Opdracht 1

Topics on Parsing and Formal Languages - fall 2010

Rick van der Zwet
<hvdzwet@liacs.nl>

8 december 2010

Samenvatting

Dit schrijven zal uitwerkingen van opgaven behandelen uit het boek [JS2009] gebruikt bij het college. In deze opdracht zullen zeven opgaven (3,20,22,47,54,68,69) van hoofdstuk 3 behandeld worden.

1 Opgave 3.3

Als $L \subseteq \Sigma^*$ is regulier dan is de taal

$$2L := a_1, a_1, a_2, a_2, \ldots, a_k, a_k \ : \ elke\ a_i \in \Sigma\ en\ a_1 a_2 \cdots a_k \in L \qquad (1)$$

regulier. Zie dat er een 'verdubbeling' optreed van symbolen, dit gedrag is de modelleren in een *DFA* . Omdat L regulier is bestaat er een *DFA* $M = (Q, \Sigma, \delta, q_0, F)$ die L beschrijft. Construeer nu een nieuwe *DFA* P die de nieuwe taal $2L$ gaat beschrijven, neem hiervoor het alfabet (Σ), en de begintoestand q_0 over van L. De toestanden (Q) worden verdubbeld. *voor alle* $q \in Q$: *voeg* q' *toe aan* Q. Neem ook de transities over van M, maar maak aanpassingen zodanig dat de nieuwe toestanden ook gelezen worden. Dus $\delta(q, x)$ wordt $\delta(q, q'), \delta(q', x)$. De acceptatie toestand F is gelijkt aan de oude acceptatie toestand.
De nieuwe *DFA* P beschrijft $2L$ en dus is $2L$ regulier. □

2 Opgave 3.20

Laat $\Sigma = \{0, 1\}$ zijn. Een voorbeeld van de taal $L \subseteq \Sigma^*$ voor welke geldt dat, de Myhill-Nerode[JS2009][pg. 77–81] gelijkheid relatie R_L de eigenschap heeft dat elk woord in Σ^* zijn eigen equivalentie klasse is. Dit wilt zeggen dat xR_ly dan en slechts dan voor alle $z \in \Sigma^*$, is er een $xz \in L$ dan en slechts als $yz \in L$. Omdat hier gezocht wordt naar een taal waarbij de woorden allemaal equivalentie klassen op-zich zijn, zal alle elementen met elkaar vergeleken kunnen worden door middel van R_L en hier mogen geen positieve 'gevallen' uitkomen.

De (triviale) taal $L = 01, 10$ is een voorbeeld hiervan. Beiden woorden hebben hun eigen klasse. Als $x = 0$ en $y = 1$, bestaat er een $z = 1$, zodanig dat $xz \in L, yz \notin L$.

De taal $M = \{0^* : |M|_0 \; is \; een \; priemgetal\}$ is een ander voorbeeld. Als we kijken naar aR_lb kan er altijd een z gevormd worden zodanig dat $az \in M, bz \notin M$. Bijvoorbeeld $a = 0^3, b = 0^5, z = 0^4$. Deze eigenschap wordt afgedwongen door het 'feit' dat er geen relatie bestaat tussen de lengtes tussen opeenvolgende priemgetallen.[1]

3 Opgave 3.22*

Om aan te tonen dat een *2DFA* exponentieel meer expressief is dan een *DFA* voor bepaalde talen kijken we naar de volgende taal; Laat n een integer ≥ 1 en $F_n \subseteq \{0, 1, 2, 3, 4\}^*$ als volgende gedefinieerd:

$$F_n = \{3\ 0^{i_1}\ 1\ 0^{i_2} \cdots 1\ 0^{i_n}\ 2^k\ 0^{i_k}\ 4 : 1 \leq k, j, i_j \leq n\} \tag{2}$$

Als eerste kan de F_n door een *2DFA* van $O(n)$ toestanden geaccepteerd worden, welke aan te tonen is door eerst te kijken naar de eigenschappen van F_n hierbij is de zien dat de 0-reeks van de lengte i_k zowel voor als na de 2^k moet voorkomen. Waarbij 2^k aangeeft, waar deze 0-reeks gevonden kan worden bij de plek k. De *2DFA* moet dus twee dingen doen. Het aantal keer nul op twee

[1] Ik heb dit voor gemak aangenomen, de praktijk is echter dat is nog een groot wiskundig vraagstuk is.

plekken vergelijken. Waarbij 1 plek vast staat en de tweede plek variabele wordt gedefinieerd. De aanpak kan als volgt gezien worden:

```
1. controleren of |1| <= 2de 0-reeks dmv 'knopen-strook' (knopen 0.X en 1.X).
2. |2| tellen onthouden dmv 'knopen-strook' (knopen 2.X).
3. naar gewenste locatie lopen (knopen 3.X).
4. |0| tellen onthouden dmv 'knopen-strook' (knopen 4.X).
5. naar begin 2de 0-reeks lopen (knopen 5.X).
6. 0-reeks vergelijken (knopen 6.X).
```

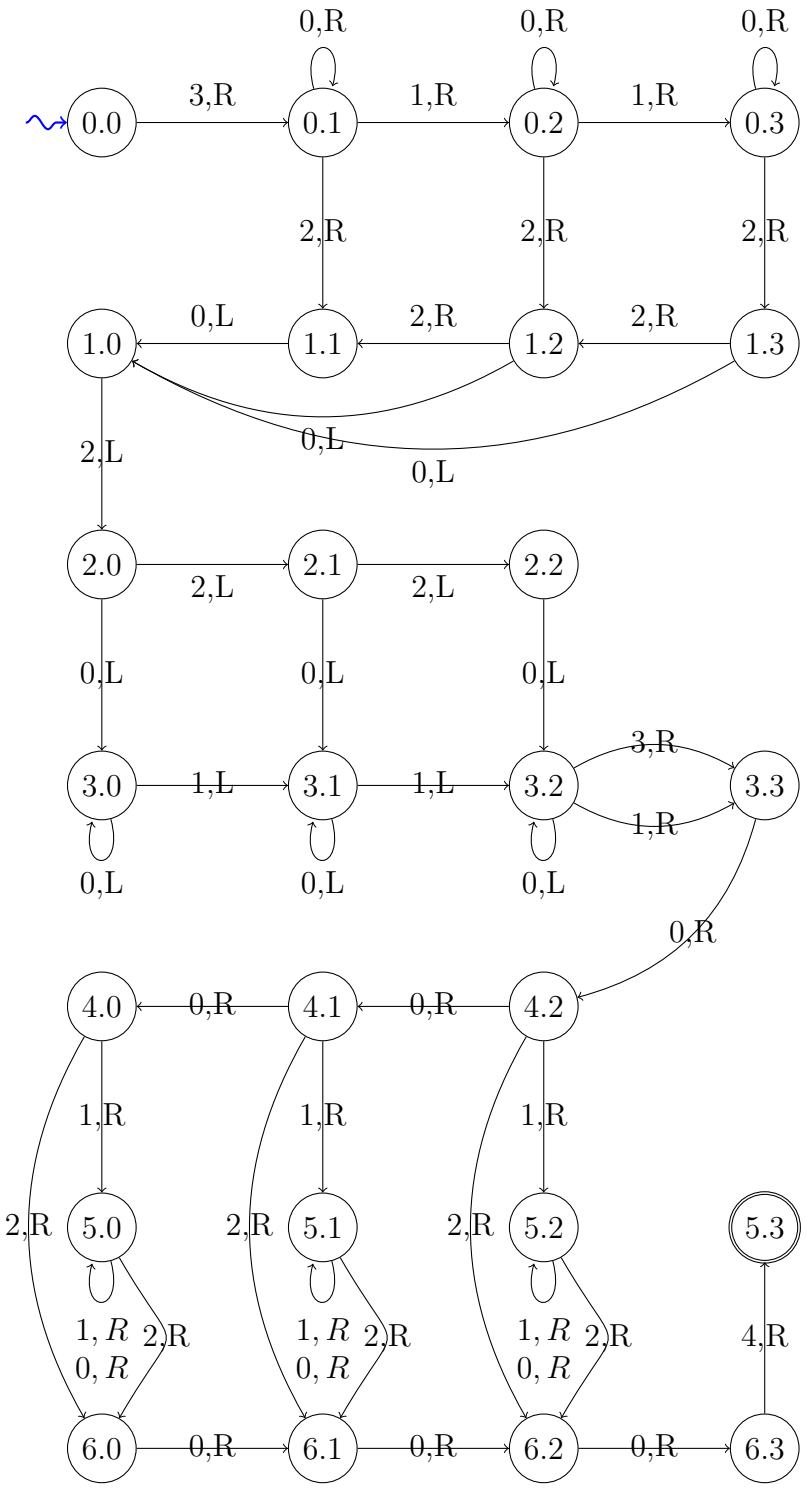
Voor deze aanpak zijn 3 'knopen-stroken' nodig, 3 'transitie-lagen' en 1 controle strook. In totaal dus in ongeveer $O(7n)$ knopen. Figuur 1 laat de *2DFA* zien.

Om te laten zien dat een *DFA* minstens n^n toestanden nodig heeft moet je gebruik maken van het feit dan een *DFA* niet terug kan lopen. Het zal dus een 'geheugen' moeten maken om het maximale 0 woord voor de 'splitsing' (2^k) te kunnen onthouden om zo te kijken of het overeen komt het 0 woord na de 'splitsing'. Omdat je voor elke F_i waarbij $1 \geq i \geq n$ minimaal n toestanden nodig hebt om te kunnen tellen en hiervan weer n unieke sets bestaan, heb je dus minimaal n^n toestanden nodig.

4 Opgave 3.47

Om te bewijzen dat 'de klasse van talen geaccepteerd door *DFA* ' =¿ 'de klasse van talen gespecificeerd door de reguliere expressies'[JS2009][Theorem 1.4.2, (b) =¿ (a)] aan de hand van het volgende voorbeeld: Laat $M = (Q, \Sigma, \delta, q_1, F)$ een *DFA* zijn, waarbij $Q = \{q_1, q_2, \ldots, q_n\}$. Definieer nu $R_{i,j,k}$ als de taal van alle woorden die van toestand i naar toestand j gaan zonder door toestanden te gaan die hoger als k genummerd zijn.

Om dit te doen is een recursieve formule nodig, welke $R_{i,j,k}$ als invoer heeft. a) Kijk welke transities er mogelijk zijn in δ met toestand i als begin positie. Plaats deze op de 'stapel'. b) Werk de elementen op de stapel stuk voor stuk af volgens dezelfde methodiek. Stop pas als de δ resultaat j bevat. Let erop dat oneindige herhalingen gedetecteerd moeten worden om het algoritme te



Figuur 1: Opdracht 2.22a uitgewerkt voor $1 \leq n \leq 3$, niet geldige stappen zijn niet genoemd en moet allen naar een 'trap' knoop gestuurd worden. Uitbreiden dmv expanderen van 'traviale' knopen

laten termineren. Je kan het zien als het volledig doorzoeken van een boom met in de wortel de knoop i.

5 Opgave 3.54

Laat $\Sigma = \{1, 2, 3, \ldots, n\}$ zijn en definieer:

$$L_n = \{w \in \Sigma^* : |w|_i = 1 \; voor \; alle \; i\} \tag{3}$$

Dit zijn dus de woorden waarbij alle elementen precies één keer in voor komen. Bijvoorbeeld $L_3 = \{123, 132, 213, 231, 312, 321\}$.

Om te laten zijn dat er minimal een reguliere expressie van lengte 2^{n-1} nodig is om L_n te specificeren, moet de *Myhill-Nerode* stelling toegepast worden. Vanwege de eigenschap dat prefix uniek is, zat het nooit mogelijk worden om aan beiden woorden hetzelfde toe te voegen zodanig dat ze in elkaars klasse terecht komen. (elk 'bit' informatie is relevant). Voor alle klasse zal dus apart gekeken worden of aan de eisen voldaan wordt. Om dus alle getallen n van de string te controleren of zij uniek is zijn minimaal 2^{n-1} toestanden nodig.

Echter voor $\overline{L_n}$ is wel een reguliere expressie te vinden en wel in de grootte $O(n^2)$. Door simpelweg voor elke $i \in \Sigma^*$ een reguliere expressie te maken van de vorm $x^* \; i \; x^* \; i \; x^*$ waarbij $x = Sigma^* - i$.

6 Opgave 3.68

Voor een woord $w \in \Sigma^*$, is een palc(w) de korte palindroom x zodanig dat w een prefix is van x. en palc(L) = $\bigcup_{w \in L} \{palc(w)\}$.

Om te laten zien dat palc(w) = $wt^{-1}w^R$, waarbij wt^{-1} het woord is w waar het suffix t eraf gehaald is en t de langste palindroom suffix van w is, moet er gebruik gemaakt worden van een tegenstelling. Als w nog een palindroom aan het einde zal bevatten uu, dan ziet x er als volgt uit $wuuuuw$, dit is niet de korte palindroom (ww), welke $uuuu$ er nog makkelijk uit weggehaald had kunnen worden. Een willekeurige suffix van w kan ook geen palindroom vormen die nog 'weggesneden' kan worden, omdat hij anders al in den beginne een palindroom had moeten zijn.

Als L is regulier, dan is palc(L) ook regulier, welke er een unieke 'vertaling' van een woord w naar zijn palc(w). De nieuwe woorden zullen in het slechtste geval evenveel blijven, maar de taal kan ook kleiner worden.

De andere kant op geldt dit niet, als L regulier is, dan is het onbeslist of palc$^{-1}(L)$ ook regulier is. Het kan namelijk zeer goed dat een (ingewikkelde) functie die woorden genereerde welke een palindroom zijn en die aan een vast prefix (p) toegevoegd. De *palc* functie zal deze allen naar één woord omzetten, welke dan regulier is.

7 Opgave 3.69

Laat $x_1, x_2, \ldots, x_k \in \Sigma^*$. Om te laten zien dat $\Sigma^* - x_1^* x_2^* \cdots x_k^*$ eindig is dan en slechts als $|\Sigma| = 1$ en $\gcd(|x_1|, |x_2|, \ldots, |x_k|) = 1$ moet er een paar dingen bewezen worden a) aantonen dat het niet geldt voor de tegenvoorbeelden a) $gcd > 1$ b) $|\Sigma| > 1$ en tevens moet aangetoond worden waarom de eindigheid voor dit specifieke geval geldt.

Merk op dat met een gcd van 1 alle getallen in de verzameling {1, priemgetallen} zitten. Bij een alfabet van 1 letter hoeft er enkel maar gesproken worden over lengtes. Standaard zitten alle lengtes in de taal. Aangezien de 'deellengtes' $(x_1, x_2, \ldots, x_k)$ nul of meer keer in de verzameling in de verzameling mogen zitten. Zal je uiteindelijk overblijven met een eindige set lengtes. In het geval dat er lengte 1 gevonden wordt, zal zelfs de lege set het antwoord zijn.

a) Als $|\Sigma| > 1$, bijvoorbeeld $\{1, 2\}$ dan kan er een oneindige verzameling gemaakt worden, door $x_1, x_2, \ldots, x_k \in 1^*$. De $2*$ zal dan een oneindige verzameling vormen.

b) $gcd > 1$, bijvoorbeeld 2 dan kan er een oneindige verzameling gemaakt worden, doordat niet alle woorden gemaakt kunnen worden. Enkel worden van 'even' lengte in dit geval. Waardoor de 'oneven' woorden een eindige string gaan vormen. Bij grotere waardes (r) kunnen ook 'groepen' (de $|r| * -1$) bijvoorbeeld niet meer bereikt worden, welke dan tot een oneindige verzameling kunnen groeien.

Referenties

[JS2009] Jeffrey Shallit, *A second course in formal languages and automata theory* , *Cambridge University Press*, 2009.