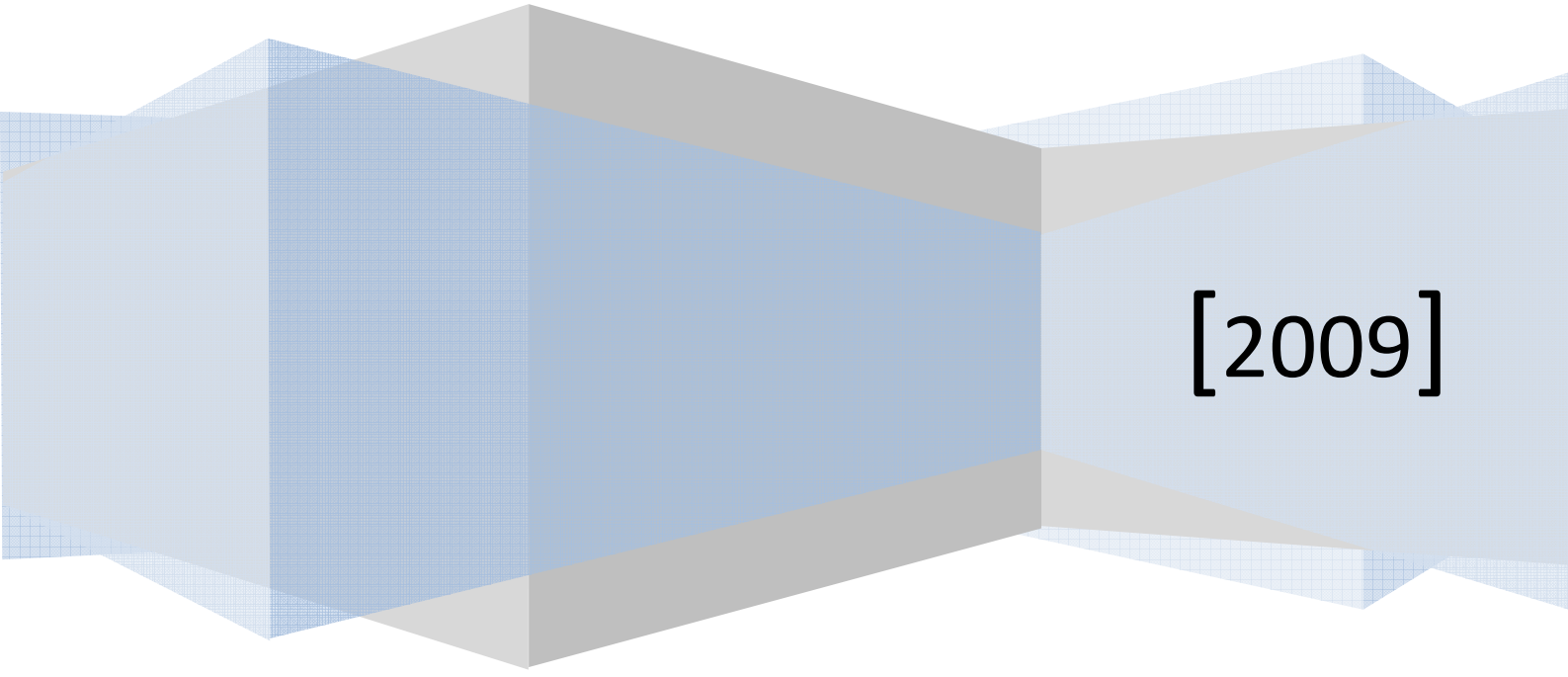
Universitatea POLITEHNICA Bucuresti

Proiectarea Automatelor Deterministe

Alexandru Stefanescu

FILS 1241E



[2009]

CUPRINS

PROIECTAREA AUTOMATURILOR DETERMINISTE

I. INTRODUCERE

Tema prezentarii este **sinteza unui automat determinist minim cu un limbaj acceptat dat**. Pentru acest lucru, am realizat un **program** prin care se pot sintetiza si analiza grafuri si automate finite. Acesta implementeaza un algoritm propriu de generare a unui automat nondeterminist pe baza unui limbaj dat si algoritmi pentru gasirea automatului determinist minim echivalent. Programul asigura o interfata pentru desenarea intuitiva a grafurilor/automatelor cu ajutorul mouse-ului si module pentru proiecterea si conversia lor, precum si posibilitatea de calcul a diferitelor marimi care li se pot asocia.

II. NOTIUNI

Limbaje si expresii regulate

Un automat este un instrument de recunoastere a unui limbaj. Datele de intrare sunt reprezentate de un „string" apartinand unui anumit alfabet si rezultatul este apartenenta sau nu a stringului la limbaj. Un **alfabet** poate fi orice multime finita; un **string (cuvant)** este o succesiune de simboluri din alfabet; iar un **limbaj** este o *submultime* de stringuri pe un alfabet. Un limbaj este dat sub forma unei *expresii regulate* formata din stringuri si **operatorii**: „+"-concatenare (operator implicit prin omisiune), „U"-reuniune, „*"-inchiderea.

Un exemplu de limbaj pe alfabetul {a, b, c} este (a U b)c*, ceea ce inseamna orice sir de simboluri care incepe cu „a" sau „b" (exclusiv), urmat de oricate simboluri „c" (inclusiv niciunul si o infinitate). Stringul vid va fi notat cu *e*.

Automate deterministe si nedeterministe

Un **automat** este o structura definita pe o multime de stari, un alfabet si o functie de tranzitie intre stari (pe baza alfabetului). Printre stari, exista una pe care o vom numi stare initiala („start") si o submultime (posibil vida) de stari finale („finish"), de acceptare. Un limbaj este acceptat daca in urma parcurgerii oricarui string din acesta pornind din starea initiala se ajunge intr-o stare finala.

Automatele se clasifica in doua tipuri: **deterministe si nedeterministe**. Diferenta dintre ele este ca in cazul celor deterministe, starea urmatoare a automatului depinde numai de starea curenta si de simbolul urmator din string; deci se poate prezice evolutia automatului numai pe baza stringului de intrare. Pentru cele nondeterministe, starea urmatoare este numai partial determinata de simbolul urmator; pentru acelasi simbol citit automatul putand sa treaca arbitrar

in mai multe stari. Mai mult, un automat nedeterminist poate citi la un moment dat stringuri, nu doar simboluri. In general, gasirea unui automat determinist finit cu un limbaj acceptat dat este o problema complicata. Aceeasi problema pentru un automat nedeterminist poate fi mai simpla din cauza flexibilitatii crescute ale constructiei.

Reprezentarea grafica a automatelor

Automatele pot fi intuitiv reprezentate cu ajutorul **grafurilor orientate**. Varfurile vor fi starile (cea initiala va fi marcata cu „>", cele finale vor fi incercuite) iar muchiile vor avea asociat un simbol sau string care marcheaza faptul ca se poate trece din varful/starea sursa a muchiei in cel de destinatie in urma „citirii" acelui simbol sau string.

Reprezentarea grafica permite recunoasterea usoara a unui automat determinist: fiecare muchie contine un singur simbol din alfabet si din orice varf porneste cate o muchie pentru fiecare simbol din alfabet. Cele nondeterministe pot avea de exemplu muchii continand stringuri sau sirul vid *e*.

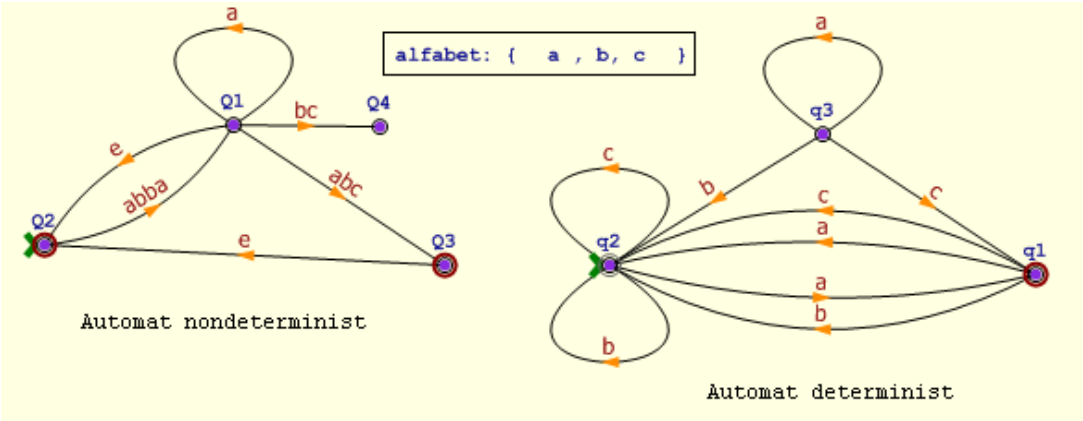


Figura 1

In exemplele din Figura 1, ambele automate sunt create pe acelasi alfabet. De remarcat este ce se intampla in urma citirii sirului „abbaabc". Automatul determinist va ajunge in mod sigur inapoi in starea initiala trecand prin q2-q1-q2-q2-q1-q2-q2-q2 pe cand cel nondeterminist poate ajunge intr-una din starile {Q2, Q3, Q4} in conformitate cu drumul parcurs Q2-Q1-Q3-Q2, Q2-Q1-Q3 si respectiv Q2-Q1-Q1-Q4.

Echivalenta automatelor

Doua automate sunt **echivalente** daca au acelasi limbaj acceptat. Un rezultat interesant este ca pentru orice automat nondeterminist se poate gasi unul determinist echivalent. Acesta este si unul dintre scopurile lucrarii mele.

In exemplul din Figura 2, se poate constata ca toate cele 4 automate au *acelasi* limbaj acceptat – deci sunt echivalente. Dintre acestea, numai /este determinist.

Un automat **minim** are cel mai mic numar de stari dintre automatele echivalente cu el.

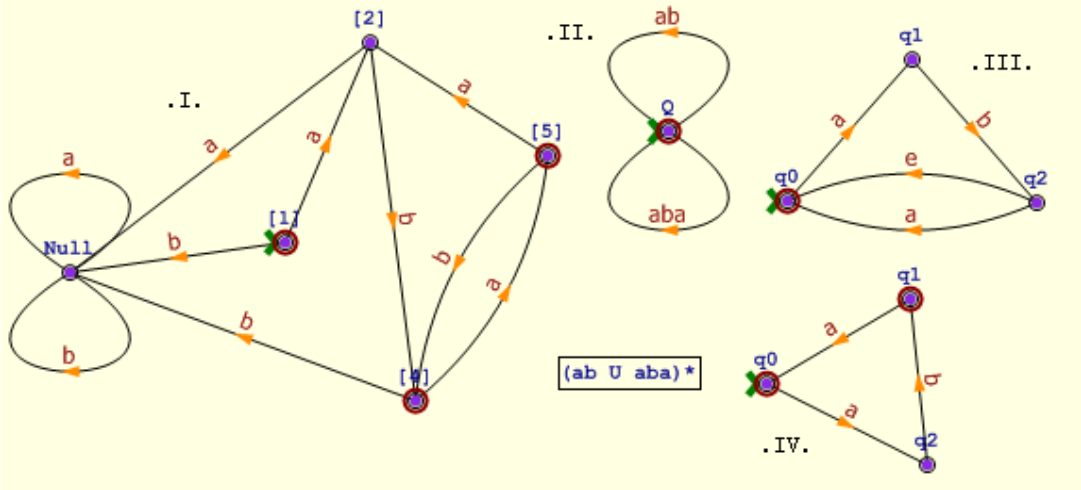


Figura 2

III. ALGORITMI FOLOSITI

Programul permite obtinerea unui automat determinist minim pe baza unui limbaj acceptat dat. Acest lucru este realizat in **trei etape**: mai intai se creeaza un automat nedeterminist cu acest limbaj, apoi se gaseste unul determinist echivalent si in final se minimizeaza.

Sinteza automatului nedeterminist

Algoritmul propriu de sinteza a unui automat nedeterminist numai pe baza unui limbaj acceptat dat se foloseste de **arborele sintactic** al limbajului, acesta fiind o expresie regulata, cu operatorii U,*,(,) si + (implicit). Un exemplu de arbore avem in Figura 3. Pentru obtinerea arborelui, mai intai se individualizeaza componentele expresiei in operatori si operanzi – stringurile, si apoi se creeaza legaturi intre acestia cu ajutorul a doua „stive” (de operanzi si operatori), pe baza prioritatii operatorilor (*-maxima, +, U-minima).

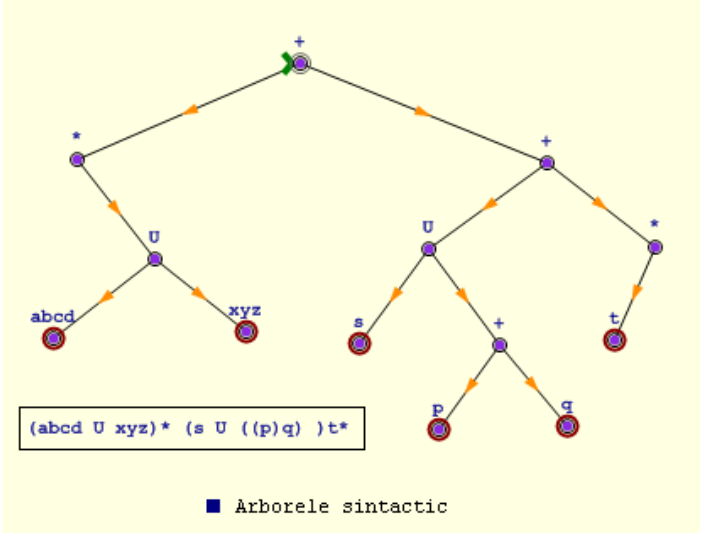


Figura 3

Constructia automatului nondeterminist se face printr-o functie recursiva ce primeste ca parametri doua varfuri si o celula din arborele sintactic al expresiei. Aceasta adauga varfuri sau muchii intre cele doua varfuri in functie de tipul celulei, operator sau string (dupa cum se poate vedea in Figura 4), si se autoapeleaza, dupa caz. La inceput, se creeaza un varf de start si unul de finish (care va fi si singurul) si se apeleaza functia pentru aceste doua varfuri si radacina arborelui sintactic.

Sinteza automatului determinist

Algoritmul de **conversie automat nondeterminist – determinist** are la baza ideea inlocuirii mai multor stari din automatul initial cu una singura in automatul final. Mai intai, se elimina tranzitiile multiple, adica cele bazate pe stringuri, adaugand stari suplimentare si tranzitii de un sigur simbol pentru fiecare simbol din stringul tranzitiei multiple. Apoi, pentru fiecare varf q se afla $E(q)$, adica multimea starilor initiale in care se poate ajunge din q trecand numai prin stringul nul, e. Starile din automatul final vor fi o reuniune de aceste multimi. Pentru fiecare stare finala Q nou creata si fiecare litera c din alfabet se afla $\delta(Q, c)$ ca reuniunea $E(q)$ unde q este o stare initiala necuprinsa in Q in care se poate ajunge din oricare din starile initiale cuprinse in Q trecand numai prin litera c. O stare din automatul determinist este de finish daca contine o stare de finish din automatul nedeterminist.

Formal:

- Automat determinist $D = (Q, \Sigma, \delta, s, F)$ *unde*
  - Q multime finita de stari
  - Σ alfabet
  - $s \in Q$ stare initiala
  - $F \subseteq Q$ stari finale
  - $\delta : Q \times \Sigma \rightarrow Q$ functia de tranzitie

- Automat nedeterminist $N = (K, \Sigma, \Delta, s, F)$ unde
  - K multime finita de stari
  - Σ alfabet
  - $s \in K$ stare initiala
  - $F \subseteq K$ stari finale
  - $\Delta \subseteq K \times \Sigma^* \times K$ relatie de tranzitie

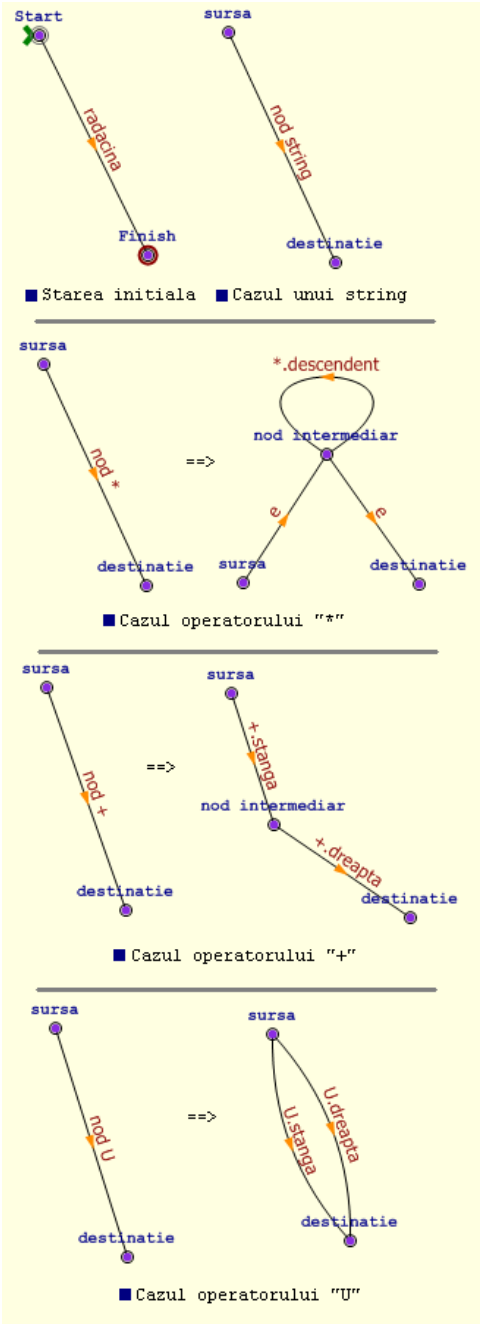


Figura 4

- $E(q) = \{q\} \cup \{p \in K'; exista\ tranzitii\ din\ p\ in\ q\ numai\ prin\ sirul\ vid\}$ unde *K'* este multimea starilor *K* din automatul nedeterminist la care se aduaga starile obtinute prin elimiarea tranzitiilor multiple
- $\delta(Q,a) = \bigcup_{q \in K'} \{E(q);\ \exists p \in Q\ astfel\ incat\ (p,a,q) \in \Delta'\}$

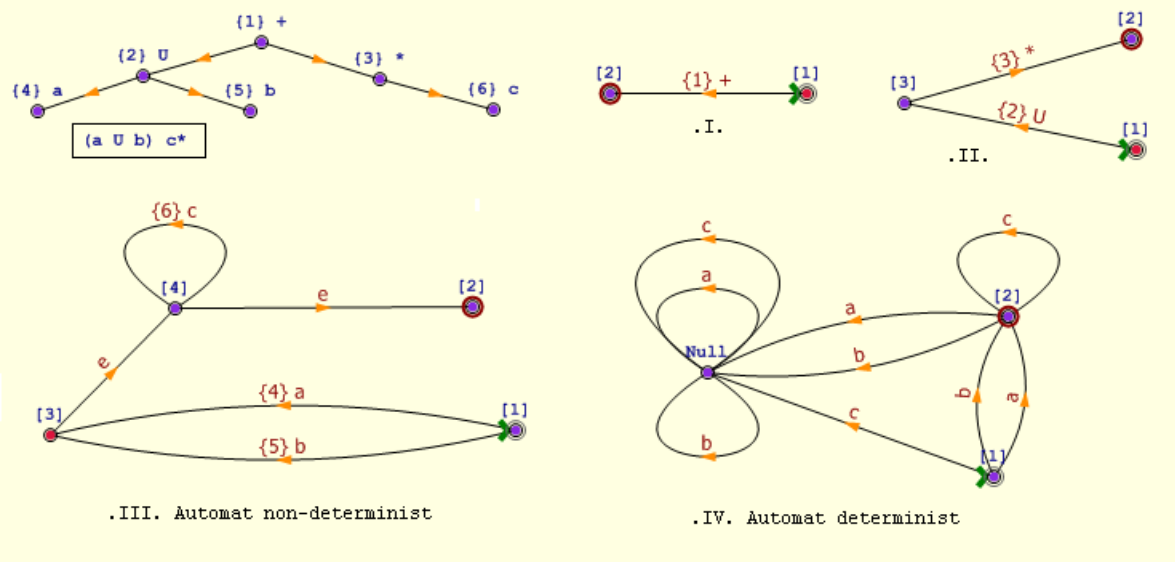


Figura 5

```
int ind = 0; // prima stare finala este singura multime E(q) unde q este varful de start
while (ind < states.Count) // pentru toate starile finale existente ..
{
        current = (ArrayList) states[ind];   // se salveaza starea curenta
        for ( int i = 0; i<alphabet.Length; i++ )   // pentru fiecare litera din alfabet ..
        {
                ArrayList qs = getDelta(current,alphabet[i].ToString()); // se afla δ
                ArrayList reunion = new ArrayList();
                foreach ( Object ov in qs ) //  pentru toate starile initiale din δ ..
                {
                        reunion = concat ( reunion , getEofQ( (Vertex) ov )); // reuniunea E(q)
                }
                int rez = contains(states, reunion); // se verifica daca este o stare noua
                Vertex src = (Vertex) det.vertices[ind];
                Vertex dst;
                if ( rez < 0 )
                {
                        states.Add(reunion);   // daca este noua, se adauga in lista
                        dst = new Vertex(0,0,(reunion.Count==0)?"Null":"");
                        det.vertices.Add(dst);
                }
                else
                {
                        dst = (Vertex) det.vertices[rez];
                }
                det.edges.Add( new Edge(src,dst,alphabet[i].ToString())); // se leaga cele doua stari
        }
        ind++;
}
```

Trebuie mentionat ca niciunul dintre acesti algoritmi nu este optim; primul produce muchii cu sirul vid uneori inutile iar al doilea necesita un timp relativ indelungat de executie si depinde intr-o anumita masura de reprezentarea aleasa a automatului nondeterminist.

Minimizarea automatului determinist

Pentru minimizarea automatului determinist am implementat algoritmul incremental al lui Watson si Daciuk, care spre deosebire de alti algoritmi de minimizare cunoscuti, poate fi oprit intre iteratii, producand automate echivalente intermediare intre cel original si cel minim.

Algoritmul are la baza identificarea starilor echivalente. Doua stari sunt echivalente intr-un automat determinist daca pentru fiecare string citit care duce din prima stare intr-o stare finala, acelasi string duce din a doua stare intr-o stare finala. Clasele de echivalenta ale starilor echivalente formeaza starile automatului minim.

Algoritmul foloseste o functi auxiliara *equiv* care testeaza daca doua stari sunt echivalente. Al treilea parametru, *k*, este folosit numai pentru eficientizarea algoritmului, prin specificarea nivelului de recurenta. Tot pentru eficientizare se foloseste o multime de perechi de stari care se presupunea ca sunt echivalente.

```
def equiv (p , q , k ) :
        i f k = 0 :
                return (p in F and q in F) or (not p in F and not q in F)
        e l i f (p , q ) in S :
                return True
        el se :
                eq = (p in F and q in F) or (not p in F and not q in F)
                S = S ∪ {(p , q )}
                for a in Σ :
                        i f not eq :
                                return Fal s e
                        eq = eq and equiv (δ(p, a) , δ(q, a) , k−1)
                S = S − {(p , q )}
        return eq
```

Avand aceasta functie, putem testa echivalenta tuturor perechilor de stari din automatul initial pentru a forma clasele de echivalenta. Tranzitiile din automatul minim se obtin din tranzitiile din automatul initial, unde starile intre care se face tranzitia in automatul minim sunt clasele de echivalenta ale starilor intre care se face tranzitia in automatul initial. Similar, noile stari de start si finish ale automatului minim sunt clasele de echivalenta ale statilor initiale de start si finis.

Algoritmul in forma bruta are o complexitate exponentiala cu numarul de stari si litere din alfabet. Totusi, folosind tehnici de memoizare a functiei *equiv* si structuri de date adecvate constructiei claselor de echivalenta intre stari, poate fi imbunatatit pana la un nivel de complexitate patratica. Desi nici astfel nu se apropie de complexitatea de numai *n* log(n)* a algoritmului lui Hopcroft, are avantajul metodei incrementale.

IV. DESCRIEREA PROGRAMULUI

Pornind aplicatia veti observa ca **fereastra principala** contine o suprafata de lucru, pe care s-a desenat un automat simplu, si deasupra acesteia, un „tab" corespunzator numelui grafului curent. In acest mod, se pot deschide simultan mai multe grafuri/automate, care pot fi si salvate pentru o vizualizare/modificare ulterioara. Plimbarea mousului peste automatul desenat va pune in evidenta faptul ca muchiile si varfurile reactioneaza schimbandu-si culoarea atunci cand cursorul se afla deasupra lor. In partea de jos a ferestrei se afla o bara cu informatii ce afiseaza: numele grafului/automatului selectat, numarul de varfuri si noduri, tipul acestuia (orientat, neorientat, determinist, nondeterminist), alfabetul (in cazul automatelor) si in unele cazuri, limbajul acceptat.

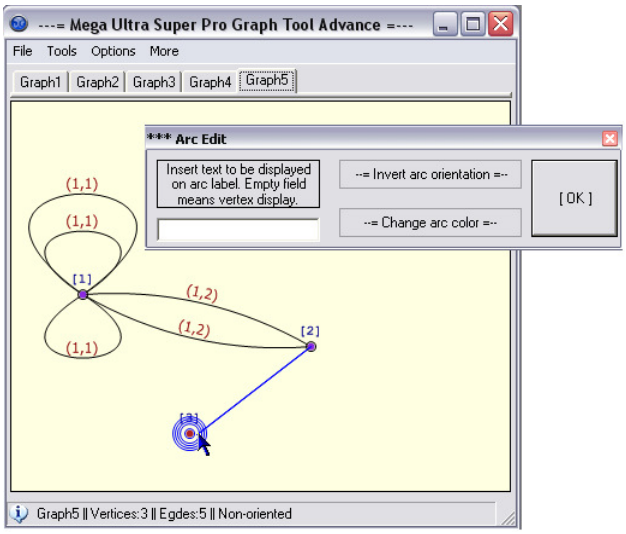


Figura 7

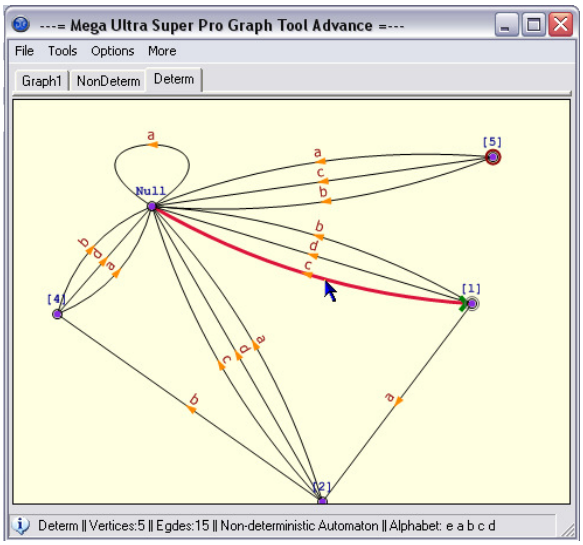


Figura 6

Un **graf orientat** difera de un **automat nondeterminist** prin faptul ca are un varf de start (doar unul singur este posibil), cel putin unul de finish si toate muchiile avand campul informatie completat cu *litere mici*.

Dupa cum se poate vedea apasand pe meniul „Quick Help" (sau F1), grafurile/automatele se pot **construi** in felul urmator:

- Click dreapta in spatiu gol ➔ plaseaza varf nou
- Click dreapta varf ➔ eliminare varf selectat (impreuna cu toate muchiile adiacente)
- Click stanga & drag varf ➔ muta varf selectat
- Dublu-click stanga varf ➔ modifica proprietati varf
- [CTRL] + Click stanga varf ➔ seteaza varful sursa al unei noi muchii (aceeasi operatie pentru alegerea varfului destinatie)
- Click dreapta muchie ➔ eliminare muchie selectata
- Dublu-click stanga muchie ➔ modifica proprietati muchie.

Proprietatile unui varf sunt: [a] informatia continuta, care se afiseaza imediat deasupra acestuia (daca acest camp este gol, se va afisa indexul varfului); [b] tipul nodului (de start si/sau de finish, in cazul automatelor).

Proprietatile unei muchii sunt: [a] informatia continuta, care se afiseaza imediat deasupra muchiei (daca acest camp este gol, se vor afisa indecsii varfurilor adiacenti); [b] sensul (la grafuri orientate sau automate) care se poate inversa; [c] culoarea, cea implicita fiind negru.

In mod implicit, grafurile care se construiesc sunt orientate (ca si automatele). Pentru **conversia** intre cele doua tipuri de grafuri se foloseste meniul „Tools>Convert". Acesta permite:

- Conversie graf orientat ➔ p-graf neorientat
- Conversie graf orientat ➔ 1-graf neorientat (intre doua varfuri se pastreaza o singura muchie)
- Conversie automat ➔ graf orientat (varfurile de start/finish devin oarecare)
- Conversie graf neorientat ➔ graf orientat, prin dedublarea muchiilor
- Conversie graf neorientat ➔ graf orientat, in mod aleator (se alege la intamplare un sens pe muchie).

Dupa cum am spus, grafurile/automatele pot fi **salvate**, in fisiere de format text (cu extensia „.graf" sau „.txt") si deschise pentru vizualizare. Aceste operatii pot fi facute apeland la meniul „File".

Pentru **generarea** obiectelor, din meniul „Tools>Create" putem obtine:

- Automate nondeterministe pe baza unui limbaj acceptat, dat sub forma unei expresii regulate, ce contine litere mici si operatorii U,(,),*
- Automate deterministe pornind de la un automat nondeterminist
- Automate minime pornind de la un automat determinist
- Grafuri (complete) neorientate de un numar dat de varfuri
- Completarea campului informatie al muchiilor cu litere mici in mod aleator.

Pentru **analiza** grafurilor/automatelor, meniul „Tools>Compute" permite:

- Calcularea lungimii si evidentierea drumului minim intre doua varfuri date
- Calcularea numarului de drumuri de lungime (mai mica sau) egala cu un numar dat intre doua varfuri alese
- Numarul de componente conexe si tari conexe pentru grafuri neorientate si respectiv orientate precum si identificarea componentelor de care apartin varfurile
- Aflarea starii finale a unui automat determinist in urma parcurgerii unui cuvant dat (si implicit daca acel cuvant este acceptat)
- Aflarea numarului ciclomatic al unui graf neorientat (= nr.muchii – nr.varfuri + nr.componente conexe).

Din meniul „Options" putem face urmatoarele **setari**:

- Modificarea distantei dintre muchii, cu ajutorul a doi parametri
- Dispunerea imediata a varfurilor grafului curent in cerc sau la intervale regulate
- Dezactivarea afisarii informatiei atasate muchiilor
- Imbunatatirea calitatii desenului grafic
- Afisarea ferestrei de editat proprietati muchii/varfuri imediat dupa plasarea lor.

V. ASPECTE DE PROGRAMARE

Programul este scris in C# prin MS Visual Studio .NET. Codul este in intregime scris de mine si mai contine pe langa algoritmii prezentati algoritmi standard de lucru cu grafuri si matrice.

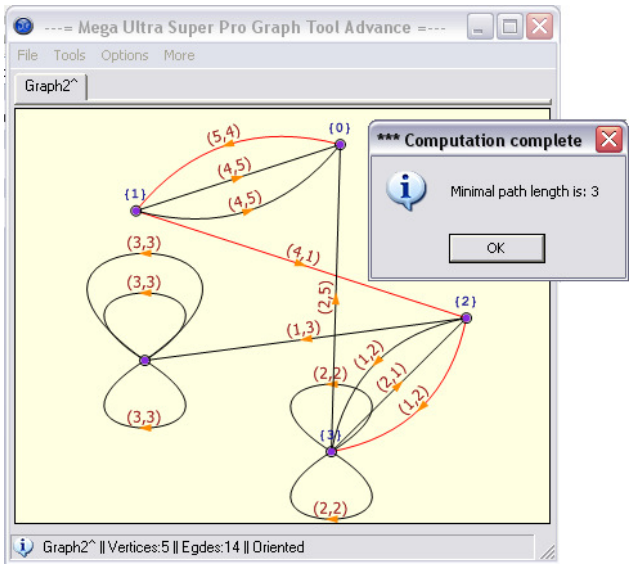


Figura 9

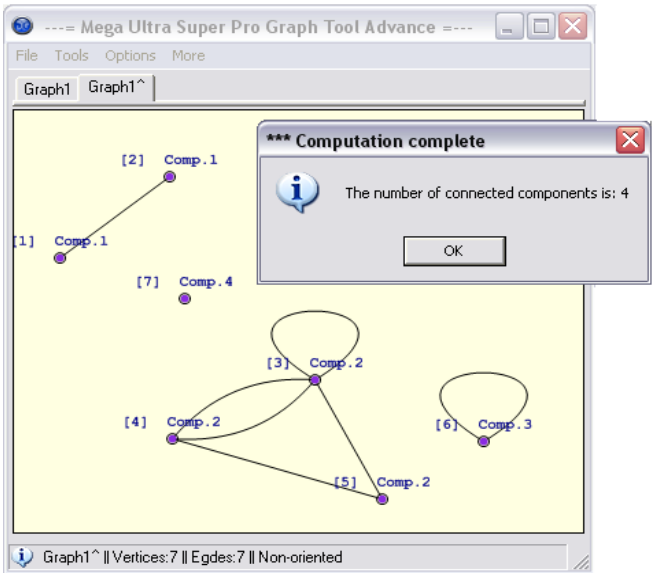


Figura 8

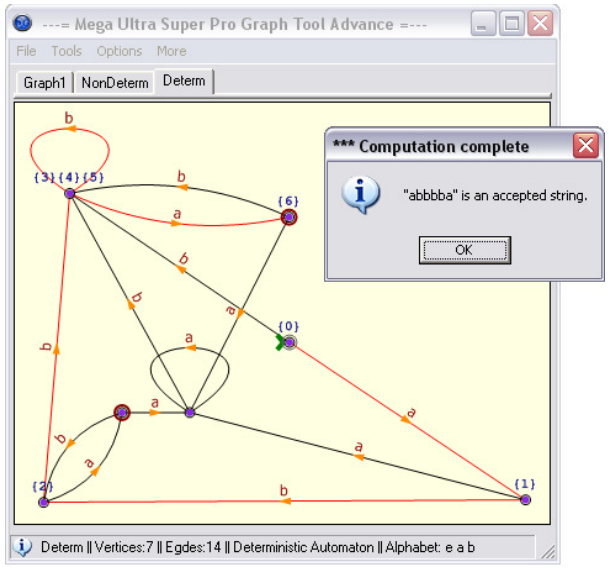


Figura 10

Grafurile (si implicit automatele) sunt retinute printr-o lista de „Varfuri” si una de „Muchii”, pe baza carora se poate construi cand este cazul matricea de adiacenta. Aceasta matrice este folosita la **explorarea in adancime/latime** a grafurilor pentru aflarea drumului minim intre doua varfuri, componentelor (tari) conexe etc. De asemenea, puterile acestei matrice folosesc la calcularea numarului de drumuri intre doua puncte de lungime (mai mica sau) egala cu o valoare data. Cele 3 imagini alaturate reprezinta cate un exemplu de drum minim, separare in componente conexe si stare finala de automat determinist dupa un anumit cuvant.

O alta problema importanta a fost **reprezentarea grafica** a grafurilor/automatelor, mai precis a muchiilor dintre aceleasi doua varfuri; acestea nu mai pot fi simple drepte, ci curbe. Astfel, muchiile sunt de fapt curbe Bezier cu doua puncte de control, ale caror coordonate pot fi calculate astfel incat curbele sa fie la distanta estetic regulata. Curba este parametrizata prin polinoame de gradul 3. Cele doua puncte de control sunt pe cate o dreapta perpendiculara la o treime din segmentul ce uneste varfurile, iar pozitia pe dreapa depinde de doi parametri

ajustabili. Apasand tasta DEL punctele de control vor aparea sub forma unor cercuri mov. Am ales acest tip de curba Bezier deoarece este un tip des intalnit in programele de grafica pe calculator si singurul tip de curba pe care limbajul stie implicit sa o deseneze. Dar desenarea facila nu m-a scutit de problema aflarii distantei de la un punct in plan (cursorul mouse-ului) la o curba Bezier (necesara pentru evidentierea muchiei selectate). Solutia folosita nu a fost in acest caz una matematica, ci numerica prin compararea distantei dintre cursor la anumite puncte de pe curba.

O eventuala imbunatatire a programului ar include un algoritm de planarizare a grafurilor; pe cat posibil, tinand cont de existenta a mai multor muchii intre aceleasi doua varfuri.

VI. Bibliografie

- Olteanu M. – Discrete Mathematics – Printech, 2009
- Almeida M., Moreida N., Reis R. – On the Performance of Automata Minimization Algorithms - Technical Report Series: DCC-2007-03, 2007
- Watson B. W., Daciuk J. – An efficient DFA minimization algorithm – Natural Language Engineering, pag 49–64, 2003