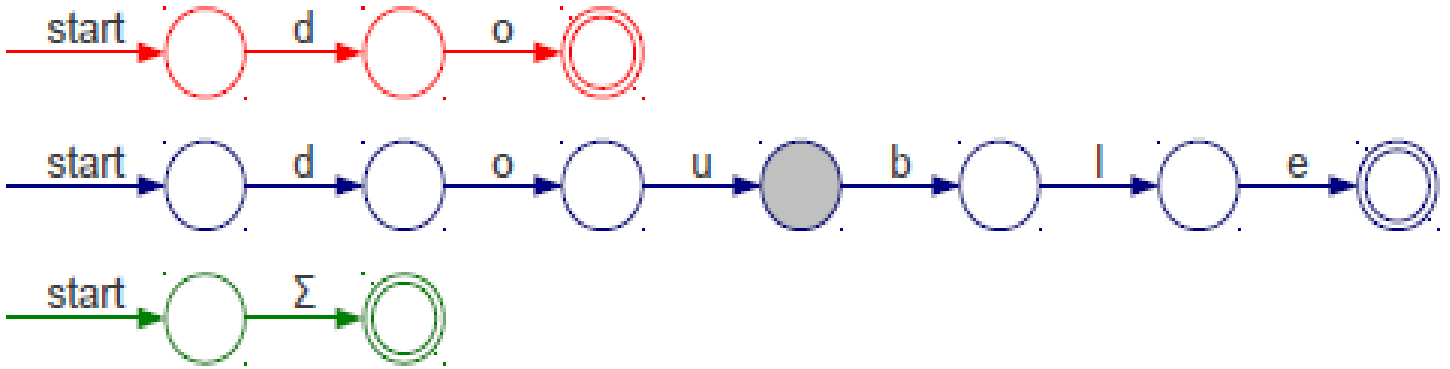
T_do	do
T_double	double
T_nonso	[a-zA-Z]

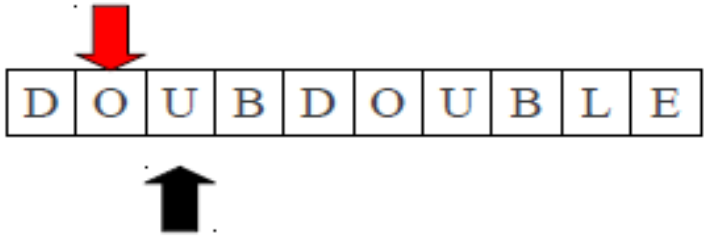




Più in dettaglio…

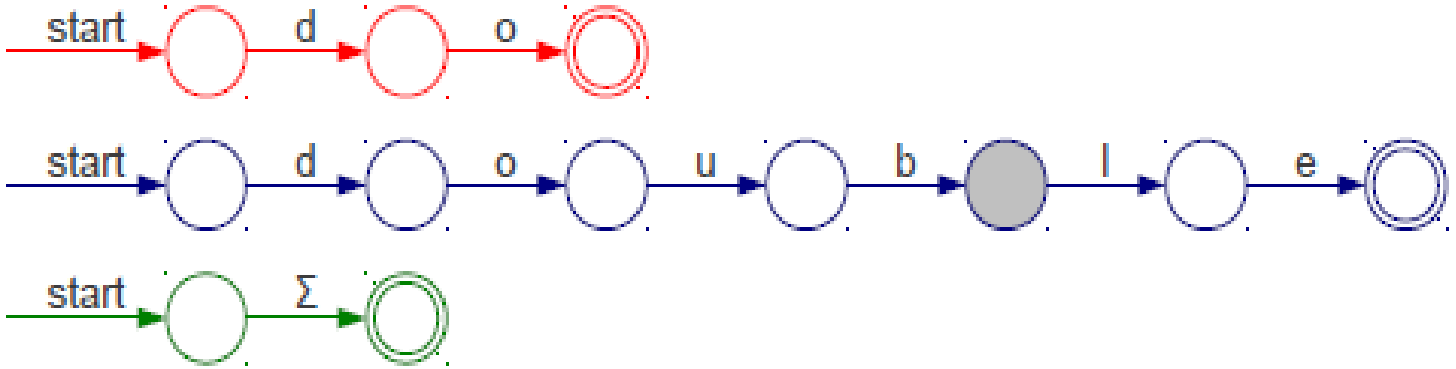
T_do	do
T_double	double
T_nonso	[a-zA-Z]

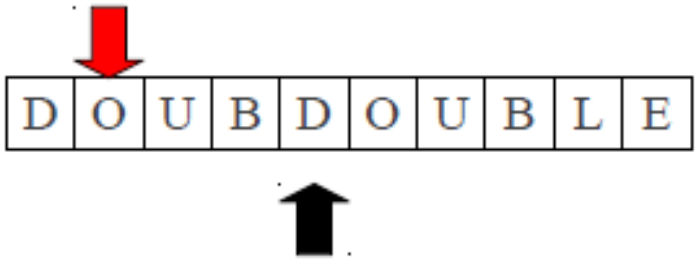




Più in dettaglio…

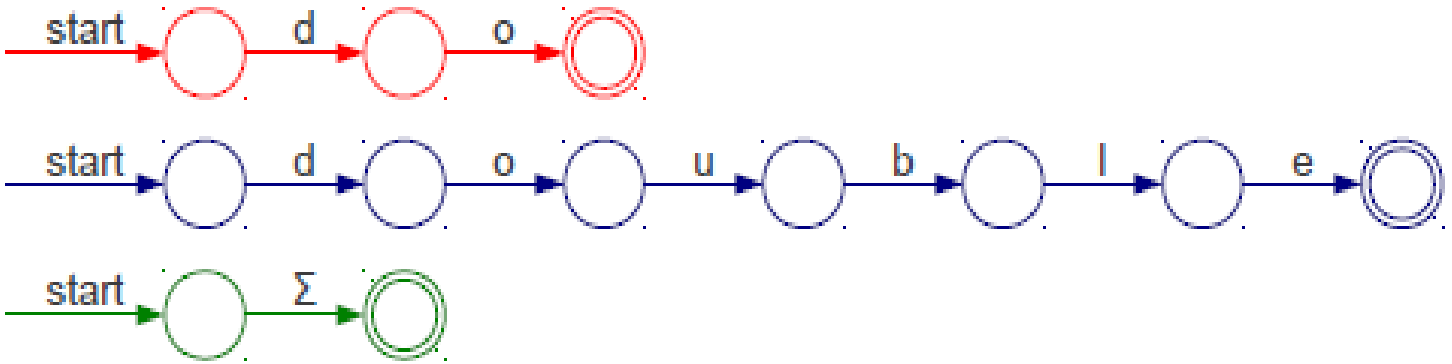
T_do	do
T_double	double
T_nonso	[a-zA-Z]

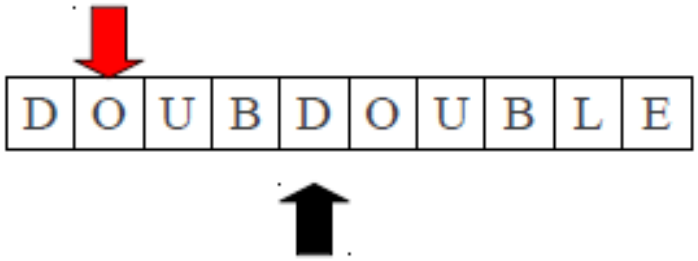




Più in dettaglio…

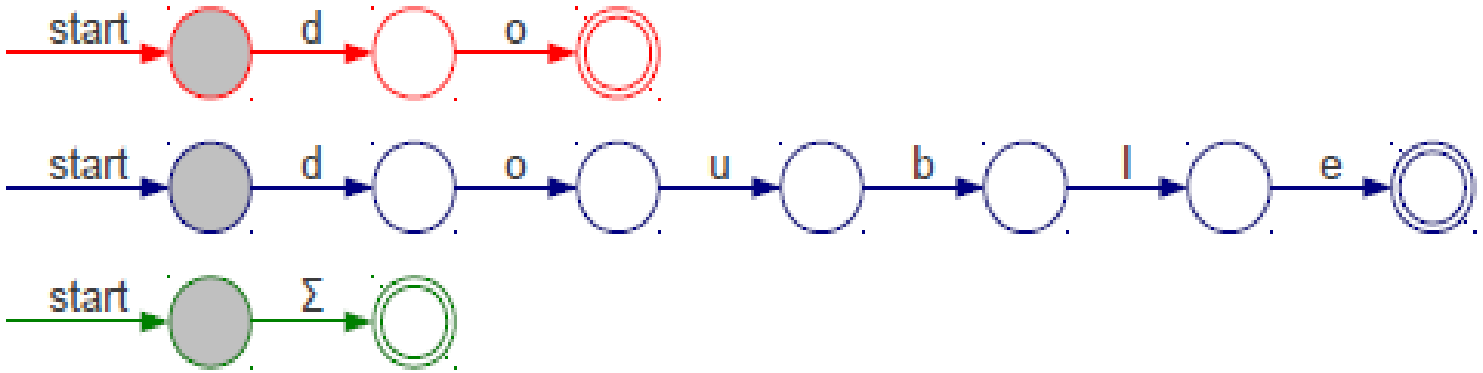
T_do	do
T_double	double
T_nonso	[a-zA-Z]

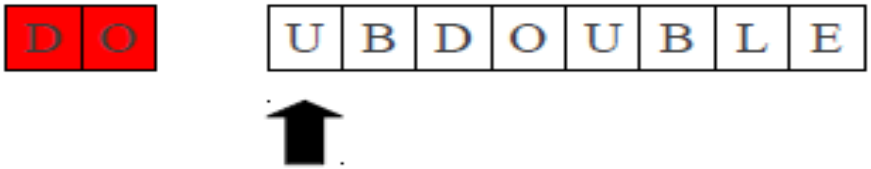




Più in dettaglio…

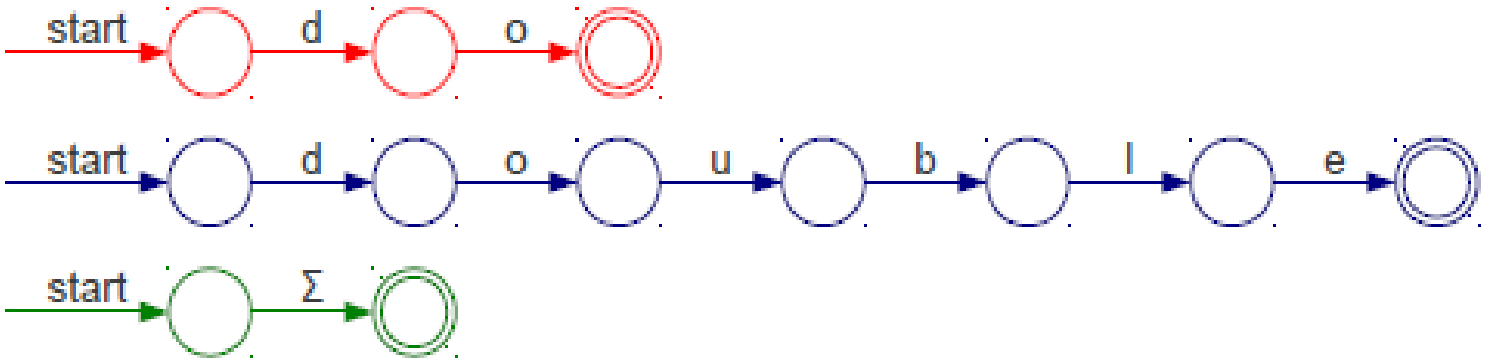
T_do	do
T_double	double
T_nonso	[a-zA-Z]

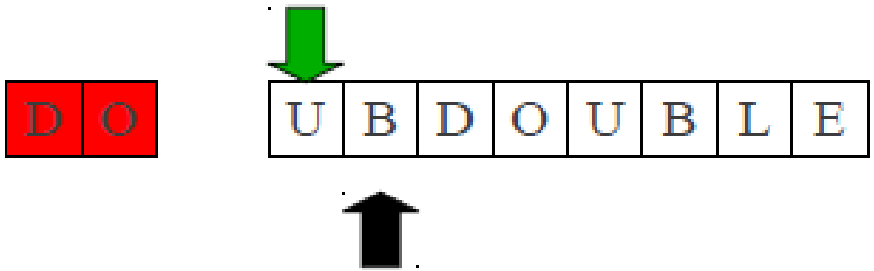




Più in dettaglio…

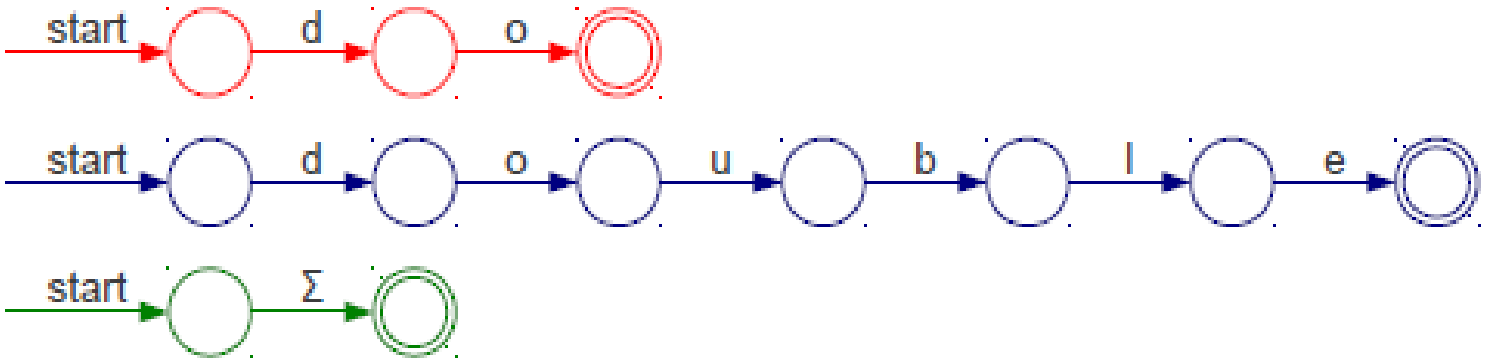
T_do	do
T_double	double
T_nonso	[a-zA-Z]

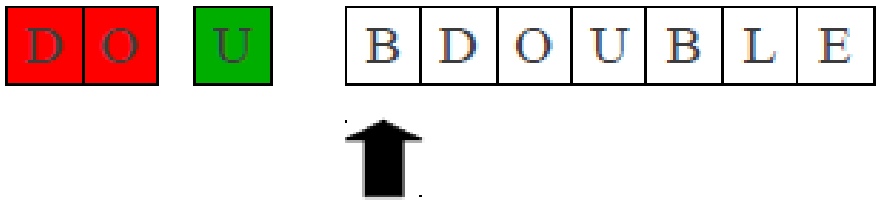




Più in dettaglio…

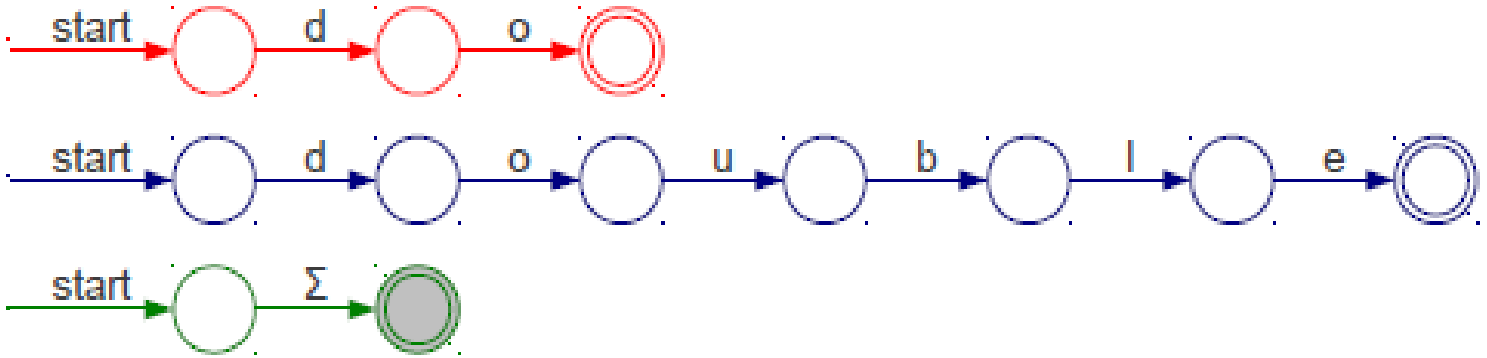
T_do	do
T_double	double
T_nonso	[a-zA-Z]

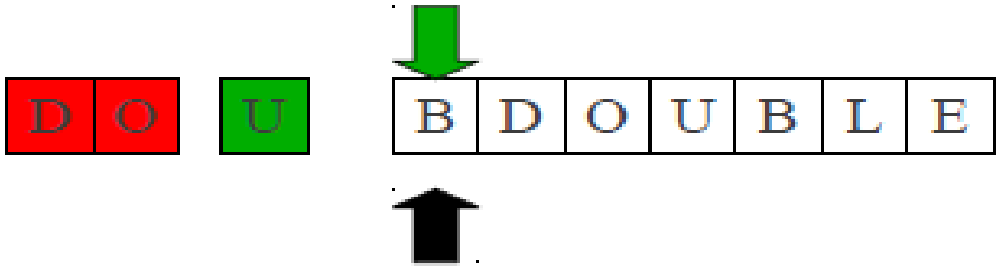




Più in dettaglio…

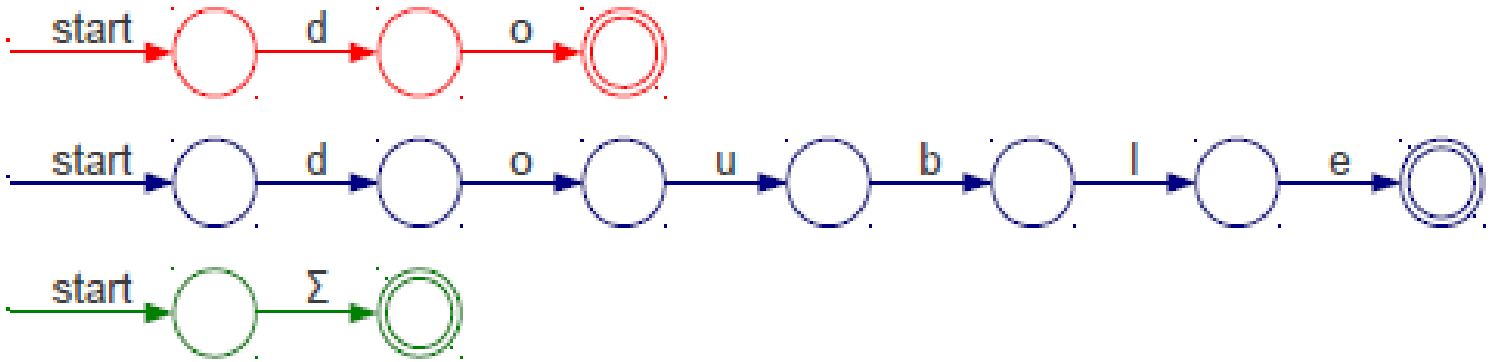
T_do	do
T_double	double
T_nonso	[a-zA-Z]

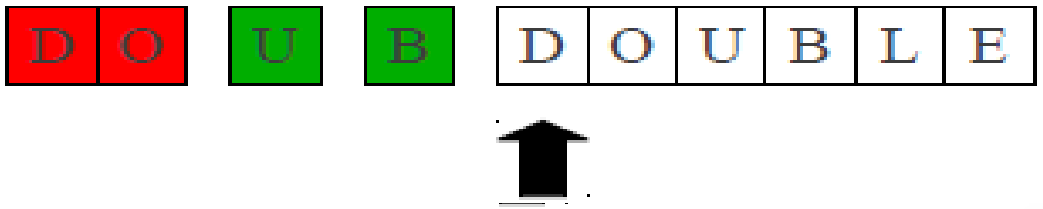




Più in dettaglio…

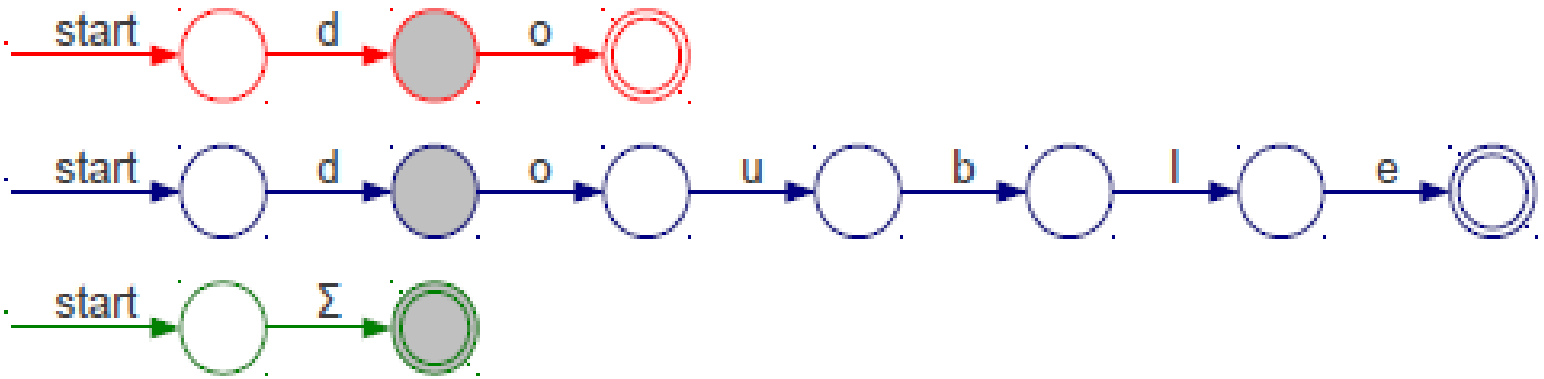
T_do	do
T_double	double
T_nonso	[a-zA-Z]

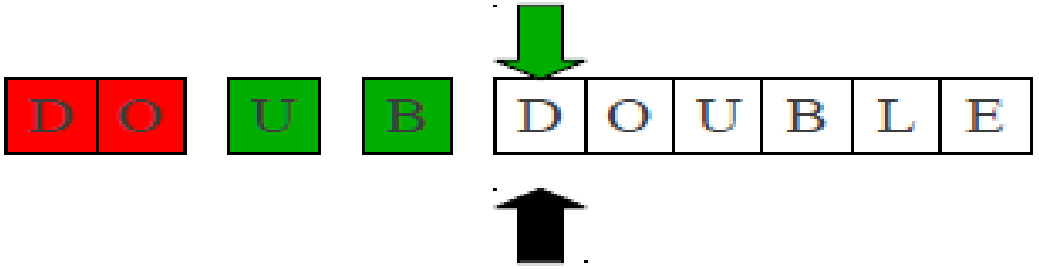




Più in dettaglio…

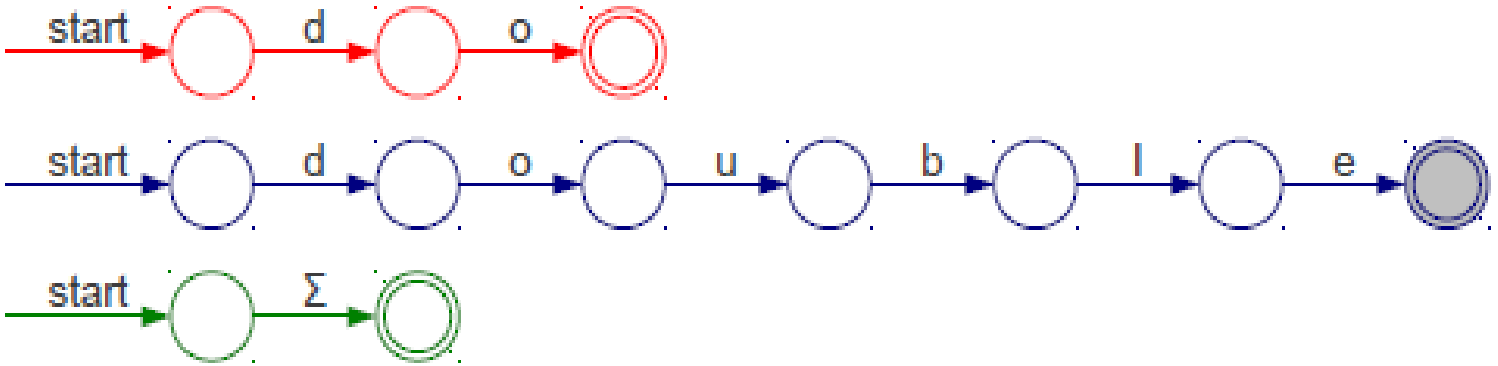
T_do	do
T_double	double
T_nonso	[a-zA-Z]

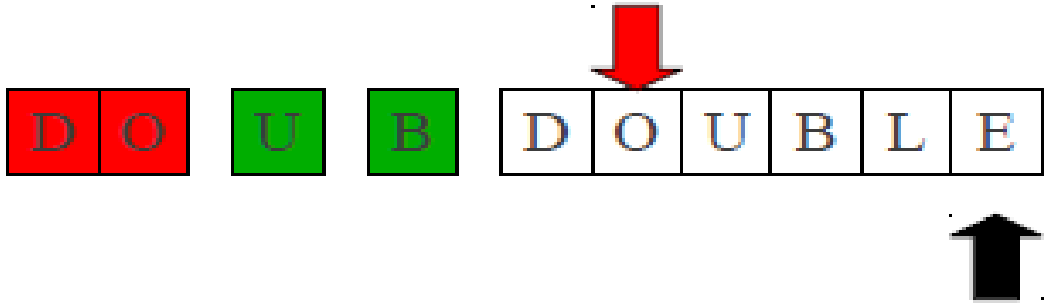




Più in dettaglio…

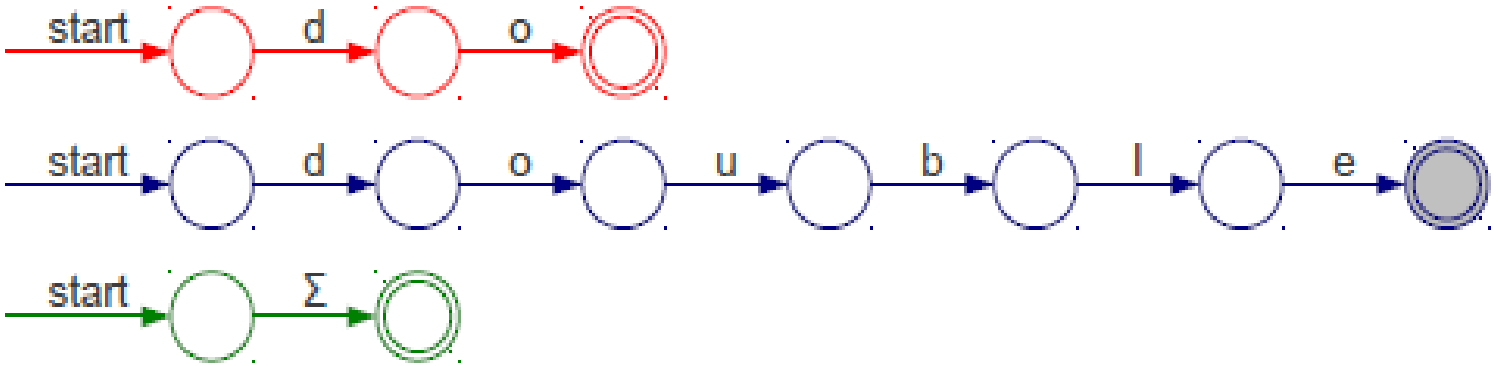
T_do	do
T_double	double
T_nonso	[a-zA-Z]

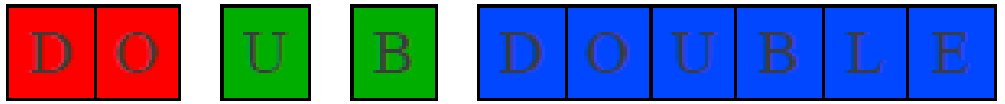




Più in dettaglio…

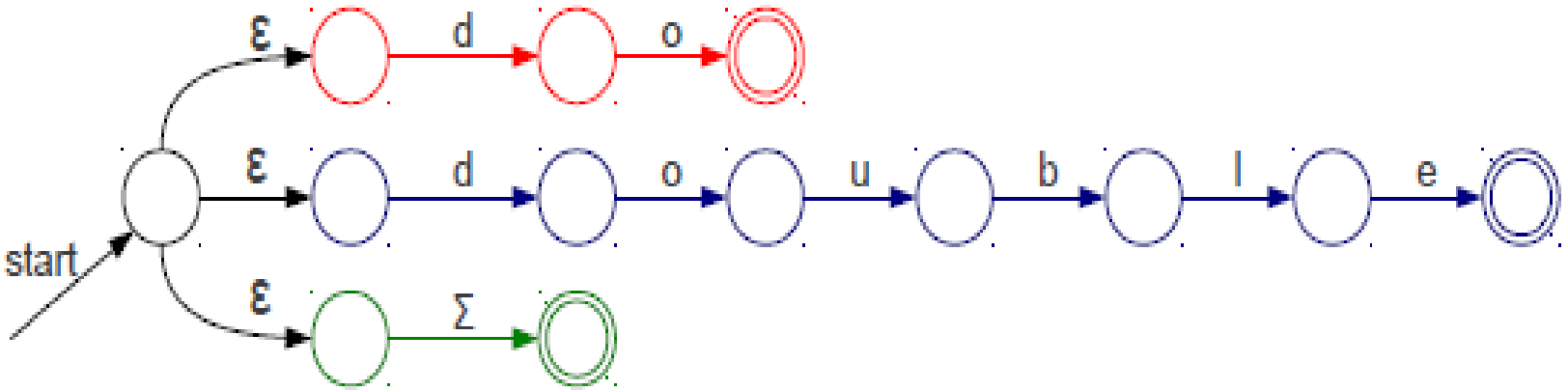
T_do	do
T_double	double
T_nonso	[a-zA-Z]





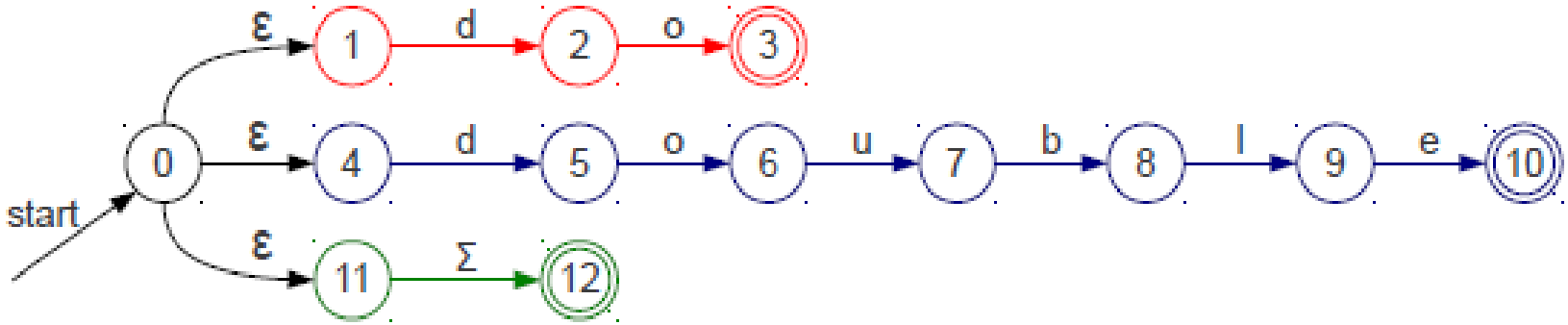
Cosa succede nel DFA?

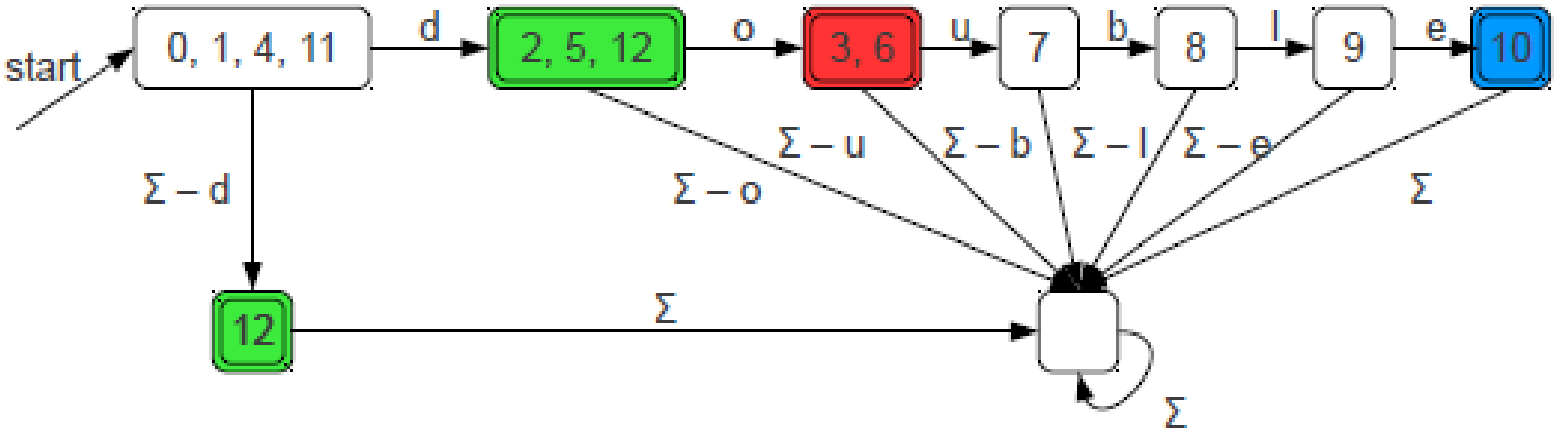
Variante della subset construction



Si applica la subset construction
Nel DFA ogni stato che contiene stati accettanti, conterrà anche l'informazione sul NFA di provenienza.

Dal NFA al DFA



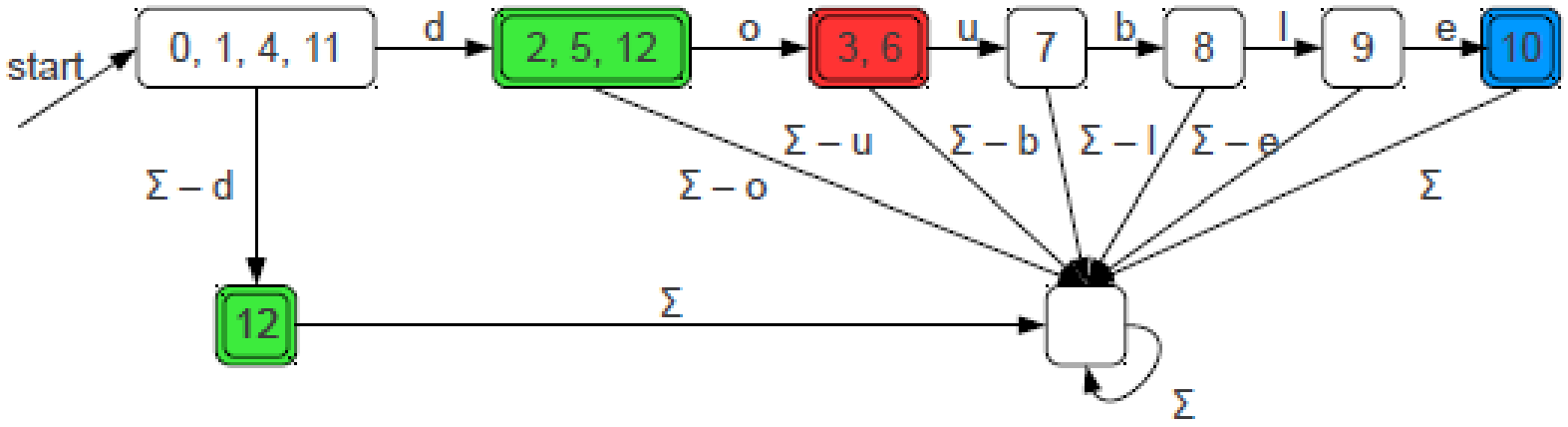


Dal NFA al DFA

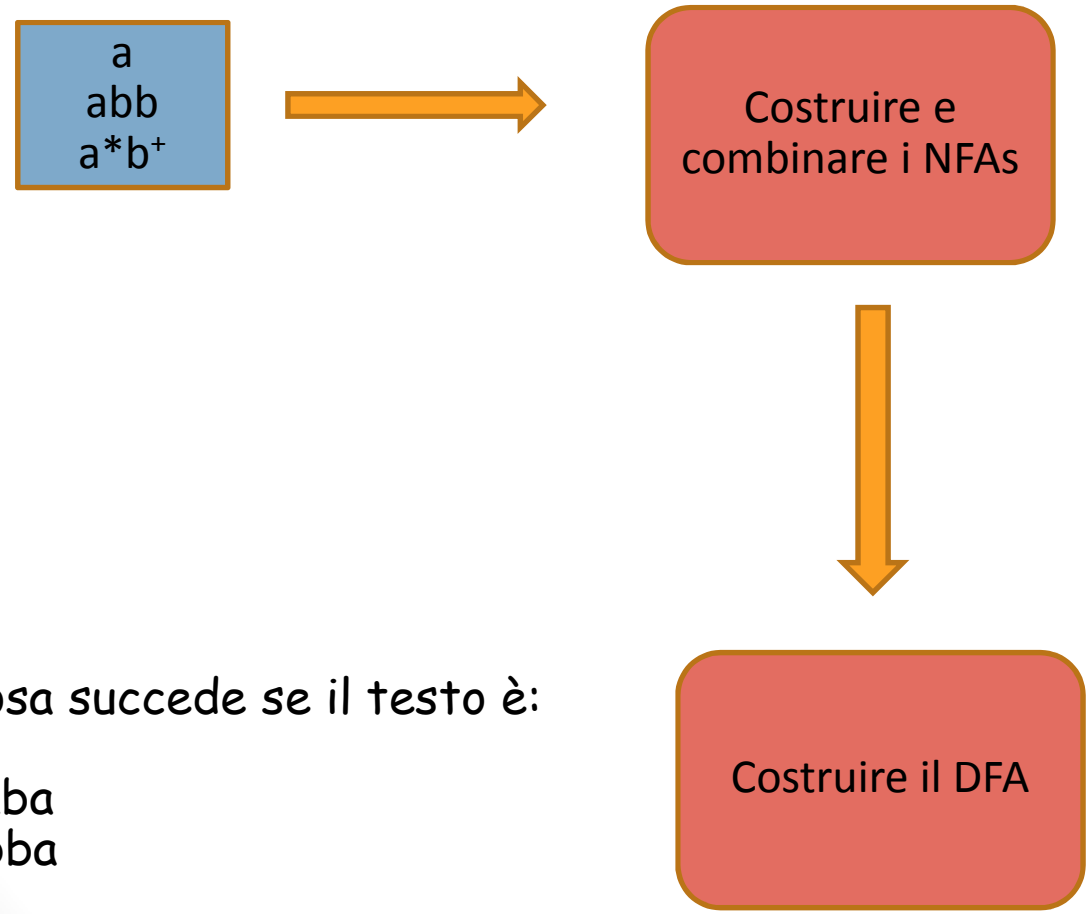
- Se in uno stato ci fossero più stati accettanti verrebbe definito un ordine tra questi.

- Si legge il testo un carattere per volta e mantiene memoria degli stati intermedi. Appena si arriva in uno stato pozzo o non ci sono le transizioni uscenti, si cerca a ritroso lo stato di accettazione.

- *Suggerimento per l'implementazione: E' possibile usare la pila in cui si inseriscono via via gli stati di accettazione incontrati durante il processo di matching*



Per esercizio…

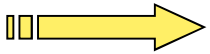


Cosa succede se il testo è:

aaba
abba

I primi esempi...

Il programma più semplice:

primo.fl 

```
%%
```

Esiste la regola di default:
il testo che non “matcha” con nulla viene ricopiato sull’output.

Genera uno scanner che semplicemente copia il suo input (un carattere per volta) nel suo output.

N.B. Sia nella sezione delle definizioni che in quella delle regole:
- ogni testo indentato o racchiuso tra %{ e %} viene copiato parola per parola sull’output.
- %{ e %} devono apparire a inizio di linea (non indentati).
- nella sezione delle definizioni, un commento non-indentato che comincia con /* e finisce con */ diventa un commento nel file di output

Come si usa:
1. flex primo.fl
2. gcc lex.yy.c -o primo
3. primo < prova.txt >output.txt

Esempio 1

- Mostriamo come deve essere scritto il file primo.fl

```
/* E' un opzione che serve nel caso si voglia utilizzare la
funzione main generata di default */
%option main

%%
```

Alcune opzioni

%option seguito da

main: flex fornisce un routine main() di default

noyywrap: lo scanner non chiamerà la routine yywrap che serve per la gestione di più file di input

Come definire i pattern nella sezione delle regole?

x match con il carattere'x'

. Qualsiasi carattere escluso il newline

[xyz] una “classe di caratteri"; in questo caso, il pattern matcha o con 'x', 'y', o 'z'

[abj-oZ] una “classe di caratteri“ contenente un range; matcha con 'a‘, 'b', una qualsiasi lettera da 'j' a 'o', oppure 'Z'

[^A-Z] una “classe negata di caratteri", i.e., qualsiasi carattere escluso quelli della classe. In questo caso, un qualsiasi carattere escluso le lettere maiuscole.

[^A-Z\n] qualsiasi carattere tranne le maiuscole e il newline

Note:
Con le parentesi quadre solo \ - e ^ sono caratteri speciali; per includere il carattere – questo deve apparire come primo o ultimo carattere.

Come definire i pattern nella sezione delle regole?

r*	zero o più espressioni r
r+	una o più espressioni r
r?	zero o una r
r{2,5}	da due a cinque r
r{2,}	due o più r
r{4}	esattamente 4 r
{name}	espansione della definizione “name”
"[xyz]\"foo"	la stringa: [xyz]"foo

Note:
il carattere \ ha il ruolo di carattere di escape.

Come definire i pattern nella sezione delle regole?

\x se x è 'a', 'b', 'f', 'n', 'r', 't', o 'v', allora \x ha la stessa interpretazione che per ANSI-C. Altrimenti, sta per il carattere x. (usato per l'escape di operatori come '*')

\0 carattere NULL (ASCII code 0)

\123 carattere con valore ottale 123

\x2a carattere con valore esadecimale 2a

(r) Espressione r; le parentesi servono per imporre delle precedenze

rs espressione r seguita da s

r|s o r oppure s

r/s una r ma solo se seguita da s. Il testo matchato da s è considerato incluso per effetto del "longest match", ma è restituito all'input prima che venga eseguita l'azione. Questo tipo di pattern è chiamato "trailing context".

Nota: valgono le regole di precedenza per le espressioni regolari; ad es. ciao|bu* è equivalente a (ciao)|(b(u)*)

Come definire i pattern nella sezione delle regole?

^r	r, ma solo all'inizio di una linea
r$	r, ma solo alla fine di una linea. Equivalente a "r/\n".
<s>r	r, ma solo nella start condition s
<s1,s2,s3>r	stessa cosa, ma in una qualsiasi start condition s1, s2, o s3
<*>r	r in una qualsiasi start condition, anche in una esclusiva.
<<EOF>>	end-of-file
<s1,s2><<EOF>>	end-of-file in start condition s1 o s2

[:digit:] indica tutti i caratteri per cui "isdigit" restituisce true
ES: [[:alpha:][:digit:]] è equivalente a [a-zA-Z0-9]

Come avviene il match?

- Quando viene determinato un match, il testo corrispondente al match viene reso disponibile attraverso la variabile globale **yytext** e la sua lunghezza viene memorizzata in **yyleng.**
- **Vengono quindi eseguite le azioni corrispondenti al pattern per il quale avviene il match.**
- Prosegue la scansione dell'input alla ricerca di altri match.
- Se non viene trovato alcun match, viene eseguita la *default rule:il carattere dell'input viene considerato un match e copiato nell'output*

Nota: la variabile yytext può essere definita come char pointer o array; E' possibile controllare tale definizione attraverso la direttiva %pointer o %array nella sezione delle definizioni.
Se si usa l'opzione –l (compatibilità con Lex), yytext è un array.
% pointer consente scansioni più veloci

Come definire le azioni?

pattern **azione**

Direttive speciali che possono essere incluse in un'azione:
ECHO: copia yytext nell'output dello scanner
BEGIN: seguito dal nome di una condizione START posiziona lo scanner in tale condizione
REJECT: dirige lo scanner a procedere sul secondo longest best match (di solito un prefisso dell'input)

- Se l'azione è vuota (contiene solo **;**), al match del pattern non succede nulla.
- Se l'azione contiene un {, allora verranno eseguite le azioni fino al raggiungimento di }.
- Un'azione che consiste solo di | indica che è definita nella stesso modo della regola che segue.

Es1:

```
[ ]     |
\t      |
\n      ;
```

Es2:

```
[a-z]+          {printf("%s",yytext);}
```

È equivalente a

```
[a-z]+          {ECHO;}
```

Variabili e routine disponibili all'utente

- yylex() routine di scanning
- yytext stringa con la quale avviene il match
- yyin input file (default: stdin)
- yyout output file (default: stdout)
- yyleng lunghezza di yytext
- unput(c) rimette il car c nella stringa dei simboli da esaminare
- input() legge il carattere successivo
- …

Esempio 2

- Costruire uno scanner che conta il numero di caratteri e il numero di linee dell'input:

```
/* Nella sezione definitions i commenti vengono copiati nel file di
   output*/
%{
    int num_lines = 0, num_chars = 0;
%}
%option noyywrap
%%
\n      {++num_lines; ++num_chars;}
.       {++num_chars;}
%%
int main(void)
        {
        yylex();
        printf( "# of lines = %d, # of chars = %d\n",
                num_lines, num_chars );
        return 0;
        }
```

Esempio 3

- Costruire uno scanner che conta il numero di caratteri e il numero di linee di un file dato in input:

```
%{
    int num_lines = 0, num_chars = 0;
%}
%option noyywrap
%%
\n      ++num_lines; ++num_chars;
.       ++num_chars;

%%
int main(int argc, char *argv[])
            {
    --argc;  /* skip over program name */
    if ( argc > 0 )
            yyin = fopen(argv[1], "r" );
    else
            yyin = stdin;

    yylex();
    printf( "# of lines = %d, # of chars = %d\n",
                num_lines, num_chars );
    return 0;
         }
```

Primi esercizi

- Scrivere un programma in Flex che :
  1. raddoppi tutte le occorrenze delle vocali in un file
  2. conti le occorrenze dei numeri multipli di 3 e dei multipli di 5
  3. comprima gli spazi e i tab in un unico singolo blank, e rimuova quelli alla fine di una linea.
  4. Elimini il testo inserito tra { e }
  5. Elimini il testo inserito tra { e } su un'unica linea
  6. Mantenga solo le linee che finiscono o cominciano con una consonante, cancellando le altre.
  7. Converta tutte le lettere maiuscole di un file in minuscole, ad eccezione di quelle racchiuse tra " e "
  8. Restituisca il più piccolo dei numeri naturali presenti nel testo
  9. Conti le occorrenze dei numeri pari e di quelli dispari

Esercizio 1

- Scrivere un programma in Flex che raddoppi tutte le occorrenze delle vocali in un file

```
%option main

vocali    [aeiouAEIOU]
%%
{vocali}      {ECHO;ECHO;}
```

Esercizio 2

- Scrivere un programma in Flex che conti le occorrenze dei numeri multipli e dei multipli di 5

```
    int conta_3=0, conta_5=0, num;
%option noyywrap

nat 0|[1-9][0-9]*

%%
{nat}       {num=atoi(yytext);
        if (num%3==0)
            conta_3++;
        if (num%5==0)
            conta_5++;}
\n      ;
.       ;
%%
int main(void){
    yylex();
    printf("il numero dei numeri multipli di 3 è:
%d",conta_3);
    printf("il numero dei numeri multipli di 5 è:
%d",conta_5);
    return 0;
}
```