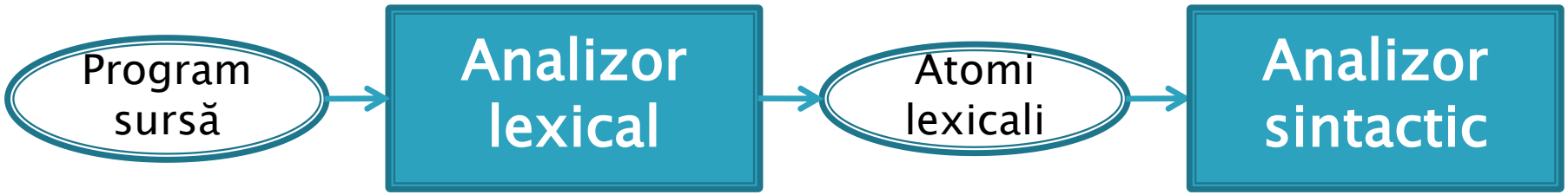
Limbaje formale și tehnici de compilare

Curs 4: analiza lexicală; definiții regulate; diagrame de tranziții

Analiza lexicală

- Analiza lexicală este prima fază a translatării (compilării)
- Ea se realizează de către **analizorul lexical** (ANLEX)
- ANLEX primește la intrare textul sursă la programului sub forma unui șir de caractere și va genera un șir de **atomi lexicali** (**tokens**) corespondent. Atomii lexicali produși vor fi folosiți în faza următoare, analiza sintactică




```
// valoarea absolută
int abs(int v){
    if(a<0)return -a;
    return a;
    }
```

```
TYPE_INT ID:abs LPAR TYPE_INT
ID:v RPAR LACC IF LPAR ID:a LESS
INT:0 RPAR RETURN SUB ID:a
SEMICOLON RETURN ID:A
SEMICOLON RACC
```

Sarcinile analizei lexicale

- Obține șirul de atomi lexicali corespondent codului sursă
- Elimină construcțiile care nu mai sunt necesare în fazele ulterioare: comentarii, spații, linii noi, …
- Reține locațiile (linie, coloană) din codul sursă corespunzătoare atomilor lexicali pentru a se genera ulterior, dacă este nevoie, mesaje de eroare localizate
- Realizează unele transformări din caractere în alte formate, cum ar fi: constantele numerice → tipuri numerice, secvențe escape → coduri corespunzătoare, …
- Separă cuvintele cheie de identificatori

```
3.14159265358979323846 → double
"Ion\nAna" → char []={'I','o','n','\r','\n','A','n','a','\0'}   // Windows
```

Avantajele analizei lexicale

- Permite simplificarea gramaticii sintactice și implicit a fazei de analiză sintactică
- De obicei ANLEX poate fi implementat cu limbaje simple (limbaje regulate), ceea ce mărește viteza de procesare
- Distribuie complexitatea verificării sintactice în mai multe module, care pot fi implementate independent

```
// gramatică folosind o singură fază
comentariu = ...
spatiu = ' ' | '\t' | '\n'
skip = ( comentariu | spatiu )*
instr = 'while' skip '(' skip expr skip ')' skip instr skip
```

```
// analiza lexicală: elimină skip
comentariu = ...
spatiu = ' ' | '\t' | '\n'
skip = ( comentariu | spatiu )*
```

```
// analiza sintactică
instr = 'while‘ '(‘ expr ')‘ instr
```

Atomi lexicali (tokens)

- Unități primare de informație, care sunt formate pe baza caracterelor din programul sursă
- Constituie alfabetul de intrare al analizorului sintactic
- Atomii reprezintă diverse constituente ale unui LP:
  - identificatori: tmp, i, x0, A, main, printf
  - cuvinte cheie (keywords): int, void, for, return
  - constante numerice (întregi și reale): 0, 108ll, 0.5, 93f
  - constante caracter: '#', '\\', '\x5C'
  - constante șir de caractere: "salut\n", "#", ""
  - operatori: + -- && ^=
  - separatori/delimitatori: ; { } ,
- În funcție de locul lor, unii atomi pot reprezenta diverse constituente. De exemplu, parantezele pot fi atât separatori cât și operatori: int f(int x){…}, k*f(x)

Componentele atomilor lexicali

- **cod/tip** – de obicei constante numerice, care arată despre ce atom este vorba: ID, TYPE_INT, INT, LPAR, RPAR, ...
- **poziție în codul sursă** – toate informațiile care localizează originea atomului în sursă: linie, coloană, nume fișier
- **informații lexicale asociate** – caracterele din care sunt compuși identificatorii sau constantele caracter/șir de caractere, valorile constantelor numerice

```
typedef struct{
    int linie, coloana;
    char *numeFis;
    }Pozitie;
```

```
enum{ID, TYPE_INT, INT, LPAR, RPAR};
typedef struct{
    int cod;
    Pozitie poz;
    union{
        double r;       // numere reale
        long i;         // numere intregi
        char c;         // caractere
        char *text;     // identificatori, siruri de caractere
        };
    }Atom;
```

Exemplu de extragere atomi lexicali

```
// aria cercului
int main(){
    double r;              // raza
    printf("raza: ");
    scanf("%g",&r);
    printf("aria=%g\n",3.14*r*r);
    return 0;
    }
```

Linie	Atomi
2	TYPE_INT, ID:main, LPAR, RPAR, LACC
3	TYPE_DOUBLE, ID:r, SEMICOLON
4	ID:printf, LPAR, STR:"raza: ", RPAR, SEMICOLON
5	ID:scanf, LPAR, STR:"%g", COMMA, AND, ID:r, RPAR, SEMICOLON
6	ID:printf, LPAR, STR:"aria=%g\n", COMMA, DOUBLE:3.14, MUL, ID:r, MUL, ID:r, RPAR, SEMICOLON
7	RETURN, INT:0, SEMICOLON
8	RACC

Definirea atomilor lexicali

- Atomii lexicali se definesc în cadrul **gramaticii lexicale**. Această gramatică cuprinde doar definițiile atomilor lexicali, specificându-se și care atomi trebuie eliminați (ex: spații, comentarii, …)
- Pentru marea majoritatea a LP, definirea atomilor lexicali se poate realiza folosind doar limbaje regulate. În această situație se va folosi o formă ușor modificată a expresiilor regulate, numită **definiții regulate**
- În cazul unor LP care au reguli mai complexe de definire a atomilor lexicali (ex: comentarii imbricate), limbajele regulate nu mai sunt suficiente și atunci este nevoie de un formalism mai puternic (ex: gramatici independente de context – GIC)

Definiții regulate

- Definițiile regulate (DR) se obțin din expresii regulate (ER), prin următoarele modificări:
- Fiecărei ER i se va atribui un nume. Toate aceste nume formează o mulțime $\{d_1, d_2, \ldots d_n\}$
- Spațiile nu mai sunt semnificative
- În interiorul DR pot să apară numele altor DR. Pentru a se evita recursivitatea, în interiorul DR_i vor putea apărea doar numele din mulțimea $\{d_1, d_2, \ldots d_{i-1}\}$
- Pentru a se face distincția între nume de DR și caractere obișnuite, caracterele vor fi puse în clase de caractere sau între apostroafe sau ghilimele. Din acest motiv și apostroafele și ghilimelele devin metacaractere.

```
WHILE = 'while'
ESC = [\\] [abfnrtv'?"\\0]
CHAR = ['] ( ESC | [^'\\] ) [']
STRING = ["] ( ESC | [^"\\] )* ["]
```

Diagrame de tranziții (DT)

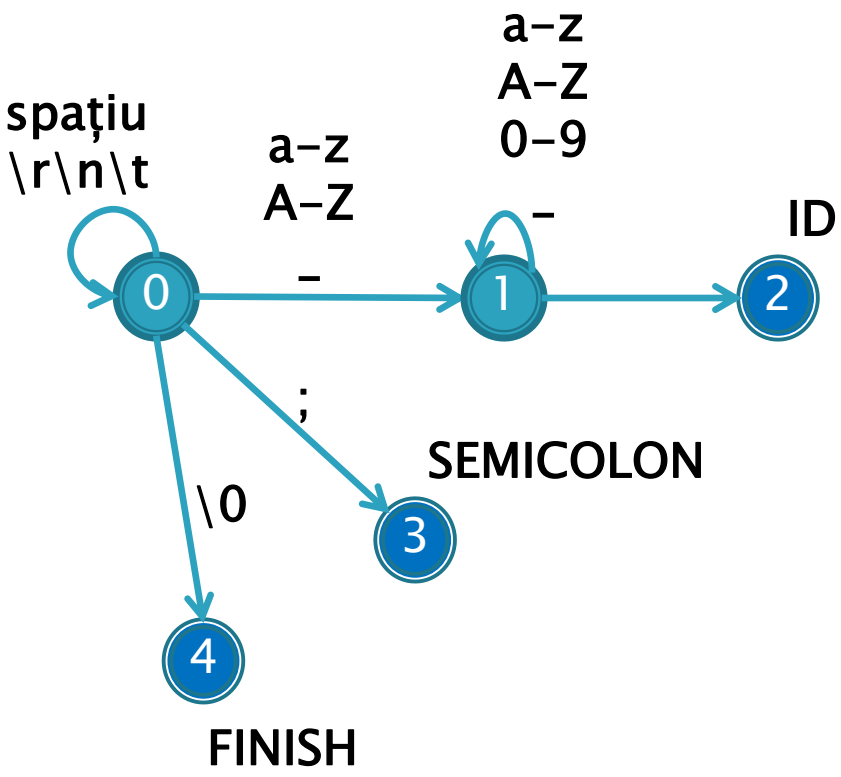
- Sunt o modalitate grafică de reprezentare a atomilor lexicali, folosind **automate cu stări finite** (ASF)
- ASF se pot folosi atunci când atomii lexicali se definesc folosind gramatici regulate, deoarece acestea au aceeași putere expresivă ca și ASF
- O DT are două tipuri de tranziții:
  - **tranziții care consumă caractere** – aceste tranziții consumă fiecare câte un caracter/clasă de caractere. Caracterul se notează punându-se pe tranziția respectivă.
  - **tranziții care nu consumă caractere** (tranziții **epsilon**) – aceste tranziții nu se notează cu nimic și singurul lor efect este trecerea în starea următoare

Funcționarea DT

- În șirul de caractere de intrare (codul sursă) există un cursor care indică poziția curentă
- Se începe întotdeauna din starea inițială (0)
- Din starea curentă se caută o tranziție care poate consuma caracterul curent. Dacă există o asemenea tranziție, se consumă caracterul curent (se avansează cursorul la caracterul următor) și se trece în starea indicată de tranziție
- Dacă din starea curentă nu există nicio tranziție care poate consuma caracterul curent, dar există o tranziție epsilon, se trece în starea indicată de aceasta fără a consuma caracterul curent (tranzițiile epsilon se comportă ca **else** pentru **if**).
- Dacă nu există în starea curentă nici tranziții care consumă caracterul curent și nicio tranziție epsilon, se raportează eroare
- Când s-a ajuns într-o stare finală, se adaugă atomul găsit la lista de atomi și se revine în starea inițială
- Pentru DR care se elimină (ex: spații), starea finală este 0

Exemplu funcționare DT

```
ID = [a-zA-Z_] [a-zA-Z0-9_]*
SEMICOLON = ';'
FINISH = '\0'
SKIP = [ \r\n\t]      // se elimină
```



tmp;

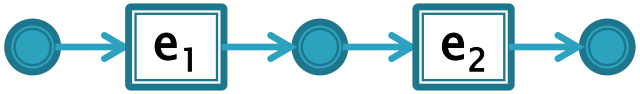
Caracter curent	Stare curentă	Stare următoare
t	0	1
m	1	1
p	1	1
;	1	2 (=>ID)
;	0	3 (=>SEMICOLON)
\0	0	4 (=>FINISH)

Algoritm pentru construcția DT

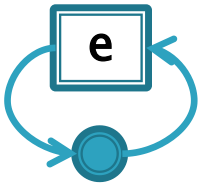
- Fiecare DR pornește din starea 0 și are o singură stare finală în care se ajunge la sfârșitul DR
- Componentele unei DR se reprezintă grafic conform regulilor de mai jos
- Se continuă recursiv până când există doar caractere/clase pe tranziții

c – caracter sau clasă de caractere

$e_1\ e_2$

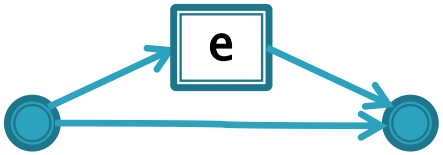


e*

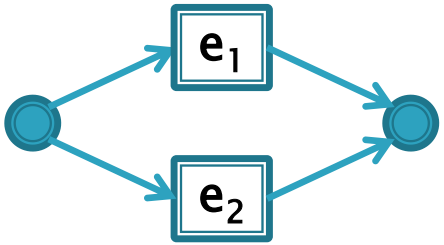


e+ → e e*

e?



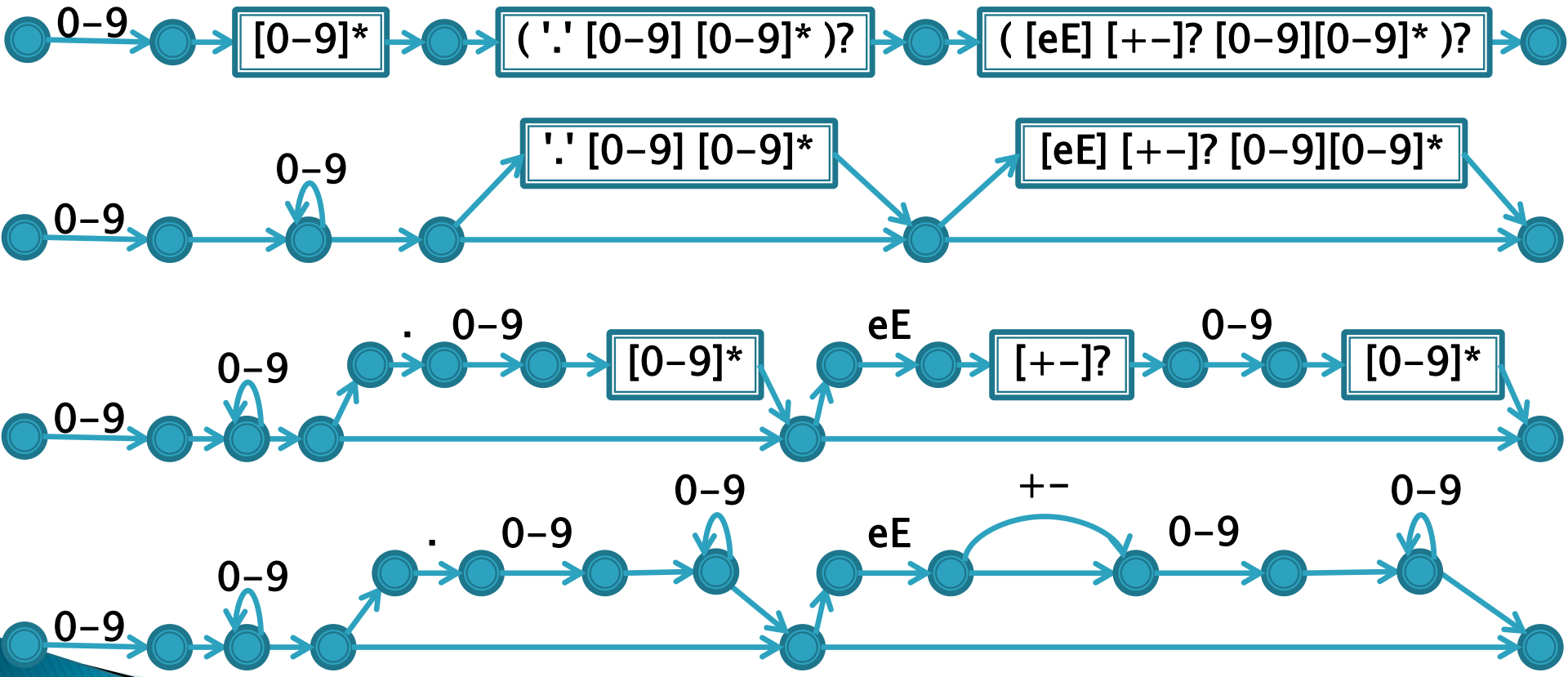
$e_1 \mid e_2$



Exemplu construcție DT (1)

NR = [0–9]+ ('.' [0–9]+)? ([eE] [+–]? [0–9]+)?

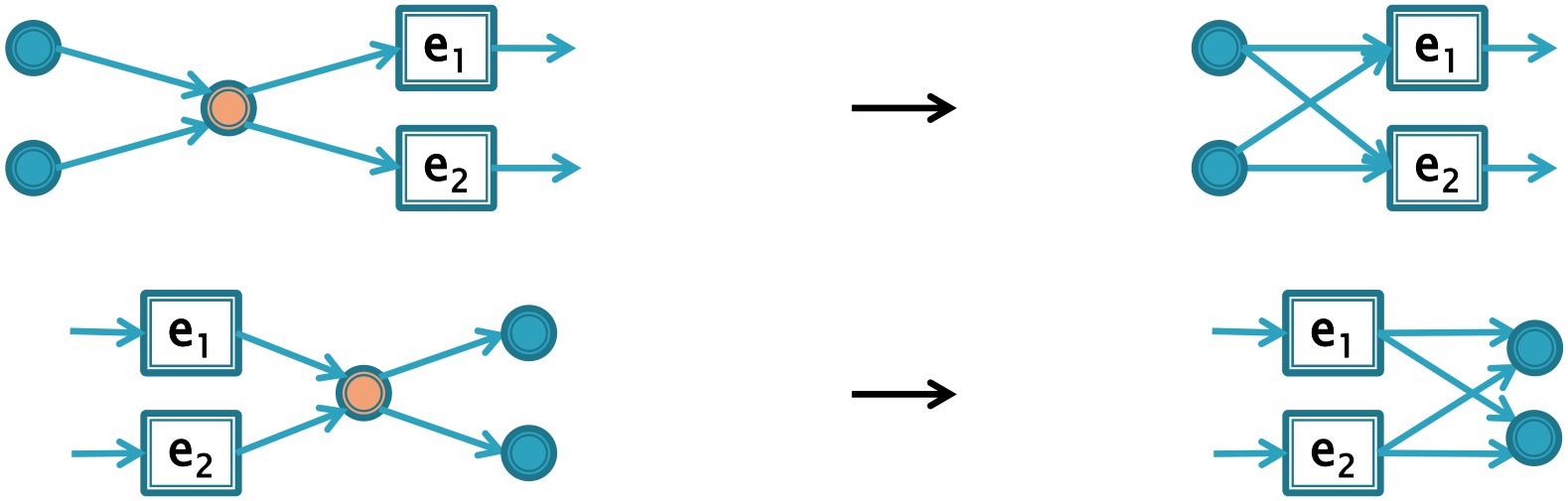
[0–9] [0–9]* ('.' [0–9] [0–9]*)? ([eE] [+–]? [0–9][0–9]*)?



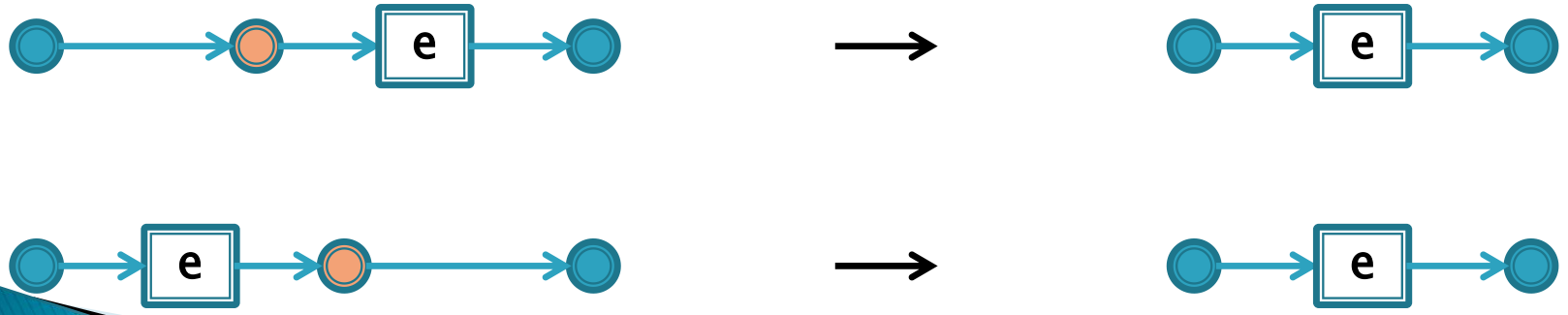
108; 3.14; 52e–1; 9.02E7

Optimizări DT

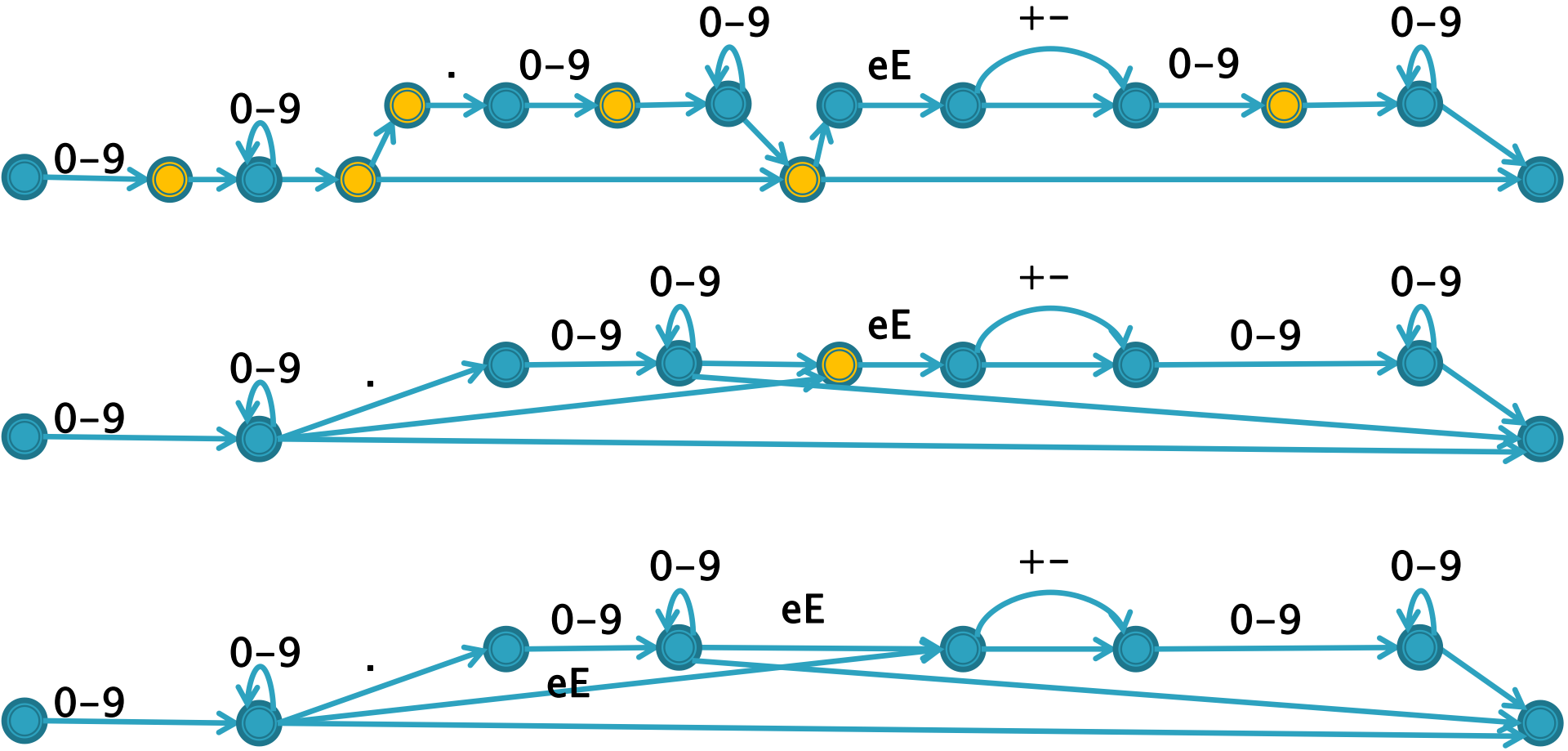
- Dacă într-o stare intermediară intră sau ies doar tranziții epsilon, acea stare se poate elimina



cazuri particulare: o intrare și o ieșire



Exemplu optimizare DT



NR = [0-9]+ ('.' [0-9]+)? ([eE] [+-]? [0-9]+)?

108; 3.14; 52e-1; 9.02E7

Nedeterminări în DT

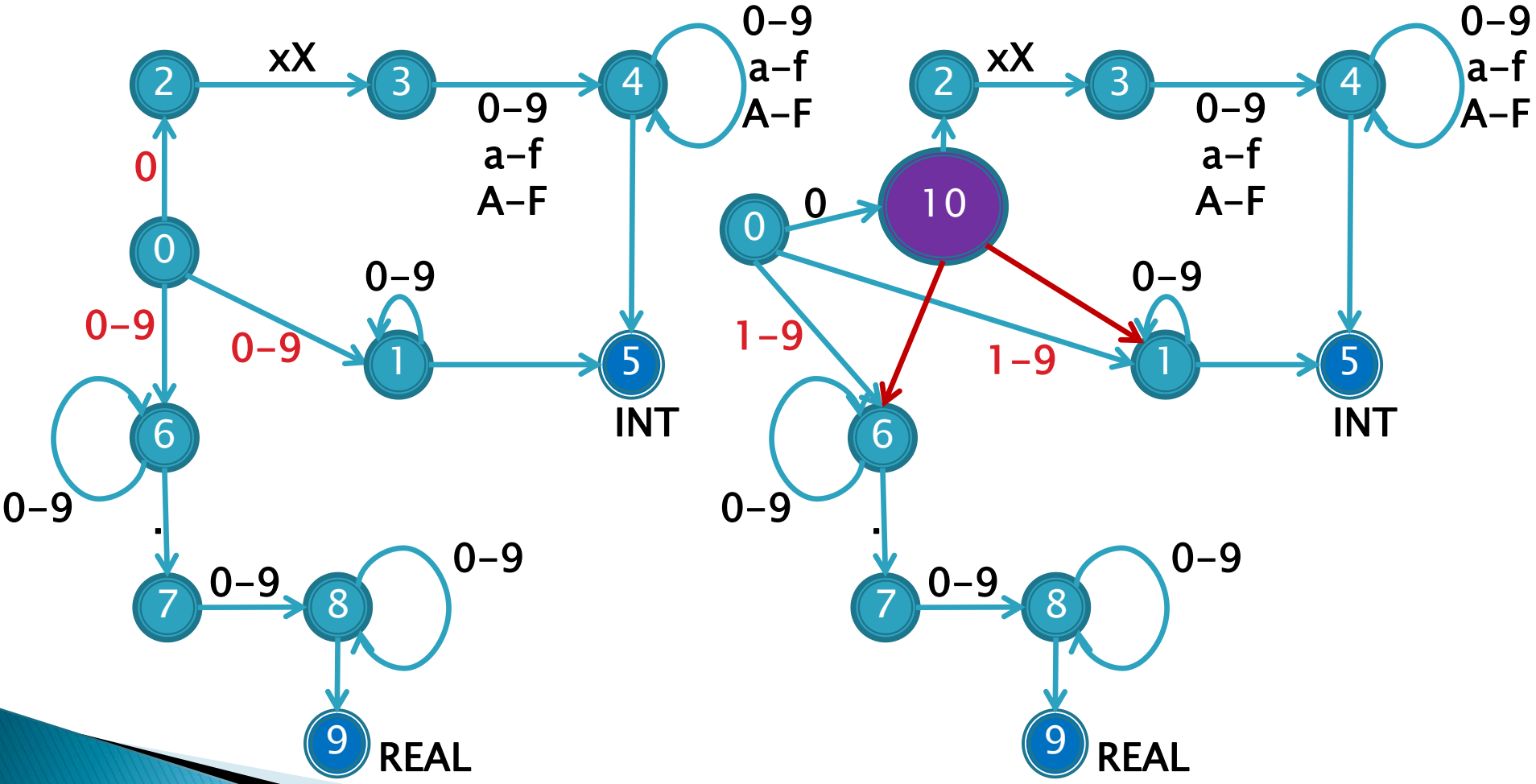
- Considerăm o stare S din care pleacă două tranziții, T_1 și T_2
- Fie M_1, M_2 mulțimile caracterelor consumate respectiv de T_1 și T_2
- Fie $M = M_1 \cap M_2$
- Dacă $M \neq \emptyset$, atunci înseamnă că există caractere comune în M_1 și M_2. În această situație, DT nu mai poate funcționa conform algoritmului prezentat anterior, deoarece în starea S, dacă apare un caracter $c \in M$, atunci nu se știe pe care tranziție (T_1 sau T_2) se va merge mai departe
- Dacă dintr-o stare pleacă două sau mai multe tranziții epsilon, apare și atunci o nedeterminare, deoarece nu se va ști pe care dintre tranziții se va merge
- Asemenea nedeterminări trebuie rezolvate, astfel încât dintr-o stare să nu plece mai multe tranziții care consumă aceleași subseturi de caractere sau mai multe tranziții epsilon

Eliminarea nedeterminărilor

- Dacă dintr-o stare pornesc mai multe tranziții care consumă caractere comune, se creează o stare intermediară la care se ajunge din starea originară consumând caracterele comune. Dacă sunt mai multe subseturi de caractere comune, se alege acela care apare pe cele mai multe tranziții. Din starea intermediară se continuă cu tranziții epsilon către stările destinație ale tranzițiilor originare. Se optimizează aceste tranziții. De la tranzițiile originare se elimină subsetul comun.
- Dacă dintr-o stare pornesc mai multe tranziții epsilon, se urmărește eliminarea lor, propagând în locul lor tranzițiile care le urmează în mod direct sau la care se poate ajunge fără să se consume caractere.
- Dacă dintr-o stare pornesc doar tranziții epsilon, iar acestea duc către stări care au bucle ce consumă caractere comune, se pot transfera aceste bucle la starea de pornire.

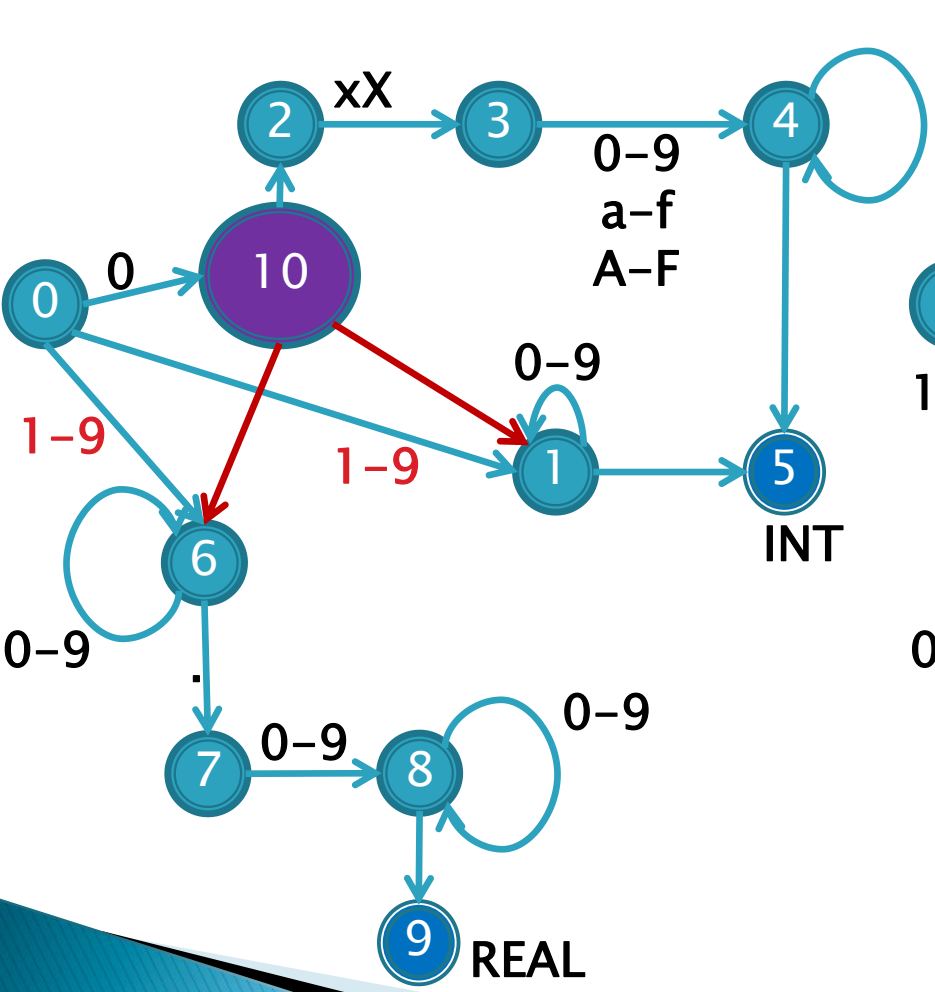
Exemplu eliminare nedeterminări

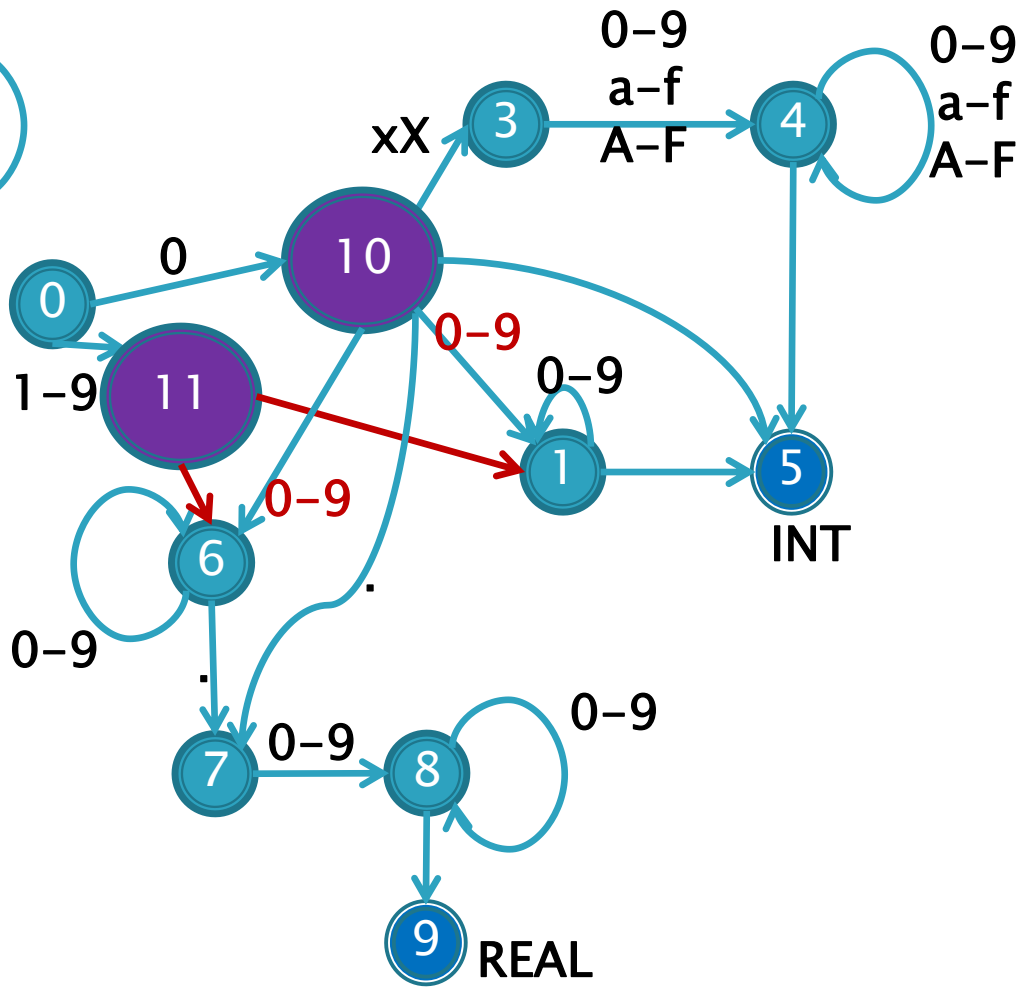
INT = [0-9]+ | '0' [xX] [0-9a-fA-F]+
REAL = [0-9]+ '.' [0-9]+



Exemplu eliminare nedeterminări

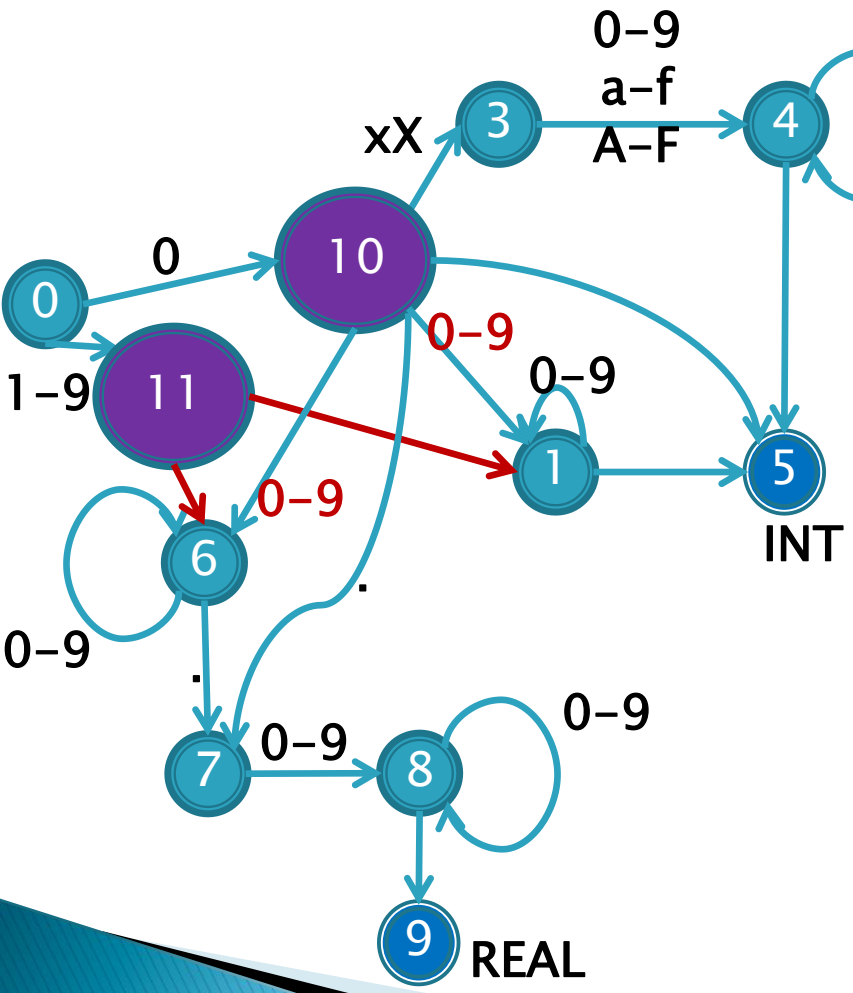
INT = [0–9]+ | ‘0’ [xX] [0–9a–fA–F]+
REAL = [0–9]+ ‘.’ [0–9]+

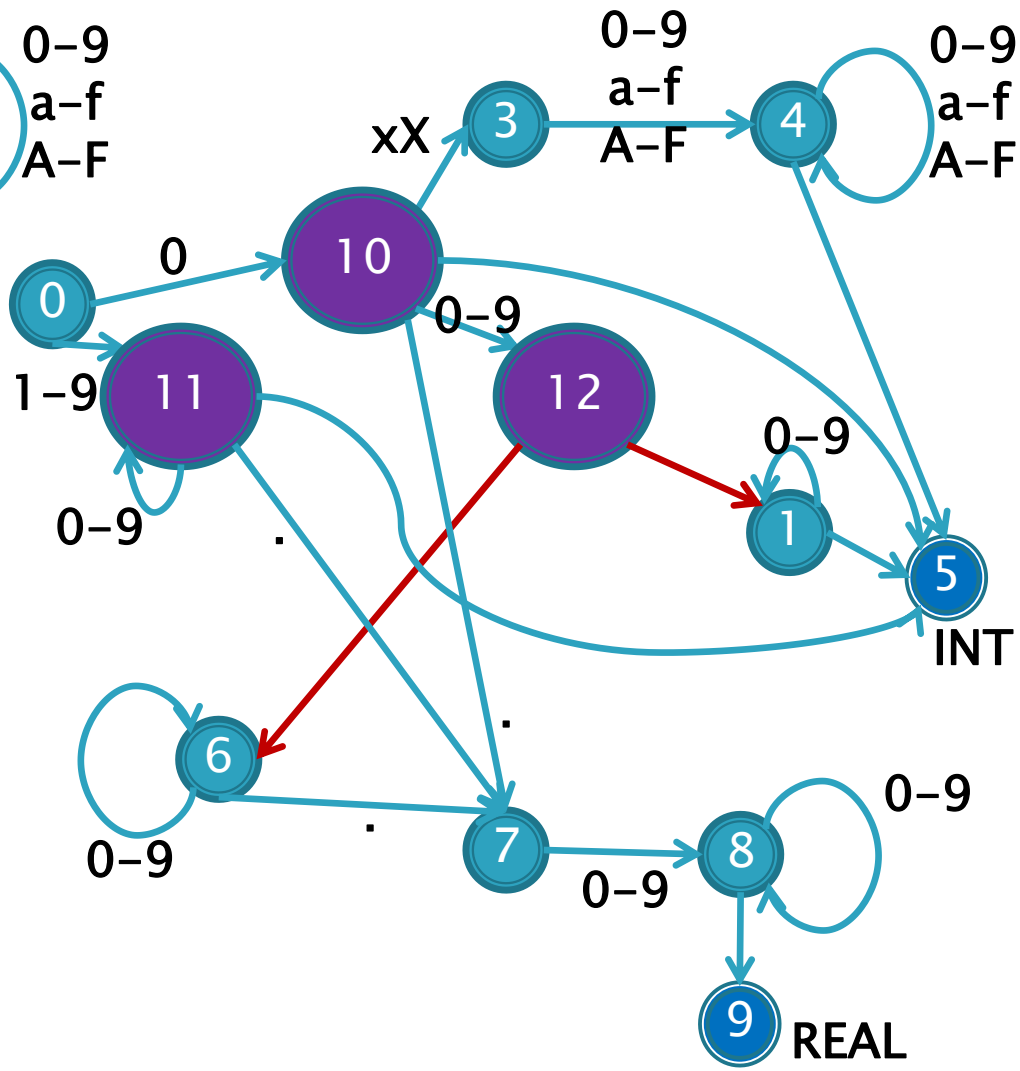




Exemplu eliminare nedeterminări

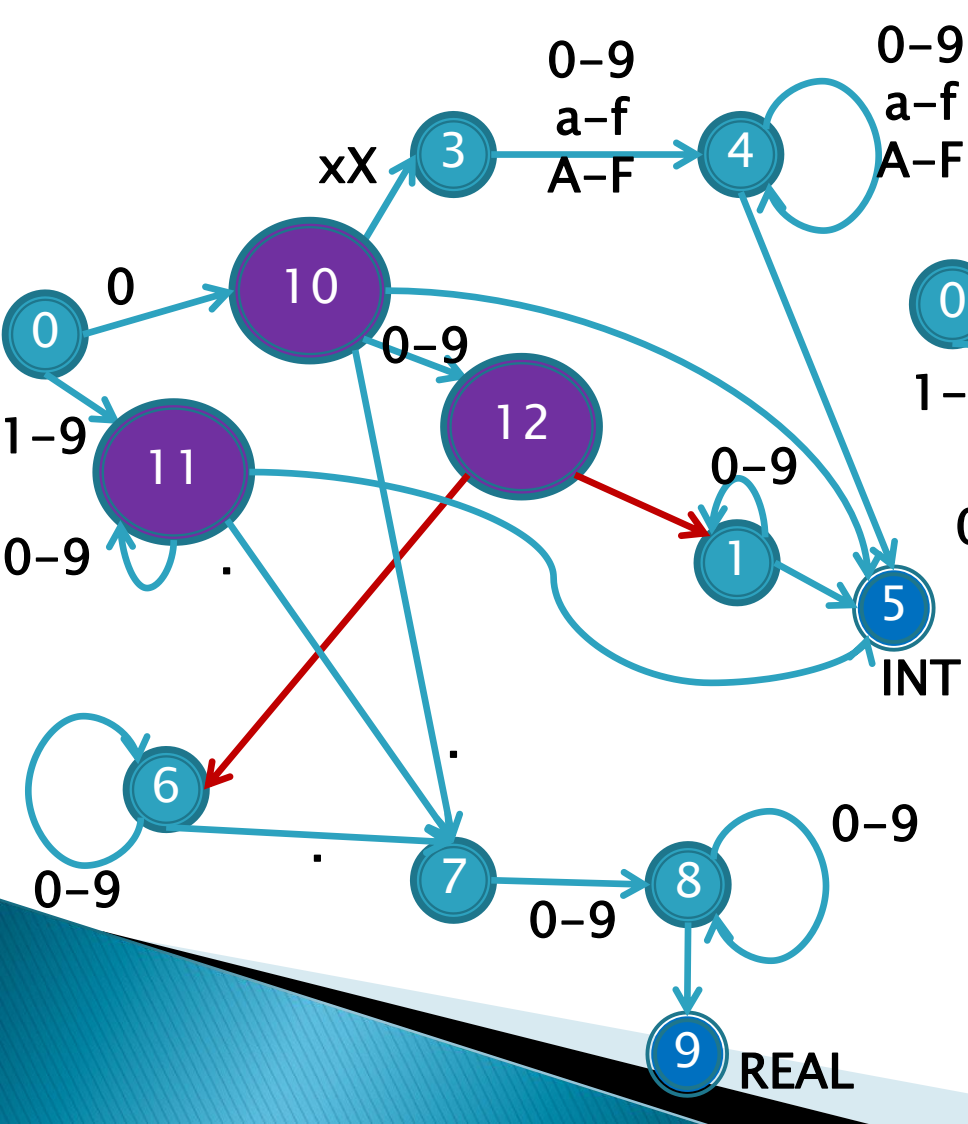
INT = [0–9]+ | ‘0’ [xX] [0–9a–fA–F]+
REAL = [0–9]+ ‘.’ [0–9]+

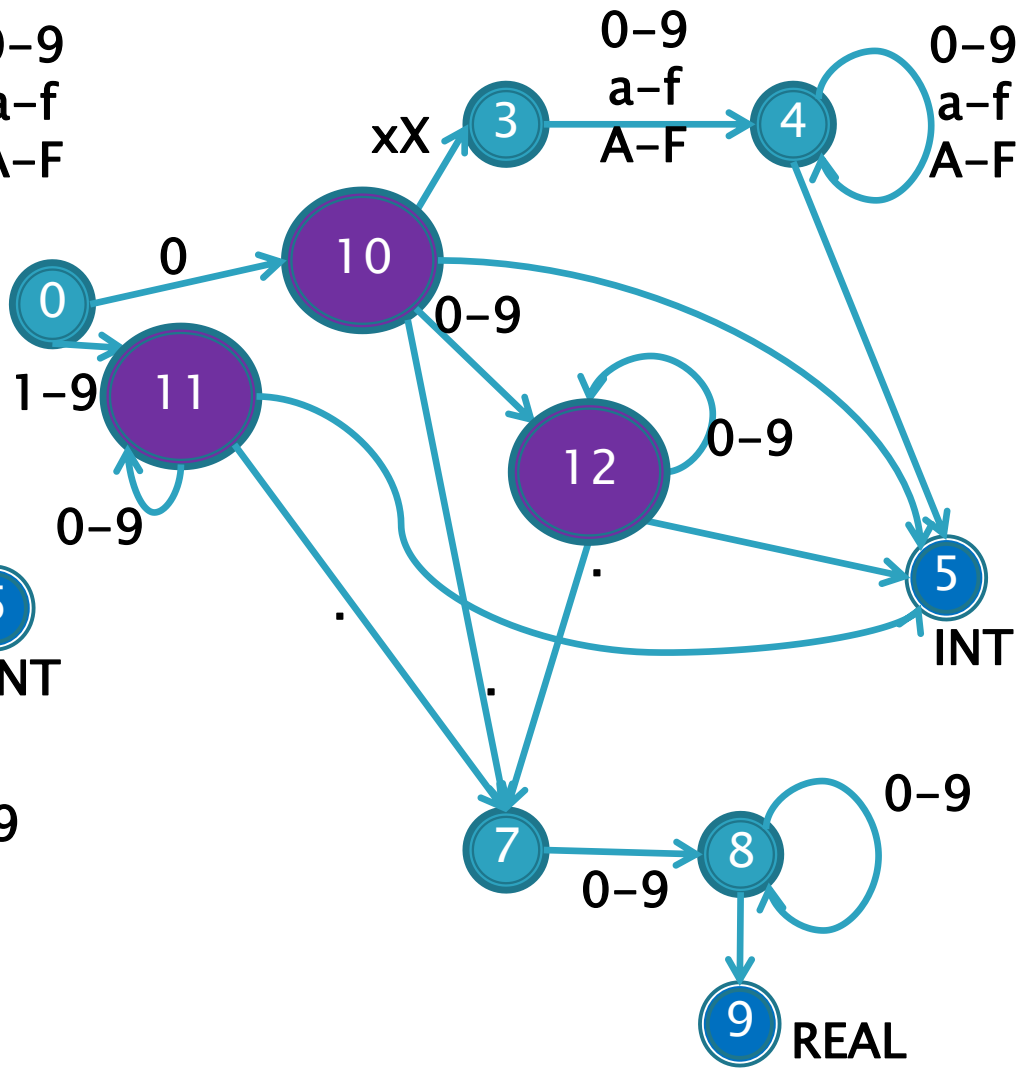




Exemplu eliminare nedeterminări

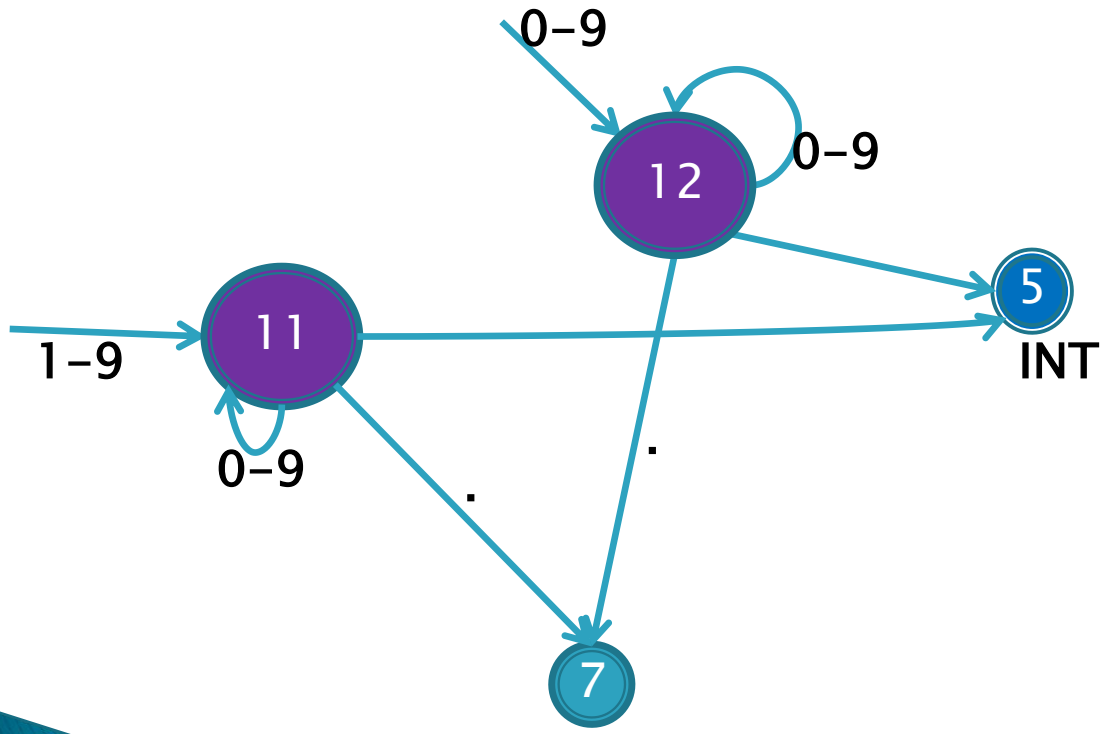
INT = [0–9]+ | ‘0’ [xX] [0–9a–fA–F]+
REAL = [0–9]+ ‘.’ [0–9]+

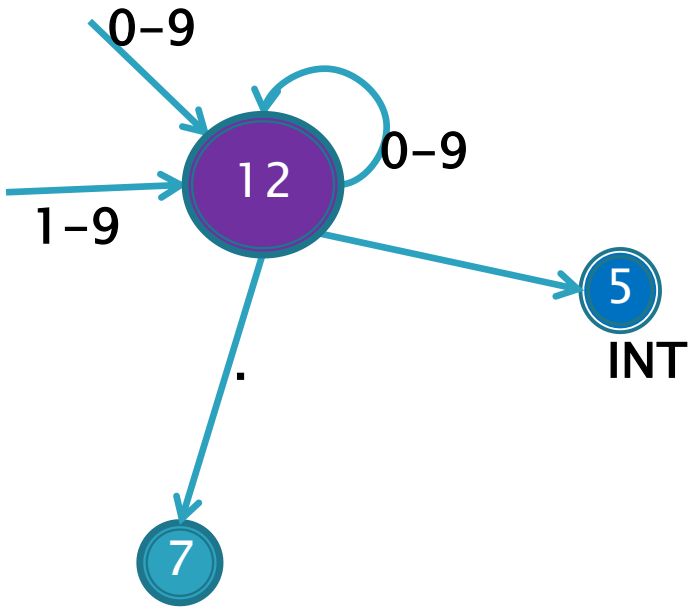




Optimizare stări cu ieșiri identice

- Dacă două stări au ieșiri identice (tranzițiile care pleacă din ele sunt identice și conduc în aceleași stări destinație), se poate renunța la una dintre ele. Intrările stării la care se renunță se vor redirecționa către cealaltă stare.





Exemplu eliminare nedeterminări

INT = [0-9]+ | '0' [xX] [0-9a-fA-F]+
REAL = [0-9]+ '.' [0-9]+