Tema 1 LFA
Limbaje regulate

George Daniel MITRA

26 decembrie 2013

Cuprins

1 Specificații temă **3**
1.1 Cerință . . . 3
1.2 Limbaje de programare . . . 3
1.3 Sistem de operare . . . 3
1.4 Conținutul arhivei . . . 3
1.5 Specificații program . . . 3
1.5.1 Parametri în linie de comandă . . . 3
1.5.2 Intrări . . . 4
1.5.3 Ieșiri . . . 4
1.6 FLEX . . . 4

2 Noțiuni introductive **4**
2.1 Limbajul de descriere . . . 4
2.2 Simbol, Alfabet, Șir, Limbaj . . . 5
2.2.1 Simbol . . . 5
2.2.2 Alfabet . . . 5
2.2.3 Șir . . . 5
2.2.4 Limbaj . . . 6

3 Expresii Regulate(ER) **6**
3.1 Descriere . . . 6
3.2 Specificații . . . 7

4 Automate Finite Deterministe(AFD) **7**
4.1 Descriere . . . 7
4.2 Specificații . . . 8

5 Automate Finite Nedeterministe(AFN) **8**
5.1 Descriere . . . 8
5.2 Specificații . . . 9

6 [Bonus]Gramatici Regulate(GR) **9**
6.1 Descriere . . . 9
6.2 Gramatici regulate dreapta . . . 9
6.3 Gramatici regulate stânga . . . 9
6.4 Specificații . . . 10

7 [Ghid de utilizare]Checker **10**

8 Punctaj **10**
8.1 Checker 10
8.2 Depunctări 10

9 Sugestii **11**

10 Concluzii **11**

Bibliografie **12**

Rezumat

Tema constă în implementarea unui program care face conversii între toate formele de reprezentare finită a limbajelor regulate.

1 Specificații temă

1.1 Cerință

Să se implementeze un program care, primind o formă de reprezentare finită a unui limbaj regulat, să îl convertească într-o altă formă sau să stabilească dacă anumite cuvinte fac parte din limbaj.

Programul trebuie să folosească FLEX pentru parsarea intrării. Vezi secțiunea 1.6 pentru mai multe detalii. Soluțiile care nu folosesc FLEX vor fi depunctate după cum e menționat în secțiunea 8.

Nu trebuie implementate toate metodele de conversie. Detalii puteți găsi în secțiunea 9.

1.2 Limbaje de programare

Pentru această temă se pot folosi C/C++, Java sau Haskell. Dacă aveți o altă propunere pentru care există un generator de parsere de tipul flex, postați pe forum și dacă e suficient de similar vă vom aproba utilizarea.

1.3 Sistem de operare

Pentru temă se pot folosi atât sisteme *NIX, cât și Windows.

1.4 Conținutul arhivei

Arhiva trebuie să conțină:

- surse, a căror organizare nu vă e impusă de noi
- un fișier Makefile care să aibă target de build și run.
- un fișier README care să respecte exact formatul din exemplul de pe site, în care să descrieți abordarea pentru lexer/parser, ce algoritmi ați implementat și de ce i-ați ales și care e limbajul ales pentru ultimele patru puncte.

1.5 Specificații program

1.5.1 Parametri în linie de comandă

Programului i se vor da parametri în linie de comandă. Parametrii pot descrie intrarea sau ieșirea. Din fiecare grup va apărea exact un exemplar.

Parametrii care descriu intrarea sunt:

- --from-RE: Specifică faptul că se va citi o expresie regulată, în formatul specificat mai jos
- --from-DFA: Specifică faptul că se va citi un automat finit determinist

- --from-NFA: Specifică faptul că se va citi un automat finit nedeterminist
- --from-RG: Specifică faptul că se va citi o gramatică regulată

Parametrii care descriu ieșirea sunt:

- --to-RE: Specifică faptul că se va afișa o expresie regulată
- --to-DFA: Specifică faptul că se va afișa un automat finit determinist
- --to-NFA: Specifică faptul că se va afișa un automat finit nedeterminist
- --to-RG: Specifică faptul că se va afișa o gramatică regulată
- --contains: Permite interogări ale apartenenței cuvintelor la limbajul descris. Parametrii de după „--contains" sunt cuvintele căutate. Pot fi zero sau mai multe cuvinte.

1.5.2 Intrări

Programul va citi de la intrarea standard expresia, automatul sau gramatica, în formatul specificat în secțiunea alocată fiecăruia.

Se consideră corect orice primește programul ca intrare.

1.5.3 Ieșiri

Programul va afișa la ieșirea standard rezultatul, în formatul corespunzător. În cazul unei interogări de tipul „--contains", pentru fiecare cuvânt se va afișa pe câte o linie „True", respectiv „False", dacă limbajul conține cuvântul sau nu.

1.6 FLEX

FLEX este o unealtă pentru generarea de analizoare lexicale. Variantele acceptate sunt:

- flex(C/C++) [1, 2, 3]
- jflex(Java) [4, 5, 6]
- alex(haskell) [7, 8, 9]

Dacă găsiți un alt flex, vă putem aproba folosirea limbajului pentru care a fost făcut dacă postați pe forum și e suficient de apropiat de flex.

2 Noțiuni introductive

2.1 Limbajul de descriere

Limbajul este descris printr-o gramatică BNF și folosește aceeași convenție de culori ca articolul Wikipedia despre BNF:

- albastru - neterminali
- verde - operatori ai limbajului BNF și paranteze ajutătoare
- rosu - terminali (elemente care fac parte efectiv din limbajul descris)

2.2 Simbol, Alfabet, Șir, Limbaj

2.2.1 Simbol

Un simbol este un caracter, semnal sau orice obiect ce poate fi interpretat cumva. Exemple de simboluri sunt în secțiunea 2.2.2

În temă, un simbol poate fi literă, cifră sau caracter special:

<symbol> ::= <lower-case letter> | <digit> | <other>

<lower-case letter> ::= (a | b | c | d | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z)

<digit> ::= (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)

<other> ::= (! | # | $ | % | & | - | . | / | : | ; | < | > | = | @ | [|] | ^ | ‘ | ~)

2.2.2 Alfabet

Un alfabet este orice mulțime nevidă finită de simboluri.
Exemple:

- alfabetul roman: $\Sigma = \{A, B, C, ..., Z\}$
- alfabetul format din cifre: $\Sigma = \{0, 1, 2, ..., 9\}$
- alfabetul semnalelor pentru 3-Way Handshake: Σ = {SYN, ACK, SYN-ACK}

În limbajul temei, un alfabet conține cel puțin un simbol, deci se declară astfel:

<alphabet> ::= { <symbol> (, <symbol>)* }

2.2.3 Șir

Un șir(cuvânt) este o secvență finită de simboluri dintr-un alfabet. Un șir poate să nu conțină niciun caracter. Acesta, șirul vid, se notează cu e(în alte surse se notează cu ε).
Lungimea unui șir w se notează cu $|w|$ și reprezintă numărul de simboluri din care e format w.
Exemple:

- $\Sigma = \{a, b, c, d\}, w = cbba, |w| = 4$
- $\Sigma = \{0, 1\}, w = 10010101, |w| = 8$
- $\Sigma = \{a, b\}, w = e, |w| = 0$

În limbajul temei, șirul vid se notează cu e, în timp ce alte șiruri reprezintă o concatenare de unul sau mai multe simboluri. Din acest motiv, e nu se consideră simbol, deci nu va face niciodată parte din niciun alfabet.

<word> ::= e | (<symbol>)+

2.2.4 Limbaj

Un limbaj peste un alfabet Σ este o mulțime de cuvinte formate din simboluri din Σ. Limbajul vid se notează cu $\emptyset$, și este limbajul în care nu se află niciun cuvânt. Acesta este diferit de limbajul care conține doar șirul vid. Un limbaj poate fi finit sau infinit.
Exemple:

- $L = \emptyset, |L| = 0$
- $L = \{e\}, |L| = 1$
- $L = \{e, a, ab\}, |L| = 3$
- $L = \{a, aa, aaa, ...\}, |L| = \infty$

În temă, limbajele apar reprezentate doar sub formă de expresii regulate, automate finite și gramatici regulate.

3 Expresii Regulate(ER)

3.1 Descriere

Expresiile regulate reprezintă o metodă de reprezentare finită a limbajelor. Ele pot descrie orice succesiune finită de operații de reuniune, concatenare și Kleene Star.

Fie Σ un alfabet. O expresie regulată este o secvență finită de simboluri din $\Sigma \cup \{\cup, ^*, (,), \varnothing, e\}$, ținând cont de următoarele proprietăți:

1. $\varnothing$ și e sunt expresii regulate, reprezentând limbajul vid, $L_1 = \emptyset$, respectiv limbajul care conține doar șirul vid, $L_2 = \{e\}$;
2. $\forall a \in \Sigma$, a este expresie regulată, reprezentând limbajul ce conține un singur cuvânt format din simbolul a, $L = \{a\}$;
3. $\forall \alpha, \beta$ expresii regulate, $\alpha \cup \beta$ este expresie regulată, reprezentând reuniunea limbajelor descrise de expresiile regulate α și β, $\mathcal{L}(\alpha \cup \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$;
4. $\forall \alpha, \beta$ expresii regulate, $\alpha\beta$ este expresie regulată, reprezentând concatenarea limbajelor descrise de expresiile regulate α și β, $\mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha) \circ \mathcal{L}(\beta)$;
5. $\forall \alpha$ expresie regulată, α^* este expresie regulată, reprezentând aplicarea operatorului Kleene Star limbajului descris de expresia regulată α, $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$;
6. $\forall \alpha$ expresie regulată, (α) este expresie regulată. Parantezele cresc prioritatea operatorilor. Operatorii, ordonați de la prioritate maximă la minimă sunt: Kleene Star, operator de concatenare, operator de reuniune;
7. Orice altceva nu este expresie regulată.

Notații ajutătoare:

- Fie α o expresie regulată. Notăm $\alpha^+ = \alpha\alpha^*$.

- Fie α o expresie regulată. Notăm $\alpha? = (\alpha \cup e)$

Exemple:

- $E = e \cup ab \cup bb^*$. $\mathcal{L}(E) = \mathcal{L}(e) \cup \mathcal{L}(ab) \cup \mathcal{L}(bb*) = \{e\} \cup \{ab\} \cup \mathcal{L}(b) \circ \mathcal{L}(b^*) = \{e, ab\} \cup \{b\} \circ \mathcal{L}(b)^* = \{e, ab\} \cup \{b\} \circ \{b\}^* = \{e, ab\} \cup \{b\} \circ \{e, b, bb, bbb, ...\} = \{e, ab\} \cup \{b, bb, bbb, bbbb, ...\} = \{e, ab, b, bb, bbb, ...\}$;
- $E = (a \cup b)^*$. $\mathcal{L}(E) = \mathcal{L}((a \cup b)^*) = \{a, b\}^* = \{e, a, b, aa, ab, ba, bb, aaa, ...\}$;
- $E = a^* \cup b^*$. $\mathcal{L}(E) = \mathcal{L}(a^*) \cup \mathcal{L}(b^*) = \{e, a, aa, aaa, ...\} \cup \{e, b, bb, bbb, ...\} = \{e, a, aa, aaa, ..., b, bb, bbb, ...\}$;

3.2 Specificații

În temă, din cauza faptului că le folosim pentru a desemna elemente constitutive expresiilor regulate, caracterele $\{|, *, +, ?, O, e, (,)\}$ nu pot face parte din niciun alfabet. O expresie regulată se definește în felul următor:

<RE> ::= <alphabet> : <expression>

<expression> ::= O | e | <symbol> | (<expression> | <expression>) | (<expression> <expression>) | (<expression> *) | (<expression> +) | (<expression> ?) | (<expression>)

În temă, | reprezintă reuniunea, O reprezintă expresia limbajului vid, e reprezintă expresia limbajului care conține doar șirul vid.

4 Automate Finite Deterministe(AFD)

4.1 Descriere

Un automat finit este un model de calcul care primește la intrare o bandă de simboluri și își modifică starea internă în funcție de ce are la intrare. Acesta poate fi privit ca o cutie neagră cu niște stări între care face tranziții în funcție de intrare, la fiecare tranziție mișcând capul de citire la dreapta.

Formal, un automat finit determinist este un tuplu $M = (K, \Sigma, \delta, s, F)$, cu următoarele proprietăți:

- K este mulțimea stărilor. K este finită, de unde vine și numele automatului;
- Σ este alfabetul din care sunt formate cuvintele acceptate de automat;
- δ se numește funcție de tranziție. $\delta : K \times \Sigma \to K$. $\delta(p, a) = q; p, q \in K; a \in \Sigma$ înseamnă că automatul, dacă se află în starea p și primește pe bandă a, trece în starea q. Fiindcă δ este funcție, toate tranzițiile din fiecare stare pentru fiecare simbol trebuie să fie definite.
- $s \in K$ este starea de start. Aceasta este starea în care se află automatul înainte de a primi ceva pe bandă.
- $F \subseteq K$ este mulțimea stărilor finale. Dacă automatul se află într-o stare $f \in F$ după ce a terminat de citit șirul înseamnă că acceptă șirul.

4.2 Specificații

În temă, un automat finit determinist este dat ca un tuplu, conform definiției:

<DFA> ::= (<states> , <alphabet> , <transitions> , <state> , (<states> | O))

<states> ::= { <state> (, <state>)* }

<state> ::= <name>

<name> ::= (<upper-case letter> | _) (<lower-case letter> | <digit> | _) *

<upper-case letter> ::= (A | B | C | D | E | F | G | H | I | J | K | L | M | N | P | Q | R | S | T | U | V | W | X | Y | Z)

<transitions> ::= { <transition> (, <transition>) * }

<transition> ::= d(<state> , <symbol>)= <state>

5 Automate Finite Nedeterministe(AFN)

5.1 Descriere

Un automat finit nedeterminist este un automat finit mai puțin restrictiv decât unul determinist. Acesta se bazează pe faptul că se pot rula în paralel o infinitate de ramuri de execuție și dacă cel puțin una se termină într-o stare finală, șirul este acceptat. De fapt, doar un număr finit de ramuri de execuție se rulează, fiindcă nu e nevoie de mai mult. Acel număr e cel mult $|K|$.

Un automat finit nedeterminist permite:

- absența tranzițiilor pe anumite simboluri. Dacă se întâmplă ca la un moment dat să apară un simbol pentru care nu există tranziție, ramura curentă de execuție se termină fără a accepta șirul;
- existența mai multor tranziții din aceeași stare pe același caracter. Dacă se întâmplă acest lucru, automatul lansează mai multe ramuri de execuție, câte una pentru fiecare tranziție posibilă;
- tranziții pe șiruri. Fiindcă se pot neglija tranziții, un lanț de tranziții în care din stările intermediare nu pleacă nicio tranziție se poate considera o singură tranziție pe un șir.
- tranziții pe șirul vid. Automatul își poate modifica spontan starea fără a citi vreun caracter.

Un automat nedeterminist este tot un tuplu $M = (K, \Sigma, \Delta, s, F)$ cu următoarele proprietăți:

- K este mulțimea finită a stărilor;
- Σ este alfabetul din care sunt formate cuvintele acceptate de automat;
- $\Delta \subset (K \times \Sigma^* \times K)$ nu este funcție, este relație. Δ este o finită. $(p, u, q) \in \Delta$ înseamnă că automatul poate trece din starea p în starea q citind șirul u;
- $s \in K$ este starea de start;
- $F \subseteq K$ este mulțimea stărilor finale.

5.2 Specificații

În temă, un automat finit nedeterminist este tot un tuplu, definit în felul următor:

<NFA> ::= (<states> , <alphabet> , <relations> , <state> , (<states> | O))

<relations> ::= O | { <relation> (, <relation>) * }

<relation> ::= (<state> , <word> , <state>)

Din cauza confuziei ce poate apărea, următoarul caracter nu va apărea în vreun alfabet: ','. Fiindcă un automat finit nedeterminist poate să nu aibă tranziții, mulțimea de tranziții poate fi vidă.

6 [Bonus]Gramatici Regulate(GR)

6.1 Descriere

O gramatică reprezintă o mulțime de reguli de combinare a simbolurilor dintr-un alfabet dat în cuvinte din limbajul descris.

O gramatică este, din punct de vedere formal, un tuplu $G = (V, \Sigma, R, S)$, unde V reprezintă mulțimea simbolurilor ce pot apărea în reguli, Σ reprezintă alfabetul, R reprezintă mulțimea regulilor și S reprezintă simbolul de start.

Σ este alfabetul, numit și mulțime de terminali. V este mulțimea tuturor simbolurilor care apar în reguli, dar, pe lângă terminali, conține și niște simboluri care nu apar în cuvintele limbajului, numite neterminali. Un astfel de simbol este $S \in V \setminus \Sigma$.

R este mulțimea finită a regulilor de producție. O regulă este un tuplu, $(A, \gamma), A \in V \setminus \Sigma, \gamma \in V^*$, care înseamnă că neterminalul A poate fi înlocuit cu șirul de terminali și neterminali γ. Regula (A, γ) se notează cu $A \to \gamma$. Modul în care este definit face ca $R \subset (V \setminus \Sigma) \times V^*$. În cazul în care $\exists A \in V \setminus \Sigma, \exists \gamma, \beta \in V^*, (A, \gamma) \in R \wedge (A, \beta) \in R$, atunci regulile se pot scrie $A \to \gamma|\beta$, însemnând că A se poate înlocui fie cu γ, fie cu β.

Pentru ca o gramatică să fie regulată, ea trebuie să fie fie regulată dreapta, fie regulată stânga.

6.2 Gramatici regulate dreapta

O gramatică este regulată dreapta dacă toate regulile au una din formele următoare:

- $A \to wB; A, B \in V, w \in \Sigma^*$;
- $A \to w; A \in V, w \in \Sigma^*$.

6.3 Gramatici regulate stânga

O gramatică este regulată stânga dacă toate regulile au una din formele următoare:

- $A \to Bw; A, B \in V, w \in \Sigma^*$;
- $A \to w; A \in V, w \in \Sigma^*$.

6.4 Specificații

În temă vom folosi doar gramatici regulate dreapta. În limbajul temei ele arată în felul următor:

<RG> ::= (<(non)terminals> , <alphabet> , <productions> , <nonterminal>)

<(non)terminals> ::= { <nonterminal> (, (<nonterminal> | <terminal>)) * }

<nonterminal> ::= <name>

<terminal> ::= <symbol>

<productions> ::= { <production> (, <production>) * }

<production> ::= (<nonterminal> -> <word>) | (<nonterminal> -> <word> <nonterminal>)

7 [Ghid de utilizare]Checker

Va fi completată când apare checker.

8 Punctaj

8.1 Checker

Checker-ul va oferi un punctaj între 0 și 10. Primele 9.6 puncte sunt împărțite în trei grupe, câte una pentru ER, AFD și AFN. Într-o grupă de 3.2 de puncte, sunt câte opt teste de conversie la fiecare metodă de reprezentare și de verificare a apartenenței cuvintelor. Pentru ultimele 0.4 puncte, alegeți un limbaj, descrieți în limbajul descris în enunț ER, AFN și AFD și scrieți-le în niște fișiere numite my-er, my-afd, respectiv my-afn. Checker-ul va face niște verificări și va oferi punctajul.

Pe lângă cele 10 puncte ale temei, checker-ul poate da și punctaj bonus. Acest punctaj este doar pentru a reduce depunctarea pentru întârzieri, nu pentru a completa lipsa unor teste anterioare. În categoria bonus intră tot ce implică gramatici regulate.

8.2 Depunctări

Pentru fiecare zi de întârziere se scad 0.25 puncte. Depunctarea pentru întârziere se va trunchia la 3.14 puncte. Bonusul valorează cel mult 3.2 puncte, deci poate anula complet depunctările pentru întârzieri.

Implementările care nu folosesc flex deloc vor fi punctate cu cel mult 20% din punctajul dat de checker. Parsarea de mână în C/C++/Java/Haskell se va depuncta la latitudinea asistentului care corectează. Pentru a lua punctaj maxim, se recomandă să folosiți cât mai multe din facilitățile flex/jflex/alex.

Temele care nu respectă formatul de README nu vor fi punctate, fiindcă checker-ul nu se va descurca.

Deadline soft pentru temă: 01.12.2013, ora 23:59. Calculul întârzierilor se va face la ora 03:00. Deadline hard: 05.01.2014, ora 23.59.

9 Sugestii

Nu trebuie să implementați toate conversiile pentru a le putea face pe toate. O conversie se poate face și printr-un stadiu intermediar. Pentru a le putea face pe toate, graful conversiilor trebuie să fie tare conex.

Pentru parametri în linie de comandă în Haskell, puteți folosi ca referință [10].

Pentru makefile, puteți folosi [11] sau să întrebați pe forum.

10 Concluzii

ER, AFD, AFN, GR sunt echivalente ca forme finite de reprezentare ale limbajelor regulate. Niciuna din ele nu are putere de descriere mai mare sau mai mică decât celelalte.

Bibliografie

[1] flex homepage

[2] Lexical Analysis with Flex

[3] Using flex

[4] jflex homepage

[5] jflex user manual

[6] jflex user manual in japanese

[7] alex homepage

[8] alex user guide

[9] Alex Haskell professional

[10] Haskell command line parameters

[11] Laborator 1 SO: Makefile