Hopkroftov algoritam za minimizaciju konačnog automata

Mina Krivokuća

7. novembar 2019.

Sadržaj

Sažetak

Minimizacija determinističkog konačnog automata je jedan od najznačajnijih i duboko izučavanih problema u teoriji automata i formalnih jezika. Za dati automat $\mathcal{A}$, minimizacija automata predstavlja nalaženje automata za isti jezik L takvog da ne postoji automat sa manje stanja koji prepoznaje jezik L. Hopkroftov algoritam predstavlja najbrže poznato rešenje za ovaj problem. U daljem tekstu ćemo detaljno opisati pomenuti algoritam. Koraci algoritma će biti razrađeni i približeni čitaocu kroz nekoliko primera.

1 Uvod i terminologija

Radi lakšeg razumevanja materije uvodimo osnovne pojmove i uspostavljamo notaciju.

1.1 Deterministički konačni automat

Deterministički konačni automat (DKA) uređena je petorka

$$(\Sigma, S, s_0, \delta, F)$$

za koju važi:

- Σ je azbuka nad čijim rečima primenjujemo automat
- S je konačan skup stanja
- s_0 je početno stanje, $s_0 \in S$
- $\delta : S \times \Sigma \to S$ je funkcija prelaska
- $F \subseteq S$ je skup završnih stanja (oznaka potiče od reči *Final*)

Definišemo takođe inverz funkcije prelaska $\delta^{-1} : S \times \Sigma \to \mathcal{P}(S)$ koji za dato stanje s i slovo l vraća skup S' koji sadrži sva stanja iz kojih postoji funkcija prelaska do s preko slova l: $\delta^{-1}(s, l) = \{s' \mid \delta(s', l) = s\}$.

1.2 Jezik stanja i jezik automata

Uvodimo sledeće dve oznake:

1. Oznaka $s \xrightarrow{a} s'$ ekvivalentna je $\delta(s, a) = s'$ za $\forall s, s' \in S$ i $\forall a \in \Sigma$
2. Neka je $\omega = a_1, a_2, ...a_n \in \Sigma^*$, tada je oznaka $s \xrightarrow{\omega} s'$ ekvivalentna

$$(\exists s_1, s_2, ...s_{n+1} \in S) : s = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} ... \xrightarrow{a_n} s_{n+1} = s'$$

Oznaka $s \xrightarrow{\omega} s'$ može se protumačiti kao: *„Preko reči ω se iz stanja s može stići u stanje s' "* ili *„Postoji put ω iz stanja s u stanje s' "*.

Jezik stanja q u oznaci L_q je skup:

$$L_q = \{\omega \mid q \xrightarrow{\omega} f, f \in F\}$$

Jezik automata $\mathcal{A}$ u oznaci $L(\mathcal{A})$ predstavlja skup:

$$L(\mathcal{A}) = \{\omega \mid s_0 \xrightarrow{\omega} f, f \in F\} = L_{s_0}$$

Da bi neka reč ω pripala jeziku automata $\mathcal{A}$, dovoljno je pronaći bilo koji put ω od početnog do završnog stanja automata $\mathcal{A}$.

1.3 Minimalni konačni automat

Postoji više automata koji prepoznaju isti jezik. Ne samo da se razlikuju po imenima stanja, već ih imaju i različit broj. Pri svakoj implementaciji leksičkog analizatora kao determinističkog konačnog automata, poželjno je koristiti DKA sa što manjim brojem stanja zbog efikasnije i lakše obrade. Za dva automata kažemo da su izomorfni ukoliko promenom imena stanja jednog možemo dobiti drugi.

DKA je minimalan ukoliko ne postoji automat sa manjim brojem stanja koji prepoznaje isti jezik. Minimalni DKA za bilo koji jezik L je jedinstven.

2 Ekvivalencije stanja konačnog automata

Definišemo $\delta^*(s_0, w)$ kao stanje do kog dolazimo prateći put w počevši od stanja s_0. Formalno:

- $\delta^*(s, a) = \delta(s, a), s \in S, a \in \Sigma$
- $\delta^*(s, \epsilon) = s, s \in S$
- $\delta^*(s, \omega a) = \delta(\delta^*(s, \omega), a)$, gde je ω neka reč $|\omega| \geq 1$ i $a \in \Sigma$

Naš cilj je da razumemo u kom slučaju dva stanja s i t možemo zameniti jednim koji se ponaša kao oba.

Kažemo da su stanja s i t ekvivalentna ako je za svaku ulaznu nisku ω $\delta^*(s, \omega)$ završno stanje samo ako je $\delta^*(t, \omega)$ takođe završno stanje, gde pritom ta završna stanja ne moraju biti u datom trenutku ista.

2.1 Nerodova ekvivalencija

Formalno, za dva stanja kažemo da su nerazlikujuća ako:

$$s \sim t \iff L_s = L_t \iff (\forall a \in \Sigma^*)(\delta^*(s, a) \in F \iff \delta^*(t, a) \in F)$$

Obeležavamo sa $[s]$ relaciju ekvivalencije elementa $s \in S$ i količnički skup $S/\sim$. Količnički automat nad $\mathcal{A}$ je DKA:

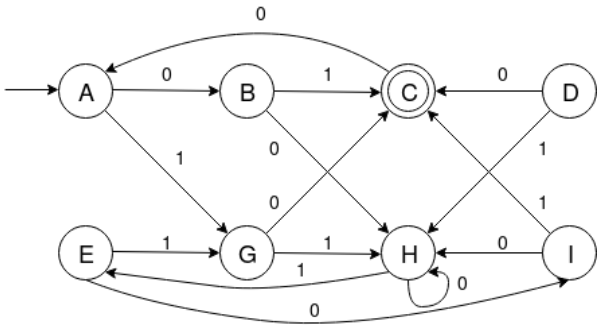
$$\mathcal{A}/\sim = (\Sigma, S/\sim, [s_0], \{[f] \mid f \in F\}, \delta^\sim)$$

za $\delta^\sim([s], a) = [\delta(s, a)]$.

Količnički automat prepoznaje isti jezik kao i polazni, i kako sadrži samo međusobno razlikujuća stanja, izomorfan je sa minimalnim automatom za dati DKA. Dakle, dovoljno je odrediti količnički automat kako bi se pronašao minimalan DKA.

2.2 Primer

Posmatramo automat sa slike 1. Sa δ označavamo funkciju prelaska. Odmah se vidi da postoje parovi stanja koji nisu ekvivalentni. Uzmimo C i H: primećujemo da je C završno stanje, dok H nije. Formalno bismo rekli da prazna niska (ϵ) razlikuje ova dva stanja: $\delta^*(C, \epsilon)$ je završno stanje, a $\delta^*(H, \epsilon)$ nije.



Slika 1: Automat sa ekvivalentnim stanjima

Posmatramo A i H. Niska ϵ ih ne razlikuje jer su oba nezavršna stanja. Niska 0 ih takođe ne razlikuje, jer je $\delta^*(A, 0) = B$, a $\delta(H, 0) = H$, gde su B, H oba nezavršna stanja. Isto važi i za nisku 1: $\delta^*(A, 1) = G$, $\delta^*(H, 1) = E$, gde pritom $G, E \notin F$. Sa druge strane, niska 01 ih razlikuje: $\delta^*(A, 01) = C$ i $\delta^*(H, 01) = E$, gde C jeste završno stanje, a E nije.

Sa druge strane, ako pogledamo A i E, primećujemo da $\delta^*(A, 1) = \delta^*(E, 1) = G$, što znači da nijedna niska koja počinje sa 1 ne može da ih razlikuje. Niska 0 ih ne razlikuje jer $\delta^*(A, 0) = B$, a $\delta^*(E, 0) = I$, gde ni I ni B nisu završna stanja. Ako proširimo nisku 0, primećujemo da $\delta^*(A, 01) = \delta^*(E, 01) = C$ i $\delta^*(A, 00) = \delta^*(E, 00) = H$, iz čega zaključujemo da su ova dva stanja ekvivalentna.

3 Hopkroftov algoritam

Hopkroftov algoritam se zasniva na nalaženju količničkog automata za dati DKA, odnosno na određivanju klasa ekvivalencije Nerodove relacije.

Ideja algoritma je da se inicijalno stanja podele u dva skupa trivijalno razlikujućih stanja: $S \setminus F$ i F, koji su respektivno skupovi nezavršnih i završnih stanja. Glavni korak je tzv. „sečenje particija". Pretpostavimo da imamo dve particije Z i Y, kao i karakter a. Kažemo da Z seče Y po karakteru a ukoliko u Y postoji:

- bar jedno stanje y za koje $\delta(y, a) \in Z$,
- bar jedno stanje y' za koje $\delta(y', a) \notin Z$

Sečenje particija bi u ovom slučaju predstavljalo podelu particije Y na dva dela:

- $Y_1 = \{y \in Y \mid \delta(y, a) \in Z\}$
- $Y_2 = \{y \in Y \mid \delta(y', a) \notin Z\}$

Razume se da je sečenje particija ništa drugo do razbijanje postojeće klase ekvivalencije na finije i egzaktnije, to jest odvajanje razlikujućih stanja u zasebne skupove. Algoritam se zaustavlja u trenutku kada ne postoji nijedna particija koja seče neku drugu particiju (ili samu sebe) po nekom slovu, što suštinski znači da jesmo došli do količničkog skupa datog automata.

U nastavku sledi pseudo-kod[11] Hopkroftovog algoritma.

```
1  Algoritam Hopkroft
2  --------------------------
3  P := {F, Q \ F};
4  W := {F};
5  while (W is not empty) do:
6     remove random element A from W;
7        for each c in Σ do:
8           let X be the set of states for which a transition on c leads to a state in A;
9           for each set Y in P for which X ∩ Y is nonempty and Y \ X is nonempty do:
10             replace Y in P by the two sets X ∩ Y and Y \ X;
11             if Y is in W then:
12                replace Y in W by the same two sets;
13             else:
14                if |X ∩ Y| ≤ |Y \ X| then:
15                   add X ∩ Y to W;
16                else:
17                   add Y \ X to W;
18          end;
19       end;
20 end;
```

P predstavlja skup do sada obrađenih particija, a W skup „za obradu", to jest u skupu W se nalaze sve particije za koje u narednim koracima proveravamo da li seku neku drugu.

Algoritam se izvršava u $O(|\Sigma|\ n\ log\ n)$ vremenu, gde je n broj stanja ulaznog automata. Ako pretpostavimo da je kardinalnost azbuke Σ konstanta, dolazimo do složenosti $O(n\ log\ n)$.

Primer 1: Posmatrajmo sliku 1:

1. Počinjemo sa $P = \{(C), (A, B, D, E, G, H, I)\}$ i $W = \{(C)\}$, kako je C jedino završno stanje.
2. Uklanjamo element A iz W tako da je $W = \{\}$ i $A = (C)$.
3. Za $c = 0$ formiramo $X_0 = (G, D)$ i proveravamo $|X_0 \cap Y \in P|$, kao i $|Y \in P \setminus X_0|$.
4. Jasno je da je $X_0 \cap (C)$ prazan skup, tako da particija (C) ne ispunjava zadati uslov.

5. Kako ga particija $Y = (A, B, D, E, G, H, I) \in P$ ispunjava, delimo je na dva podskupa: $X_0 \cap Y = (G, D)$ i $Y \setminus X_0 = (A, B, E, H, I)$ i njih ubacujemo u P umesto originalne particije: $P = \{(C), (G, D), (A, B, E, H, I)\}$.

6. Ubacujemo skup $X_0 \cap Y$ u W, znači sada je $W = \{(G, D)\}$.

7. Za $c = 1$ formiramo $X_1 = (B, I)$ i proveravamo $|X_1 \cap Y \in P|$, kao i $|Y \in P \setminus X_1|$.

8. Na prvi pogled se vidi da X_1 seče jedino particiju $Y = (A, B, E, H, I)$, koju dalje delimo na $X_1 \cap Y = (B, I)$ i $Y \setminus X_1 = (A, E, H)$ i ubacujemo ih u P umesto originalne particije: $P = \{(C), (G, D), (B, I), (A, E, H)\}$.

9. Ubacujemo skup $X_0 \cap Y$ u W, znači sada je $W = \{(G, D), (B, I)\}$.

10. Uklanjamo element A iz W tako da je $W = \{(B, I)\}$ i $A = (G, D)$.

11. Za $c = 0$ formiramo $X_0 = ()$, i kako ne postoje stanja koja preko 0 dolaze do stanja iz A, X_0 je prazno, tako da se algoritam nastavlja dalje jer sečenja particija neće biti.

12. Za $c = 1$ formiramo $X_1 = (A, E)$ i proveravamo $|X_1 \cap Y \in P|$, kao i $|Y \in P \setminus X_1|$.

13. Vidimo da $X_1 = (A, E)$ seče particiju $Y = (A, E, H)$, tako da je sada $P = \{(C), (G, D), (B, I), (A, E), (H)\}$.

14. $W = \{(B, I), (H)\}$

15. Uklanjamo element A iz W tako da je $W = \{(H)\}$ i $A = (B, I)$.

16. Za $c = 0$ formiramo $X_0 = (A, E)$, i kako X_0 ne seče nijednu particiju, algoritam se nastavlja.

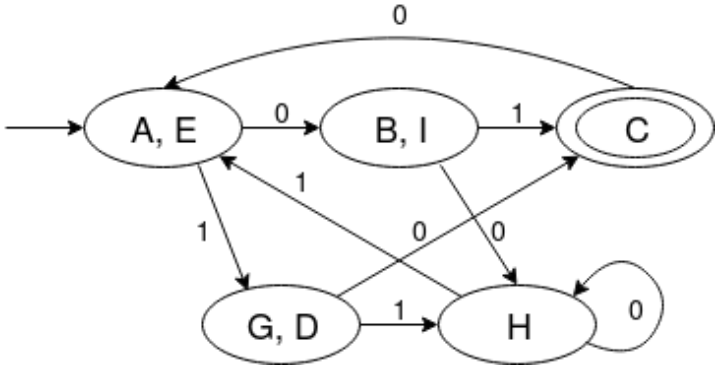
17. Za $c = 1$ formiramo $X_1 = ()$, i kako ne postoje stanja koja preko 1 dolaze do stanja iz A, X_0 je prazno, tako da se algoritam nastavlja dalje jer sečenja particija neće biti.

18. Uklanjamo element A iz W tako da je $W = \{\}$ i $A = (H)$.

19. Formiramo $X_0 = (H, B, I)$ i $X_1 = (G, D)$, primećujemo da ne seku nijednu particiju i nastavljamo algoritam.

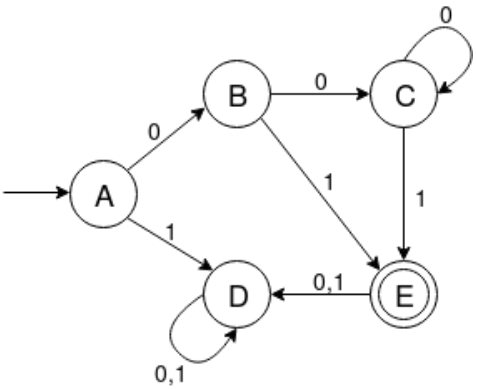
20. Kako je $W = \{\}$, algoritam se završava.

Zaista, dobijene klase ekvivalencije predstavljaju stanja minimalnog DKA 2 za dat početni automat:



Slika 2: Minimalni DKA

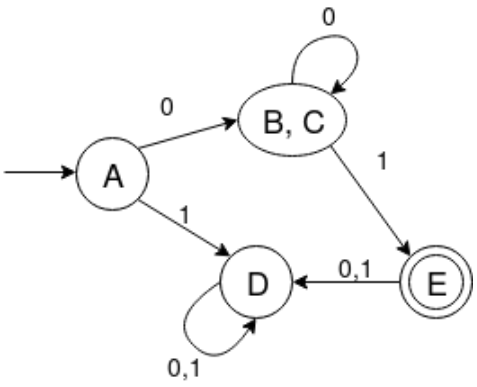
Primer 2: Posmatrajmo sliku 3:



Slika 3: Automat sa ekvivalentnim stanjima

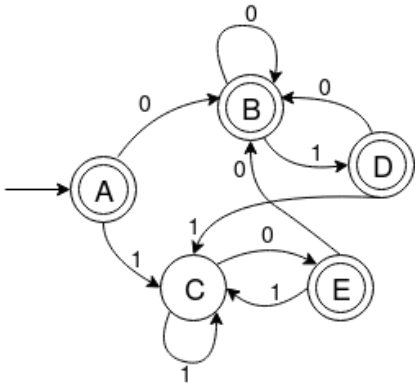
1. $P = \{(E), (A, B, C, D)\}$ i $W = \{(E)\}$.
2. $W = \{\}$ i $A = (E)$.
3. $c = 0$, $X_0 = ()$.
4. $c = 1$, $X_1 = (B, C)$, $Y = (A, B, C, D)$, $X_1 \cap Y = (B, C)$, $Y \setminus X_1 = (A, D)$.
5. $P = \{(E), (A, D), (B, C)\}$, $W = \{(B, C)\}$.
6. $W = \{\}$ i $A = (B, C)$
7. $c = 0$, $X_0 = (A, B, C)$, $Y = (A, D)$, $X_0 \cap Y = (A)$, $Y \setminus X_0 = (D)$.
8. $P = \{(E), (A), (D), (B, C)\}$, $W = \{(A)\}$.
9. $c = 1$, $X_1 = ()$.
10. $W = \{\}$ i $A = (A)$.
11. $c = 0$, $X_0 = ()$.
12. $c = 1$, $X_1 = ()$.
13. $W = \{\}$.

Dobijeni minimalni automat 4:



Slika 4: Minimalni DKA

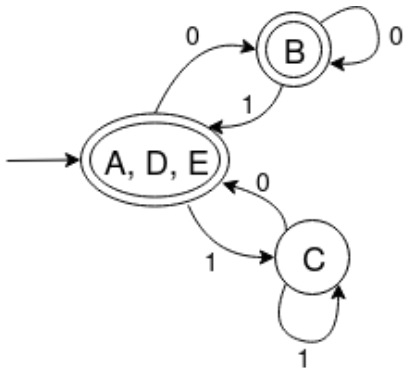
Primer 3: Posmatrajmo sliku 5:



Slika 5: Automat sa ekvivalentnim stanjima

1. $P = \{(C), (A, B, D, E)\}$ i $W = \{(C)\}$.
2. $W = \{\}$ i $A = (C)$.
3. $c = 0$, $X_0 = ()$.
4. $c = 1$, $X_1 = (A, D, C, E)$, $Y = (A, B, D, E)$, $X_1 \cap Y = (A, D, E)$, $Y \setminus X_1 = (B)$.
5. $P = \{(C), (A, D, E), (B)\}$, $W = \{(B)\}$.
6. $W = \{\}$ i $A = (B)$
7. $c = 0$, $X_0 = (A, B, D, E)$
8. $c = 1$, $X_1 = ()$.
9. $W = \{\}$

Dobijeni minimalni automat 6:



Slika 6: Minimalni DKA

4 Zaključak

U prvom delu ovog rada smo se upoznali sa glavnim pojmovima i terminima potrebnim za razumevanje algoritma.
U drugom delu predstavili smo glavnu ideju Hopkroftovog algoritma koji se pokazao kao jako efikasan u praksi. Uprkos tome što je znatno napredniji od ostalih poznatih algoritama, često biva izostavljen iz literature i gradiva, ili biva pomenut samo u kratkim crtama.
U trećem delu smo naveli nekoliko primera koji bi čitaocu pomogli da intuitivno shvati kako algoritam radi.

Literatura

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, *Compilers: principles, techniques, & tools*, Addison Wesley, 2nd edition, 2006.

[2] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, *Introduction to automata theory, languages, and computation*, Pearson, 3rd edition, 2013.

[3] Erin van der Veen, *The practical performance of automata minimization algorithms*, Radboud University, Nijmegen, Netherlands, 2017.

[4] Jean Berstel, Luc Boasson, Olivier Carton, *Hopcroft's automaton minimization algorithm and Sturmian words*, Kiel, Germany, 2008.

[5] G. Castiglione, A. Restivo, M. Sciortino, *Hopcroft's algorithm and tree-like automata*, 2010.

[6] Timo Knuutila *Re-describing an algorithm by Hopcroft*, Department of Computer Science, University of Turku, Turku, Finland, 1997.

[7] Andrei Păun, *On the Hopcroft's minimization algorithm*, University of Bucharest, Bucharest, Romania, 2007.

[8] Javier Esparza, *Automata theory: an algorithmic approach, lecture notes*, 2017.

[9] Manuel Baclet, Claire Pagetti, *Around Hopcroft's algorithm*, Toulouse, France

[10] Yingjie XU, *Describing an nlogn algorithm for minimizing states in deterministic finite automaton*, 2009.

[11] Wikipedia article, *DFA minimization*, link

[12] Nikola Ajzenhamer, Anja Bukurov, *Prevođenje programskih jezika: beleške sa predavanja*, Beograd, Srbija, 2015.