Tema 2 AA 2016-2017
Automate finite nedeterministe

George Daniel MITRA

10 decembrie 2016

Rezumat

Tema constă în propunerea unor algoritmi determiniști sau nedeterminiști pentru a efectua operații pe automate finite nedeterministe.

1 Introducere

În procesarea de text, automatele finite sunt folosite pentru interpretarea expresiilor regulate, pentru analiza lexicală, pentru evaluarea protocoalelor de comunicație etc. În subsecțiunile următoare vom defini formal automatele finite.

1.1 Noțiuni ajutătoare

Un șir sau cuvânt reprezintă o secvență finită de caractere sau simboluri.

Un alfabet reprezintă o multime finită de simboluri. De obicei alfabetele se notează cu Σ.

Notăm cu Σ^* mulțimea tuturor șirurilor formate din simboluri din Σ.

Se notează cu 2^M mulțimea tuturor submulțimilor mulțimii M, numită si mulțimea putere. Mulțimea vidă și mulțimea originală fac parte din mulțimea putere. De exemplu, $2^{\{1,2\}} = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$.

1.2 Automate finite

Un automat finit reprezintă un model de calculabilitate care primește o secvență de simboluri la intrare și își modifică în funcție de ea starea internă. La finalul trecerii întregului șir prin automat, automatul acceptă sau nu șirul.

Se numește tranziție schimbarea stării curente cu o stare nouă la întâlnirea unui anumit simbol. Starea veche și starea nouă pot să coincidă.

Se numește stare finală o stare în care automatul acceptă șirul primit până atunci dacă nu mai este niciun simbol la intrare.

Un automat este complet definit dacă se cunosc:

- stările
- simbolurile de intrare
- funcția de tranziție
- stările finale

1.2.1 Automat finit determinist

Un automat finit este determinist dacă pentru orice șir, calea de la starea inițială până la răspuns este unică.

Din punct de vedere formal, un automat finit determinist este un tuplu $(K, \Sigma, \delta, s, F)$, unde:

- K este mulțimea stărilor
- Σ este mulțimea simbolurilor de intrare
- $\delta : K \times \Sigma \to K$ este o functie care primește o stare și un simbol de intrare și întoarce valoarea noii stări
- s este starea inițială. Starea inițială este starea în care începe automatul, indiferent de șir
- F reprezintă mulțimea stărilor finale. Dacă la finalul trecerii unui șir prin automat acesta este într-o stare finală, atunci șirul este acceptat de automat

1.2.2 Automat finit nedeterminist

Vom folosi în cele ce urmează o versiune restricționată de automat finit nedeterminist. Din punct de vedere formal, un astfel de automat este un tuplu $(K, \Sigma, \delta, s, F)$.

Singura diferență față de automatul anterior este dată de faptul că δ nu mai întoarce o stare, ci o multime de stări. $\delta : K \times \Sigma \to 2^K$.

2 Probleme

Să se scrie pseudocodul pentru algoritmii care rezolvă problemele enunțate în continuare, modificând fișirele **Problem1.pc**, **Problem2.pc** și **Problem3.pc**.

2.1 Determinarea în mod determinist a existenței unui șir acceptat (3p)

Să se determine dacă există vreun șir acceptat de automatul dat, M. Un automat acceptă cel puțin un șir dacă există cel puțin o cale prin automat de la starea initială la o stare finală.

Scrieți algoritmul determinist în **Problem1.pc** folosind următoarele resurse:

```
boolean solve(M)
```

- funcția care trebuie modificată. M este un automat finit nedeterminist. Functia returnează *true* dacă M acceptă cel putin un cuvânt, *false* altfel.

```
getInitialState(M)
```

- returnează starea inițială a automatului finit nedeterminist M.

```
hasChildrenStates(M, state)
```

- returnează *true* dacă starea *state* din automatul M are cel puțin o tranziție, altfel *false*

`getChildrenStates(M, state)`

- returnează mulțimea stărilor accesibile din starea *state* a automatului M, indiferent pe ce simbol ar fi definite tranzițiile. Lista poate fi vidă dacă *hasChildrenStates(M, state)* returnează *false*. Puteți trata rezultatul funcției ca pe un vector.

`isFinal(M, state)`

- returnează *true* dacă starea *state* a automatului M e finală, altfel *false*.

Scrieți în fișierul README clasa de complexitate folosind notația O.

2.2 Determinarea în mod determinist a acceptării unui șir dat (4p)

Să se determine dacă automatul M acceptă șirul w. Automatul acceptă șirul w dacă, pornind din starea inițială, prin urmarea tranzițiilor pe simbolurile din w se ajunge la o stare finală pe cel puțin o cale.

Scieți algoritmul determinist în **Problem2.pc** folosind următoarele resurse:

`solve(M, w)`

- funcția care trebuie modificată. Returnează *true* dacă M acceptă șirul w, *false* altfel.

`getInitialState(M)`

- returnează starea inițială a automatului finit nedeterminist M.

`hasChildrenStates(M, state, symbol)`

- returnează *true* dacă starea *state* a automatului M are cel puțin o tranziție pe simbolul *symbol*.

Atenție!!! Deși are același nume cu funcția de la exercițiul anterior, are un parametru în plus.

`getChildrenStates(M, state, symbol)`

- returnează mulțimea stărilor accesibile din starea *state* a automatului M pe simbolul *symbol*. Puteți trata rezultatul funcției ca pe un vector.

`isFinal(M, state)`

- returnează *true* dacă starea *state* a automatului M e finală, altfel *false*.

Scrieți în fișierul README clasa de complexitate folosind notația O.

2.3 Determinarea în mod nedeterminist a acceptării unui șir dat (3p)

Să se determine dacă automatul M acceptă șirul w. Automatul acceptă șirul w dacă, pornind din starea inițială, prin urmarea tranzițiilor pe simbolurile din w se ajunge la o stare finală pe cel puțin o cale.

Scrieți algoritmul nedeterminist în **Problem3.pc** folosind următoarele resurse:

`solve(M, w)`

- funcția care trebuie modificată. Raportarea rezultatului se face folosind instrucțiunea *success* în cazul în care automatul M acceptă șirul w și *fail* altfel.

`GetInitialState(M)`

- returnează starea inițială a automatului M.

`hasChildrenStates(M, state, symbol)`

- returnează *true* dacă starea *state* a automatului M are cel puțin o tranziție pe simbolul *symbol*.

`getChildrenStates(M, state, symbol)`

- returnează mulțimea stărilor accesibile din starea *state* a automatului M pe simbolul *symbol*.

`choice(set)`

- alege unul din elementele mulțimii *set* în mod nedeterminist.

`isFinal(M, state)`

- returnează *true* dacă starea *state* a automatului M e finală, altfel *false*.

`success`

- termină necondiționat execuția tuturor firelor de execuție cu un răspuns afirmativ.

`fail`

- termină necondiționat execuția firului curent de execuție cu un răspuns negativ.

Scrieți în fișierul README clasa de complexitate folosind notația O.

3 Trimiterea temei

3.1 Conținutul arhivei

Arhiva trebuie să conțină:

- fișierele Problem*.pc

- un fișier README în care scrieți clasa de complexitate pentru fiecare din problemele de mai sus și în care descrieți **sumar** abordarea la fiecare problemă.

Nerespectarea oricărui aspect mai sus menționat va duce la nepunctarea temei.

Arhiva trebuie să fie zip. Nu rar, 7z, ace sau alt format ezoteric. Fișierul README și Problem*.pc trebuie să fie în rădăcina arhivei, nu în vreun director.

Numele arhivei trebuie să fie:

<nume>_<prenume1>[_<prenume2>]_<Grupa>_Tema2AA.zip

Pentru cum trebuie să arate numele si grupa, citiți subsecțiunea următoare!

3.2 Format README

Prima linie din README trebuie să conțină numele vostru, așa cum apare pe cs.curs.pub.ro. Prenumele trebuie să fie complet, numele să fie scris cu majuscule. Dacă aveți diacritice în nume, ele trebuie să apară și salvați fișierul ca UTF-8. Prima linie ar trebui să arate așa, cu mențiunea că folosiți numele vostru:

Nume: MITRA George Daniel

A doua linie trebuie să conțină seria și grupa. Seria și grupa trebuie să fie lipite, fără spații între ele, seria apărând înainte. Exemplu:

Grupa: CC324

Restanțierii de anul patru își vor scrie grupa curentă, nu grupa în care erau când au făcut AA, punând un R în față. Exemplu:

Grupa: R342C4

Urmează o linie liberă, urmată de o linie care conține doar:

I.

Urmează trei linii care respectă următorul format:

<i>) O(<f>)

unde <i> $\in \{1, 2, 3\}$ și <f> reprezintă funcția de complexitate pentru exercițiul <i>.

Urmează o altă linie liberă, urmată de o linie care conține doar:

II.

Urmează trei blocuri de text de maxim 20 de linii a câte maxim 100 de caractere în care descrieți sumar abordarea și orice altă precizare pentru fiecare din problemele de mai sus. Prima linie a fiecărul bloc începe cu:

<i>)

unde <i> $\in \{1, 2, 3\}$.

Fișierul trebuie să se numească README, nu readme, ReadMe, README.txt, readme.txt, read-me.doc, rEADME, README.md sau alte variante asemănătore sau nu. Nerespectarea acestor cerințe duce la nepunctarea temei.

3.3 Deadline

Tema are timp de lucru două săptămâni.

Termenul de predare este pe 23 decembrie 2016, ora 23:55. Posibilitatea de trimitere a arhivei se va păstra până la ora 05:00 în ziua următoare.