Automatentheorie en Formele Talen

docent:

H. Zantema
HG kamer 6.73
tel 040 - 2472749
email: `h.zantema@tue.nl`

college
instructie
inleveropgaven
tentamen

Informatie:

`www.win.tue.nl/~hzantema/aft.html`

2

(voorbeeld koffie-automaat)

Hierin hebben we te maken met

- Toestanden waarin de automaat zich kan bevinden: **(internal) states**

  In totaal zijn er eindig veel dergelijke toestanden, schrijf Q voor de verzameling van deze toestanden

- Invoersymbolen

  Ook hiervan zijn er eindig veel; de verzameling Σ van al deze invoersymbolen heet het **input alphabet**

- Toestandsovergangen

  Bij elke toestand en elk invoersymbool is er een nieuwe toestand waar je dan in terecht komt

  Wiskundig beschreven door een

  transitiefunctie $\delta : Q \times \Sigma \to Q$

3

- Een speciale toestand $q_0 \in Q$ om in te beginnen: de **initial state**

- Een verzameling $F \subseteq Q$ eindtoestanden **(final states)**

Deze vijf onderdelen $(Q, \Sigma, \delta, q_0, F)$ samen heten een **dfa**:

deterministic finite accepter

of

deterministic finite automaton

4

De manier waarop we een dfa tekenen heet een **transitiegraaf**, met

knopen (vertices) en

pijlen (arrows), elk met een label uit Σ

De beginknoop q_0 heet dan **initial vertex** en de eindknopen in F heten **final vertices**

De beginknoop wordt met een extra binnenkomende pijl getekend

De eindknopen worden met een dubbel rondje getekend

Uiteindelijk zullen we uitbreidingen van dit automaatbegrip beschrijven (Turing machines) die even krachtig (maar niet zo efficient) zijn als een moderne computer met onbeperkt geheugen

---

5

(Formele) Talen

Een taal is een verzameling woorden en zinnen die elk een betekenis kunnen hebben

Elke zin is een rij symbolen uit een alfabet Σ

Bij een taal kun je beschouwen

- de **syntax**: de vorm, welke zin zit wel en welke zin zit niet in de taal

- de **semantiek**: de betekenis van de zinnen

Wij zullen ons hier concentreren op de syntax

Dus:

Een **(formele) taal** is een verzameling zinnen

6

Wiskundig gezegd:

Een (formele) taal is een deelverzameling van Σ^* voor zeker alfabet Σ

Hierin is een alfabet een eindige verzameling Σ van symbolen, en is Σ^* de verzameling van eindige rijen (**strings**) van dergelijke symbolen

Voorbeeld:

$\Sigma = \{a, b\}$

$\{ab, aab, bbaaabb\}$ is een eindige taal

$\{aba, abba, abbba, abbbba, \ldots\} =$
$\{ab^n a | n \geq 1\}$ is een oneindige taal

Dit vak gaat voor een groot deel over het precies beschrijven van (meestal oneindige) talen: programmeertalen, specificatietalen, ...

7

Strings

Concatenatie:

Als $v = a_1 a_2 \cdots a_n$ en $w = b_1 b_2 \cdots b_m$ dan is

$$vw = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$$

Hiermee kunnen we b^k inductief definiëren door

$b^1 = b$ en

$b^{k+1} = b^k b$ voor $k \geq 1$

Als $v = a_1 a_2 \cdots a_n$ dan is $|v| = n$ de **lengte** van v

Altijd geldt: $|vw| = |v| + |w|$

8

We beschouwen ook een **lege string** λ van lengte 0

Voor elke string v geldt $v\lambda = \lambda v = v$; λ is een **neutraal element** voor string concatenatie, net zoals 0 een neutraal element is voor optelling en 1 voor vermenigvuldiging

Voor elk symbool a definiëren we

$$a^0 = \lambda$$

Als $v = a_1 a_2 \cdots a_n$ dan is $v^R = a_n a_{n-1} \cdots a_1$ de omgekeerde (**reverse**) van v

Altijd geldt: $(v^R)^R = v$

9

Een aaneengesloten deel van een string heet een **substring**, oftewel v is een substring van w als

$$w = uvu'$$

voor zekere strings u, u'

Als $w = uv$ dan heet u een beginstuk (**prefix**) van w, en v een eindstuk (**suffix**) van w

Omdat we een taal gedefinieerd hebben als een verzameling strings hebben alle begrippen uit de verzamelingenleer: $\in, \subseteq, \cap, \cup, -, \ldots$ ook hun betekenis voor talen

De begrippen concatenatie en reverse gaan we nu ook voor talen definiëren:

$$L_1 L_2 = \{xy | x \in L_1 \wedge y \in L_2\}$$

$$L^R = \{x^R | x \in L\}$$

10

Het **complement** $\overline{L}$ van een taal L is gedefinieerd door

$$\overline{L} = \Sigma^* - L$$

oftewel de verzameling van alle strings die niet in L zitten

Verder definiëren we

$L^0 = \{\lambda\}$

$L^1 = L$

$L^{n+1} = L^n L$ voor elke $n \geq 1$

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup L^3 \cup \cdots$$

11

Voorbeeld:

Als $L = \{ab^n | n \geq 0\} = \{a, ab, abb, \ldots\}$, dan is

$L^2 = \{ab^n ab^m | m, n \geq 0\}$

Let wel: hier geldt

$$L^2 = \{vw | v, w \in L\} \neq \{vv | v \in L\}$$

Bij dezelfde L:

$L^+ = a\{a, b\}^*$, oftewel de verzameling van alle strings die met a beginnen

Hoe weet je nou zeker dat twee zulke talen echt gelijk zijn?

12

Met een wiskundig bewijs

Verzamelingenleer:

Te bewijzen $A = B$

Bewijs:

Kies $x \in A$ willekeurig, bewijs dat $x \in B$
Kies $x \in B$ willekeurig, bewijs dat $x \in A$
Dan geldt $A = B$

13

Stelling

Als $L = \{ab^n | n \geq 0\}$, dan is $L^+ = a\{a, b\}^*$

Bewijs:

Kies $x \in L^+$ willekeurig
Dan is $x = x_1 x_2 \cdots x_n$ voor $x_i \in L$, $n \geq 1$
Omdat $x_1 \in L$ begint x_1 met a, dus ook x
Dus $x \in a\{a, b\}^*$

Kies omgekeerd $x \in a\{a, b\}^*$ willekeurig
Laat $n =$ aantal a's in x
Dan $x = ab^{k_1} ab^{k_2} \cdots ab^{k_n}$
met $k_i \geq 0$ voor elke $i = 1, 2, \ldots, n$
Kies $x_i = ab^{k_i}$ voor elke $i = 1, 2, \ldots, n$
Dan $x_i \in L$ voor elke $i = 1, 2, \ldots, n$
en $x = x_1 x_2 \cdots x_n$
Dus $x \in L^n$
Dus $x \in L^+$

Einde bewijs

14

Grammatica's

Grammatica's (grammars) vormen een manier om talen te definiëren

Voorbeeld:

⟨zin⟩	→	⟨lidw⟩⟨znw⟩⟨werkw⟩⟨lidw⟩⟨znw⟩
⟨lidw⟩	→	de
⟨lidw⟩	→	een
⟨znw⟩	→	poes
⟨znw⟩	→	hond
⟨znw⟩	→	koe
⟨werkw⟩	→	slaat
⟨werkw⟩	→	aait

Met deze spelregels kun je de ⟨zin⟩

de poes aait een koe

opbouwen

15

⟨zin⟩

⇒ ⟨lidw⟩⟨znw⟩⟨werkw⟩⟨lidw⟩⟨znw⟩

⇒ de ⟨znw⟩⟨werkw⟩⟨lidw⟩⟨znw⟩

⇒ de poes ⟨werkw⟩⟨lidw⟩⟨znw⟩

$\Rightarrow$ de poes aait ⟨lidw⟩⟨znw⟩

$\Rightarrow$ de poes aait een ⟨znw⟩

$\Rightarrow$ de poes aait een koe

Zo'n stel spelregels heet een **grammatica**

Preciezer gezegd:

16

Een grammatica $G = (V, T, S, P)$ bestaat uit

- een eindige verzameling V van **variabelen** of **non-terminals**

  (in dit voorbeeld: ⟨zin⟩, ⟨lidw⟩, ⟨znw⟩, ⟨werkw⟩)

- een eindige verzameling T van **terminal symbolen** of kortweg terminals

  (in dit voorbeeld: de, een, poes, hond, koe, slaat, aait)

- een speciaal symbool $S \in V$, de **startvariabele**

  (in dit voorbeeld: ⟨zin⟩)

- een eindige verzameling P van **producties** $x \to y$ met
  - $x \in (V \cup T)^+$ en
  - $y \in (V \cup T)^*$

  (meestal en in dit voorbeeld: $x \in V$)

17

Een **afleidingsstap** op een string is: vervang een substring x door y voor een productie $x \to y$

Preciezer gezegd: als $x \to y$ een productie is, dan is er voor elke $u, v \in (V \cup T)^*$ een afleidingsstap

$$uxv \Rightarrow uyv$$

Er is een **afleiding (derivation)** $u \Rightarrow^* v$ van u naar v als v uit u verkregen kan worden door 0 of meer afleidingsstappen

Preciezer gezegd: er bestaan $w_1, w_2, \ldots, w_n$ met

$$u = w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n = v$$

18

Definitie:

Voor een grammatica $G = (V, T, S, P)$ is

$$L(G) = \{w \in T^* | S \Rightarrow^* w\}$$

de **taal gegenereerd door** G

Voorbeeld:

$$G = (\{S\}, \{a, b\}, S, P)$$

waarbij P bestaat uit

$$S \to aSb$$

$$S \to \lambda$$

Dan geldt

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

dus $S \Rightarrow^* aaabbb$ en $aaabbb \in T^*$
dus $aaabbb \in L(G)$

19

Door n keer de eerste productie toe te passen en daarna een keer de tweede, is in te zien dat $a^n b^n \in L(G)$ voor elke $n \geq 0$

Omgekeerd is er geen andere manier om een afleiding

$$S \Rightarrow^* w$$

te maken met $w \in T^*$, dus

$$L(G) = \{a^n b^n | n \geq 0\}$$

20

Bij het definiëren van $G = (V, T, S, P)$ gebruiken we hoofdletters voor variabelen, en kleine letters voor terminals

We geven alleen de producties en laten impliciet dat V de verzameling van hoofdletters is die daarin voorkomt en T de verzameling kleine letters, en S het startsymbool

Voorbeeld:

G_1:			G_2:		
S	$\rightarrow$	Ab	S	$\rightarrow$	aSb
A	$\rightarrow$	aAb	S	$\rightarrow$	b
A	$\rightarrow$	λ			

Nu geldt

$$L(G_1) = L(G_2) = \{a^n b^{n+1} | n \geq 0\}$$

Twee grammatica's G_1 en G_2 heten **equivalent** als $L(G_1) = L(G_2)$

21

Alternatieve conventie:
zet variabelen tussen $\langle$ en $\rangle$

Voorbeeld:

$\langle$stm$\rangle$	$\rightarrow$	$\langle$var$\rangle$:= $\langle$expr$\rangle$
$\langle$stm$\rangle$	$\rightarrow$	$\langle$stm$\rangle$; $\langle$stm$\rangle$
$\langle$stm$\rangle$	$\rightarrow$	begin $\langle$stm$\rangle$ end
$\langle$stm$\rangle$	$\rightarrow$	if $\langle$cond$\rangle$ then $\langle$stm$\rangle$ else $\langle$stm$\rangle$
$\langle$stm$\rangle$	$\rightarrow$	if $\langle$cond$\rangle$ then $\langle$stm$\rangle$
$\langle$stm$\rangle$	$\rightarrow$	while $\langle$cond$\rangle$ do $\langle$stm$\rangle$
$\langle$cond$\rangle$	$\rightarrow$	$\cdots$
$\langle$var$\rangle$	$\rightarrow$	$\cdots$
$\langle$expr$\rangle$	$\rightarrow$	$\cdots$
$\cdots$	$\rightarrow$	$\cdots$

B(ackus) N(aur) F(orm)

schrijf $A ::= u|v|w$ in plaats van

$$A \rightarrow u$$

$$A \rightarrow v$$

$$A \rightarrow w$$

22

Voorbeeld:

$$\begin{array}{lrl} G: & S & \rightarrow & SS \\ & S & \rightarrow & \lambda \\ & S & \rightarrow & aSb \\ & S & \rightarrow & bSa \end{array}$$

kun je korter schrijven als

$$S ::= SS \mid \lambda \mid aSb \mid bSa$$

Deze grammatica genereert precies alle strings waarin precies evenveel a's als b's voorkomen

Preciezer gezegd:

$$L(G) = \{w | n_a(w) = n_b(w)\}$$

waarbij n_a, n_b het aantal a's resp. b's aangeeft

23

Bewijsschets:

Als $w \in L(G)$ dan geldt $n_a(w) = n_b(w)$, want de enige producties waarmee a en b symbolen worden ingevoerd zijn $S \rightarrow aSb$ en $S \rightarrow bSa$, en daarmee worden steeds evenveel a's alls b's ingevoerd

Omgekeerd moeten we bewijzen dat als $n_a(w) = n_b(w)$, dan $w \in L(G)$

Dit doen we met inductie naar de lengte:

Bewering:

als $|w| = k$ en $n_a(w) = n_b(w)$, dan $w \in L(G)$

Voor $k = 0$ is dit waar

Voor $k > 0$ onderscheiden we 4 mogelijkheden:

24

1. $w = avb$

   Vanwege $n_a(w) = n_b(w)$ geldt $n_a(v) = n_b(v)$, vanwege de inductiehypothese geldt $v \in L(G)$

   Dus $S \Rightarrow aSb \Rightarrow^* avb = w$

   Dus $w \in L(G)$

2. $w = bva$: net zo

3. $w = ava$

   Vanwege $n_a(w) = n_b(w)$ geldt $n_a(v) = n_b(v) - 2$

   Dan is $v = v_1v_2$ met $n_a(v_1) = n_b(v_1) - 1$ en $n_a(v_2) = n_b(v_2) - 1$ (ga na)

   Dus $n_a(av_1) = n_b(av_1)$ en $n_a(v_2a) = n_b(v_2a)$

   Vanwege de inductiehypothese geldt

   $av_1 \in L(G)$ en $v_2a \in L(G)$

   Dus $S \Rightarrow SS \Rightarrow^* av_1S \Rightarrow^* av_1v_2a = w$

   Dus $w \in L(G)$

4. $w = bvb$: net zo

Einde Bewijs

25

Terug naar automaten $\cdots$

Net als een grammatica definieert een dfa ook een taal:

Als $v = a_1a_2 \cdots a_n \in \Sigma^*$ kun je een pad in de transitiegraaf volgen door te beginnen bij q_0 en achtereenvolgens de pijlen te volgen gelabeld met $a_1, a_2, \ldots, a_n$

Als de laatst bereikte knoop een eindknoop is, zit v in de taal, en anders niet

Precieser gezegd, we breiden $\delta : Q \times \Sigma \to Q$ uit tot $\delta^* : Q \times \Sigma^* \to Q$, inductief gedefinieerd door

$$\delta^*(q, \lambda) = q$$

$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

Dit is inderdaad een uitbreiding, want

$$\delta^*(q, a) = \delta^*(q, \lambda a) = \delta(\delta^*(q, \lambda), a) = \delta(q, a)$$

26

Nu definiëren we voor een dfa

$$M = (Q, \Sigma, \delta, q_0, F)$$

de taal

$$L(M) = \{w \in \Sigma^* | \delta^*(q_0, w) \in F\}$$

Dit heet de **taal geaccepteerd door** M

Voorbeeld:

$M = (Q, \Sigma, \delta, q_0, F)$ met

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\delta(q_0, a) = q_1, \delta(q_0, b) = q_2$
- $\delta(q_1, a) = q_2, \delta(q_1, b) = q_1$
- $\delta(q_2, a) = q_2, \delta(q_2, b) = q_2$
- $F = \{q_1\}$

Dan is $L(M) = \{ab^n | n \geq 0\}$

Zo'n toestand q_2 van waaruit nooit meer een eindtoestand te bereiken is heet een **trap state**

27

Definitie:

Een taal L heet **regulier** als er een dfa M bestaat met $L(M) = L$

We hebben gezien dat $\{ab^n | n \geq 0\}$ een reguliere taal is

In het boek staat uitgewerkt dat $\{a^n b | n \geq 0\}$ een reguliere taal is

Later zullen we bewijzen dat $\{a^n b^n | n \geq 0\}$ geen reguliere taal is

---

28

Stelling

Als L een reguliere taal is dan is $\overline{L}$ ook een reguliere taal

Bewijsschets:

Laat $M = (Q, \Sigma, \delta, q_0, F)$ een dfa zijn met $L(M) = L$

Dan is $M' = (Q, \Sigma, \delta, q_0, Q - F)$ een dfa zijn met $L(M') = \overline{L}$

Einde bewijs

29

Tot nu toe was er bij elke string precies één pad waarmee je vanuit het startsymbool de automaat kon doorlopen:

de automaat is **deterministisch**

Dit komt omdat er bij elke knoop in Q en elke $a \in \Sigma$ precies één pijl vanuit die knoop is gelabeld met a: δ is een functie van $Q \times \Sigma$ naar Q

We willen nu toestaan dat er soms meerdere dergelijke pijlen zijn, of helemaal geen

Ook willen we **lege stappen** toestaan: stappen die je in de automaat mag maken zonder een symbool uit de string op te eten

Dit noemen we een **nfa:**

non-deterministic finite accepter

30

De definitie is dezelfde als die van dfa afgezien van δ, nu is

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$$

Hierin is 2^Q (ook wel $\mathcal{P}(Q)$) de power set van Q: de verzameling van alle deelverzamelingen van Q

Voor $q, r \in Q$ en $a \in \Sigma \cup \{\lambda\}$ is er een pijl van q naar r gelabeld met a precies dan als $r \in \delta(q, a)$

Voor dfa's hadden we gedefinieerd

$$L(M) = \{w \in \Sigma^* | \delta^*(q_0, w) \in F\}$$

Voor nfa's definiëren we

$$L(M) = \{w \in \Sigma^* | \delta^*(q_0, w) \cap F \neq \emptyset\}$$

31

Daartoe moeten we eerst nog definiëren

$$\delta^* : Q \times \Sigma^* \to 2^Q$$

Dit doen we recursief:

$$\delta^*(q, \lambda) = \{q\} \cup \delta(q, \lambda) \cup \cdots$$

$$\delta^*(q, wa) =$$

$$\{r \in Q | \exists s, s' \in Q :$$

$$s \in \delta^*(q, w) \wedge s' \in \delta(s, a) \wedge r \in \delta^*(s', \lambda)\}$$

Intuitief: $r \in \delta^*(q, w)$ betekent dat er een pad in de transitiegraaf is van q naar r, gelabeld met w, waarbij onderweg ook lege stappen toegestaan zijn

---

32

De definitie van $L(M)$ voor een nfa M is ingewikkelder dan voor een dfa M

Maar de definitie van een nfa geeft veel meer ruimte: we hebben niet de eis dat er voor elke q en elke a precies één uitgaande pijl vanuit q is, maar alles mag

Zo kunnen we makkelijk een nfa M maken met

$$L(M) = L(M_1) \cup L(M_2)$$

voor gegeven dfa's M_1, M_2, een dfa kan wel maar is veel moeilijker

Is het formalisme van nfa's krachtiger, in de zin dat er talen zijn die door een nfa gegenereerd worden, maar niet door een dfa?

NEE

33

Stelling

Een taal kan door een nfa worden gegenereerd dan en slechts dan als hij door een dfa kan worden gegenereerd

Bewijsschets:

Laat $M = (Q, \Sigma, \delta, q_0, F)$ een dfa zijn

Dan is $M' = (Q, \Sigma, \delta', q_0, F)$ met

$$\delta'(q, a) = \{\delta(q, a)\}$$

$$\delta'(q, \lambda) = \emptyset$$

een nfa met $L(M') = L(M)$

Laat omgekeerd $M = (Q, \Sigma, \delta, q_0, F)$ een nfa zijn

We construeren een dfa $M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)$ met $L(M_D) = L(M)$

34

$$Q_D = 2^Q$$

$$\delta_D(X, a) = \bigcup_{q \in X} \delta^*(q, a)$$

$$q_{0D} = \{q_0\}$$

$$F_D = \begin{cases} \{X \subseteq Q | X \cap F \neq \emptyset\} \\ \quad \text{als } \lambda \notin L(M) \\ \{X \subseteq Q | X \cap F \neq \emptyset\} \cup \{q_0\} \\ \quad \text{als } \lambda \in L(M) \end{cases}$$

Voor elke $w \in \Sigma^+$ geldt

$$a \in \delta^*(q_0, w) \iff a \in \delta_D^*(\{q_0\}, w)$$

Hieruit volgt $L(M_D) = L(M)$

Einde bewijsschets

35

In het boek staat dezelfde constructie waarbij alleen de vanuit $\{q_0\}$ bereikbare deelverzamelingen van Q in Q_D terecht komen

In het boek staat ook een constructie om de dfa zo klein mogelijk maken door het samenknijpen van ononderscheidbare knopen

Het resultaat is dan echt 'zo klein mogelijk':

er bestaat geen dfa met minder knopen die precies dezelfde taal genereert

---

36

We hebben nu twee equivalente karakteriseringen van reguliere talen gezien:

- een reguliere taal is een taal L met $L = L(M)$ voor zekere dfa M
- een reguliere taal is een taal L met $L = L(M)$ voor zekere nfa M

We zullen er nog diverse andere zien, die allemaal equivalent zullen blijken te zijn:

- met reguliere expressies
- met reguliere grammatica's

37

Reguliere expressies

We hebben al manieren gezien om nieuwe talen uit bestaande talen te maken:

$L_1 \cup L_2$

$L_1 L_2 = \{vw | v \in L_1 \wedge w \in L_2\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$

Met reguliere expressies beschrijf je talen die je kunt opbouwen door met losse letters te beginnen en deze constructies te gebruiken

---

38

Definitie:

Een reguliere expressie over een alfabet Σ is een uitdrukking die verkregen wordt door een eindig aantal keren de volgende spelregels toe te passen:

- $\emptyset$ is een reguliere expressie
- λ is een reguliere expressie
- a is een reguliere expressie voor elke $a \in \Sigma$
- r^* is een reguliere expressie voor elke reguliere expressie r
- $(r_1 + r_2)$ is een reguliere expressie voor alle reguliere expressies r_1 en r_2
- $(r_1 \cdot r_2)$ is een reguliere expressie voor alle reguliere expressies r_1 en r_2

39

Voorbeeld

$$((\lambda + b^*) \cdot (c^* \cdot \emptyset))$$

is een reguliere expressie over $\Sigma = \{a, b, c\}$

Net zoals met uitdrukkingen met optelling en vermenigvuldiging laten we soms overbodige haakjes weg

We geven $\cdot$ een hogere prioriteit dan $+$, dat wil zeggen:

$$r_1 \cdot r_2 + r_3 = ((r_1 \cdot r_2) + r_3)$$

$$r_1 \cdot r_2 + r_3 \neq (r_1 \cdot (r_2 + r_3))$$

Op dit verschijnsel van ambiguiteit komen we later uitgebreid terug

Soms laten we ook het symbool $\cdot$ weg

40

Elke reguliere expressie r definieert een taal $L(r)$ volgens de volgende regels

- $L(\emptyset) = \emptyset$
- $L(\lambda) = \{\lambda\}$
- $L(a) = \{a\}$ voor elke $a \in \Sigma$
- $L(r^*) = (L(r))^*$
  voor elke reguliere expressie r
- $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  voor alle reguliere expressies r_1 en r_2
- $L(r_1 \cdot r_2) = L(r_1)L(r_2)$
  voor alle reguliere expressies r_1 en r_2

41

Voorbeeld:

$$\begin{aligned} L((a+b) \cdot c^*) &= L(a+b)L(c^*) \\ &= (L(a) \cup L(b))L(c^*) \\ &= (\{a\} \cup \{b\})L(c^*) \\ &= \{a, b\}L(c^*) \\ &= \{a, b\}\{c\}^* \end{aligned}$$

$$= \{a, ac, acc, accc, \ldots, b, bc, bcc, bccc, \ldots\}$$

Voorbeeld:

Vind een reguliere expressie r over $\{a, b\}$ waarvoor $L(r)$ precies alle strings beschrijft met minstens een paar opeenvolgende a's

In zo'n string zit ergens het patroon aa; daarvoor en daarna mag van alles staan

'Alles': $\Sigma^* = \{a, b\}^* = L((a + b)^*)$

Dus:

$$r = (a + b)^* aa (a + b)^*$$

42

Zo'n expressie heet niet voor niets regulier:

er geldt dat een taal L regulier is dan en slechts dan als er een reguliere expressie r is met

$$L(r) = L$$

$\cdots$ oftewel $\cdots$

Stelling:

Voor elke reguliere expressie r is er een nfa M met $L(M) = L(r)$

en

Voor elke nfa M is er een reguliere expressie r met $L(r) = L(M)$

43

We laten eerst zien:

Voor elke reguliere expressie r is er een nfa M met $L(M) = L(r)$

Daartoe construeren we voor elk van de bouwstenen van een reguliere expressie een bijbehorende nfa met precies één eindtoestand en precies de juiste bijbehorende taal

---

44

Omgekeerd laten we zien:

Voor elke nfa M is er een reguliere expressie r met $L(r) = L(M)$

Door eventueel een nieuwe starttoestand en een pijl gelabeld met λ toe te voegen kunnen we ervoor zorgen dat $q_0 \notin F$, zonder $L(M)$ te veranderen

Door eventueel een nieuwe toestand toe te voegen en voor elke eindtoestand een pijl gelabeld met λ naar de nieuwe toestand, en alleen de nieuwe toestand tot eindtoestand te verklaren kunnen we ervoor zorgen dat er precies één eindtoestand is, zonder $L(M)$ te veranderen

Nu gaan we beschrijven hoe we stapsgewijs zo'n nfa M met $q_0 \notin F$ en $\#F = 1$ omzetten naar een reguliere expressie, zonder de bijbehorende taal te veranderen

---

45

Daartoe maken we gebruik van

gegeneraliseerde transitiegrafen, oftewel transitiegrafen waarbij de pijlen niet alleen met λ en $a \in \Sigma$ gelabeld mogen zijn, maar met willekeurige reguliere expressies over Σ

De taal gegenereerd door zo'n gegeneraliseerde transitiegraaf is de verzameling strings die je kunt schrijven als

$$v_1 v_2 \cdots v_n$$

met $v_i \in L(r_i)$ voor $i = 1, \ldots, n$ waarvoor er een pad is beginnend in q_0 en eindigend in een eindtoestand, en waarvan de pijlen achtereenvolgens gelabeld zijn met $r_1, r_2, \ldots, r_n$

Als alle pijlen gelabeld zijn met λ of $a \in \Sigma$ is dit de gewone transitiegraaf bij een nfa, en is dit precies de taal die door deze nfa geaccepteerd wordt

---

46

Bewijsidee:

- Begin met de transitiegraaf van de nfa met $q_0 \notin F$ en $\#F = 1$

- Zolang er behalve q_0 en de eindknoop nog andere knopen zijn, verwijder dan zo'n andere knoop, waarbij de graaf een gegeneraliseerde transitiegraaf blijft en de bijbehorende taal niet verandert

  (steeds wordt de graaf kleiner, maar worden de reguliere expressies in de labels ingewikkelder)

- Dit gaat door tot er alleen nog maar q_0 en de eindknoop bestaan

  Dan is de gegeneraliseerde transitiegraaf van de vorm van figuur 3.10 en is de uiteindelijke reguliere expressie:

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

47

Door + te gebruiken is er voor te zorgen dat er nooit meer dan een pijl van een knoop q_1 naar een knoop q_2 loopt

De lastigste stap is het verwijderen van een knoop:

Als je q verwijdert voeg dan voor elke pijl van q_1 naar q gelabeld met r_1 en elke pijl van q naar q_2 gelabeld met r_2 een nieuwe pijl van q_1 naar q_2 toe gelabeld met

$$r_1 r_2$$

als er geen pijl is van q naar q, en met

$$r_1 r^* r_2$$

als de pijl van q naar q is gelabeld met r

48

Reguliere grammatica's

Een grammatica heet **rechts-lineair** als er alleen maar producties zijn van de vorm

$$A \to xB$$

en

$$A \to x$$

voor $A, B \in V$ en $x \in T^*$

Een grammatica heet **links-lineair** als er alleen maar producties zijn van de vorm

$$A \to Bx$$

en

$$A \to x$$

voor $A, B \in V$ en $x \in T^*$

49

Een grammatica heet **regulier** als hij links-lineair of rechts-lineair is

Let wel: $\lambda \in T^*$ mag wel als rechterlid van een productie in een reguliere grammatica voorkomen, maar geen combinatie van regels van de vorm $A \to Bx$ en $C \to yD$ met $x \neq \lambda \neq y$

Een grammatica heet **lineair** als er alleen maar producties zijn van de vorm

$$A \to xBy$$

en

$$A \to x$$

voor $A, B \in V$ en $x, y \in T^*$

Niet elke lineaire grammatica is dus regulier

50

Stelling: Als G een rechts-lineaire grammatica is, dan is $L(G)$ een reguliere taal

Bewijsschets:

We gaan een nfa M maken met $L(G) = L(M)$

Maak voor elke $A \in V$ een toestand en kies

$$q_0 = S$$

Voeg een eindtoestand A_f toe, en kies

$$F = \{A_f\}$$

Voeg voor elke productie van de vorm

$$A \to a_1 a_2 \cdots a_n B$$

$n-1$ nieuwe toestanden toe en pijlen

$$A \xrightarrow{a_1} \cdot \xrightarrow{a_2} \cdot \ \cdots \ \cdot \xrightarrow{a_n} B$$

51

Voeg voor elke productie van de vorm

$$A \to a_1 a_2 \cdots a_n$$

$n-1$ nieuwe toestanden toe en pijlen

$$A \xrightarrow{a_1} \cdot \xrightarrow{a_2} \cdot \ \cdots \ \cdot \xrightarrow{a_n} A_f$$

Deze constructie is zo gemaakt dat er een afleiding van S naar v bestaat precies dan als er een pad van $q_0 = S$ naar A_f in deze transitiegraaf is met als labels precies achtereenvolgens de elementen van v

Einde bewijsschets

Omgekeerd gaan we nu laten zien dat elke reguliere taal gegenereerd wordt door een rechts-lineaire grammatica

---

52

Bewijsschets: We gaan bij een dfa

$$M = (Q, \Sigma, \delta, q_0, F)$$

een rechts-lineaire grammatica $G = (V, \Sigma, S, P)$ maken met

$$L(G) = L(M)$$

Kies $V = Q$ en $S = q_0$

Maak voor elke transitie $\delta(q, a) = r$ een productie

$$q \to ar$$

en voor elke eindtoestand $q \in F$ een productie

$$q \to \lambda$$

Deze constructie is zo gemaakt dat er een afleiding van S naar v bestaat precies dan als er een pad van $q_0 = S$ naar een element van F in deze transitiegraaf is met als labels precies achtereenvolgens de elementen van v

Einde bewijsschets

53

Hoe zit het nou met links-lineaire grammatica's?

Precies zo

Er geldt dat L regulier is dan en slechts dan als L^R regulier is

(draai in nfa met één eindtoestand alle pijlen om en verwissel start en eind)

Verder geldt dat $L(\widehat{G}) = (L(G))^R$ waarbij $\widehat{G}$ uit G wordt gemaakt door alle rechterkanten van de producties achterstevoren te zetten

Dus:

Als G links-lineair is

dan is $\widehat{G}$ rechts-lineair

dan is $(L(G))^R = L(\widehat{G})$ regulier

dan is $(L(G))$ regulier

54

Omgekeerd:

Als L regulier is

dan is L^R regulier

dan is er een rechts-lineaire G met $L(G) = L^R$

dan is $\widehat{G}$ links-lineair en

$$L(\widehat{G}) = (L(G))^R = (L^R)^R = L$$

Conclusie:

Een taal L is regulier dan en slechts dan als $L = L(G)$ voor een links-lineaire grammatica G

55

De volgende beweringen zijn dus allemaal equivalent

- L is regulier
- er is een dfa M met $L(M) = L$
- er is een nfa M met $L(M) = L$
- er is een reguliere expressie r met $L(r) = L$
- er is een links-lineaire grammatica G met $L(G) = L$
- er is een rechts-lineaire grammatica G met $L(G) = L$
- er is een reguliere grammatica G met $L(G) = L$

56

Stelling:

De klasse van reguliere talen is gesloten onder vereniging, doorsnede, concatenatie, complement, ster afsluiting en omkering

Met andere woorden:

als L_1 en L_2 reguliere talen zijn, dan zijn ook

- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $L_1 L_2$
- $\overline{L_1}$
- L_1^* en
- L_1^R

reguliere talen

57

Bewijs:

Kies reguliere expressies r_1 en r_2 met

$$L(r_1) = L_1 \quad \text{en} \quad L(r_2) = L_2$$

$L_1 \cup L_2 = L(r_1 + r_2)$ is regulier want $r_1 + r_2$ is een reguliere expressie

$L_1 L_2 = L(r_1 \cdot r_2)$ is regulier want $r_1 r_2$ is een reguliere expressie

$L_1^* = L(r_1^*)$ is regulier want r_1^* is een reguliere expressie

$\overline{L_1} = L(Q, \Sigma, \delta, q_0, Q - F)$ voor een dfa $M = (Q, \Sigma, \delta, q_0, F)$ met $L(M) = L_1$, dus $\overline{L_1}$ is regulier

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ en dus regulier volgens bovenstaande

---

58

Bewijs met inductie dat $L_1^R = L(rev(r_1))$ waarin rev inductief gedefinieerd is door

- $rev(\emptyset) = \emptyset$
- $rev(\lambda) = \lambda$
- $rev(a) = a$ voor $a \in \Sigma$
- $rev(r_1 + r_2) = (rev(r_1) + rev(r_2))$
- $rev(r_1 \cdot r_2) = (rev(r_2) \cdot rev(r_1))$
- $rev(r_1^*) = (rev(r_1))^*$

Dus L_1^R is regulier

Einde bewijs

59

Er zijn nog meer constructies waaronder de klasse van reguliere talen gesloten is:

- homomorf beeld: als $h : \Sigma \to \Gamma^*$ definieer dan

  $$h(a_1 a_2 \cdots a_n) = h(a_1)h(a_2)\cdots h(a_n)$$

  Als L regulier is dan is ook

  $$h(L) = \{h(w) | w \in L\}$$

  regulier

- rechter quotiënt: als L_1 en L_2 regulier zijn dan is ook

  $$L_1/L_2 = \{x | \exists y \in L_2 : xy \in L_1\}$$

  regulier

Het lijkt wel alsof alles regulier is $\cdots$

$\cdots$ maar dat is niet zo

60

Bewering:

$L = \{a^n b^n | n \geq 0\}$ is niet regulier

Bewijs:

Stel dat $M = (Q, \Sigma, \delta, q_0, F)$ een dfa is met $L(M) = L$

Omdat Q eindig is zijn de toestanden $\delta^*(q_0, a^i)$ voor $i \geq 0$ niet allemaal verschillend

Dus zijn er i en j met $i > j \geq 0$ met

$$\delta^*(q_0, a^i) = \delta^*(q_0, a^j)$$

Dan geldt ook $\delta^*(q_0, a^i b^j) = \delta^*(q_0, a^j b^j)$

Vanwege $a^i b^j \notin L$ en $a^j b^j \in L$ geldt

$$\delta^*(q_0, a^i b^j) \notin F \quad \text{en} \quad \delta^*(q_0, a^j b^j) \in F,$$

tegenspraak, dus L is niet regulier

Einde bewijs

61

We gaan nu het idee van dit bewijs generaliseren tot de zgn pompstelling die een speciale eigenschap van reguliere talen beschrijft

Als we kunnen laten zien dat een taal die speciale eigenschap niet heeft is daarmee bewezen dat de taal niet regulier is

Pompstelling:

Voor elke reguliere taal L is er een getal m zodanig dat elke $w \in L$ met $|w| \geq m$ geschreven kan worden als

$$w = xyz$$

waarvoor $|xy| \leq m$ en $|y| \geq 1$ en

$$xy^i z \in L$$

voor elke $i \geq 0$

(hierin is y een substring die wordt 'opgepompt')

62

Bewijs:

Kies een dfa $M = (Q, \Sigma, \delta, q_0, F)$ met

$$L(M) = L$$

Kies $m = \#Q$

Kies $w = a_1 a_2 \cdots a_m \cdots \in L$ met $|w| \geq m$ willekeurig

Schrijf w_i voor de prefix $a_1 a_2 \cdots a_i$ van w van lengte i

Omdat er in totaal m toestanden zijn, zijn de $m + 1$ toestanden

$$\delta^*(q_0, w_i)$$

voor $i = 0, 1, \ldots, m$ niet allemaal verschillend

Dus zijn er j, k met $0 \leq j < k \leq m$ met

$$\delta^*(q_0, w_j) = \delta^*(q_0, w_k)$$

63

Kies $x = w_j$ en y met $xy = w_k$

Schrijf $q = \delta^*(q_0, x) = \delta^*(q_0, w_j) = \delta^*(q_0, w_k)$

Er geldt

$$\begin{aligned} \delta^*(q, y) &= \delta^*(\delta^*(q_0, w_j), y) \\ &= \delta^*(q_0, w_j y) \\ &= \delta^*(q_0, xy) \\ &= \delta^*(q_0, w_k) \\ &= q \end{aligned}$$

Dus geldt $\delta^*(q, y^i) = q$ voor elke $i \geq 0$

Vanwege $w = xyz \in L$ geldt
$\delta^*(q_0, xyz) = \delta^*(q, yz) = \delta^*(q, z) \in F$

Voor elke $i \geq 0$ geldt
$\delta^*(q_0, xy^iz) = \delta^*(q, y^iz) = \delta^*(q, z) \in F$

dus $xy^iz \in L$

Einde bewijs

64

De pompstelling kan gebruikt worden om te bewijzen dat een bepaalde taal niet regulier is

Voorbeeld:

Stel dat $L = \{a^n b^n | n \geq 0\}$ regulier is

Dan is er volgens de pompstelling een m waarvoor $\cdots$

$w = a^m b^m \in L$ is dan te schrijven als $w = xyz$

Vanwege $|xy| \leq m$ geldt $x = a^j$ en $y = a^k$ voor zekere j, k met $k \geq 1$

Volgens de pompstelling geldt nu dat

$$xyyz = a^{m+k} b^m \in L,$$

tegenspraak

Dus L is niet regulier

65

Voorbeeld: palindromen

Stel dat $L = \{w \in \{a, b\}^* | w = w^R\}$ regulier is

Dan is er volgens de pompstelling een m waarvoor $\cdots$

$w = a^m b a^m \in L$ is dan te schrijven als

$$w = xyz$$

Vanwege $|xy| \leq m$ geldt ook nu $x = a^j$ en $y = a^k$ voor zekere j, k met $k \geq 1$

Volgens de pompstelling geldt nu dat

$$xyyz = a^{m+k} b a^m \in L,$$

tegenspraak

Dus L is niet regulier

66

Voorbeeld:

Stel dat

$$L = \{a, a^2, a^4, a^8, \ldots\} = \{a^{2^n} | n \geq 0\}$$

regulier is

Dan is er volgens de pompstelling een m waarvoor $\cdots$

Kies k met $2^k > m$

$w = a^{2^k} \in L$ is dan te schrijven als

$$w = xyz$$

waarvoor $y = a^j$ met $1 \leq j \leq m < 2^k$

Volgens de pompstelling geldt nu dat

$$xyyz = a^{2^k + j} \in L$$

Er geldt $2^k < 2^k + j < 2^k + 2^k = 2^{k+1}$

Tegenspraak, dus L is niet regulier

67

Gegeven een taal L, is L regulier?

JA

Te bewijzen als volgt:

- vind dfa, nfa, reguliere expressie of reguliere grammatica die L beschrijft, of
- pas geslotenheid toe op talen waarvan je al weet dat ze regulier zijn

NEE

Te bewijzen als volgt:

- pas pompstelling toe: kies geschikte $w \in L$ afhankelijk van onbekende m en laat zien dat de opgepompte variant van w niet in L zit., of
- pas geslotenheid toe op talen waarvan je al weet dat ze niet regulier zijn

---

68

De methoden voor JA en NEE zijn zeer verschillend, en in alle gevallen moet je zelf iets kiezen

Alternatieve methode zonder deze bezwaren: analyseer

$$\{ L_w \mid w \in \Sigma^* \}$$

waarbij L_w gedefinieerd is door

$$L_w = \{ v \in \Sigma^* \mid wv \in L \}$$

Stelling

Een taal $L \subseteq \Sigma^*$ is regulier dan en slechts dan als de verzameling talen

$$\{ L_w \mid w \in \Sigma^* \}$$

eindig is

69

Voorbeeld

$L = \{ a^n b^n \mid n \geq 0 \}$

Hier geldt

$$L_{a^k} = \{ a^n b^{n+k} \mid n \geq 0 \}$$

Dit is voor elke k een andere taal, dit geeft dus al oneindig veel verschillende talen L_w waarbij w van de vorm a^k is

Als we alle strings w beschouwen zijn er dus zeker oneindig veel verschillende talen L_w

L is dus niet regulier

70

Voorbeeld

$L = \{ a^n b^m \mid n \geq 0,\ m \geq 0 \}$

Hier geldt

$L_{a^k} = L$ voor elke $k \geq 0$

$L_{a^k b^m} = \{ b^n \mid n \geq 0 \}$

voor elke $k \geq 0$ en $m \geq 1$, en

$L_w = \emptyset$

voor elke andere $w \in \{a, b\}^*$

Er komen dus slechts drie verschillende talen L_w voor

L is dus regulier

71

Contextvrije talen

Een grammatica $G = (V, T, S, P)$ heet **contextvrij** als elke productie in P van de vorm

$$A \to x$$

is met $A \in V$ en $x \in (V \cup T)^*$

Een taal L heet **contextvrij** als er een contextvrije grammatica G bestaat met $L = L(G)$

Elke reguliere taal is contextvrij (want elke reguliere grammatica is contextvrij), maar niet omgekeerd

$$L(G) = \{a^n b^n | n \geq 0\}$$

voor G:

$$S \to \lambda, \quad S \to aSb$$

72

Bij lineaire grammatica's is er in de tussenstappen van een afleiding van S naar een zin in de taal altijd maar een variabele waarmee je verder kunt:

$$S \Rightarrow bS \Rightarrow bbS \Rightarrow bbaA \Rightarrow bbacA \Rightarrow bbaca$$

in

$$S \to aA,\ S \to bS,\ A \to cA,\ A \to a$$

Bij rechts-lineaire grammatica's staan de variabelen helemaal rechts

Bij links-lineaire grammatica's staan de variabelen helemaal links

Bij lineaire grammatica's kunnen de variabelen ook tussenin staan:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

in

$$S \to aSb,\ S \to \lambda$$

73

Bij contextvrije grammatica's kunnen er meerdere variabelen staan en is er keuze waarmee je verder gaat

Zo heb je in

$$S \to SaS,\ S \to b$$

een **leftmost** afleiding

$$S \to SaS \to baS \to bab$$

en een **rightmost** afleiding

$$S \to SaS \to Sab \to bab$$

74

Voor de uiteindelijke zin in de taal is die volgorde niet van belang, het gaat om de

afleidingsboom (derivation tree):

- de wortel is gelabeld met S
- elke knoop gelabeld met een variabele A heeft kinderen achtereenvolgens gelabeld met $a_1, a_2, \ldots, a_n$ voor zekere productie

  $$A \to a_1, a_2, \ldots, a_n$$

- elke knoop gelabeld met een terminal of λ is een blad, dwz heeft geen kinderen

---

75

Alle labels van de bladeren van links naar rechts achter elkaar gezet (sla λ over) vormen de opgebouwde zin in de taal: de **opbrengst** (yield) van de afleidingsboom

Als we toestaan dat een variabele geen kinderen heeft spreken we van een

partiële afleidingsboom

Zo is de boom die alleen bestaat uit een wortel gelabeld met S een partiële afleidingsboom

---

76

Het eerste wat een compiler doet met een programma is het maken van een afleidingsboom die het ingelezen programma als opbrengst heeft

Dit proces heet

ontleden (parsing):

gegeven een grammatica G en een string w, stel vast of $w \in L(G)$, en zo ja, vind een bijbehorende afleidingsboom

Zo'n afleidingsboom heet dan ook wel

ontleedboom (parse tree)

Het proces van ontleden willen we zo efficient mogelijk kunnen doen

---

77

Bewezen kan worden dat dit altijd in $O(|w|^3)$ kan, oftewel:

> voor elke contextvrije grammatica G is er een algoritme en een getal C zodanig dat voor elke string w het algoritme in ten hoogste $C|w|^3$ stappen kan vaststellen of $w \in L(G)$, en zo ja, een bijbehorende ontleedboom vindt

In veel gevallen kan het veel efficienter

Een contextvrije grammatica heet **simpel** als elke productie van de vorm

$$A \to ax$$

met $a \in T$, en voor elke $A \in V$ en elke $a \in T$ is er maar hoogstens één zo'n productie

Bij een simpele grammatica is er in het van links naar rechts zoeken naar een passende ontleedboom nooit keuzemogelijkheid

---

78

Dit geeft een heel efficient **lineair** ontleedalgoritme, dat wil zeggen dat het aantal benodigde stappen slechts $O(|w|)$ is

Of je vindt in $O(|w|)$ stappen voor w de enig mogelijke ontleedboom, of je weet zeker dat $w \notin L(G)$

Voor algemenere contextvrije grammatica's kan het voorkomen dat een string meerdere ontleedbomen heeft, zo'n grammatica heet **ambigu**

Preciezer gezegd:

Een contextvrije grammatica G heet **ambigu** als er een $w \in L(G)$ bestaat met minstens twee verschillende ontleedbomen

Zo is $S \to SS, S \to a$ ambigu

79

Ambiguiteit is een onwenselijke eigenschap: vaak wordt de grammatica gebruikt om betekenis te definiëren, en krijgt een zin met verschillende ontleedbomen dubbelzinnige betekenis

Voorbeeld: if—then en if—then—else

Vaak kan een taal zowel met een ambigue als met een niet-ambigue grammatica worden beschreven; we geven dan de voorkeur aan een niet-ambigue

Voorbeeld:

$$S \to SS, S \to a$$

is ambigu, maar

$$S \to aS, S \to a$$

niet, terwijl ze beiden $\{a^n | n > 0\}$ genereren

80

Voorbeeld:

$$\begin{array}{rcl} \langle expr \rangle & ::= & \langle factor \rangle \mid \\ & & \langle expr \rangle \ * \ \langle expr \rangle \mid \\ & & \langle expr \rangle \ + \ \langle expr \rangle \mid \\ & & \langle expr \rangle \ - \ \langle expr \rangle \\ \langle factor \rangle & ::= & \cdots \end{array}$$

is ambigu, maar

$$\begin{array}{rcl} \langle expr \rangle & ::= & \langle term \rangle \mid \\ & & \langle expr \rangle \ + \ \langle term \rangle \mid \\ & & \langle expr \rangle \ - \ \langle term \rangle \\ \langle term \rangle & ::= & \langle factor \rangle \mid \\ & & \langle term \rangle \ * \ \langle factor \rangle \\ \langle factor \rangle & ::= & \cdots \end{array}$$

niet

81

Er zijn echter ook **inherent ambigue** talen: zo'n taal L is wel contextvrij, maar elke contextvrije grammatica G met $L = L(G)$ is ambigu

Voorbeeld:

$$\{\ a^n b^m c^k \mid n = m\ \vee\ m = k\ \}$$

Een grammatica hiervoor is

$$\begin{array}{lcl} S & \rightarrow & A \mid B \\ A & \rightarrow & Ac \mid C \\ C & \rightarrow & aCb \mid \lambda \\ B & \rightarrow & aB \mid D \\ D & \rightarrow & bDc \mid \lambda \end{array}$$

A: $\{a^n b^n c^k\}$

B: $\{a^n b^k c^k\}$

82

Opschonen van grammatica's

Een taal kan vaak op veel manieren met een contextvrije grammatica worden beschreven

We gaan nu op zoek naar manieren om bij een contextvrije grammatica een 'eenvoudiger' contextvrije grammatica te vinden die dezelfde taal genereert

Wat betekent 'eenvoudiger'?

- alle variabelen doen mee
- alleen producties $A \rightarrow \lambda$ toegestaan als $\lambda \in L$
- geen producties $A \rightarrow B$
- .....

83

Algemene substitutieregel

substitutie = vervangen door iets anders

Als $B \rightarrow y_1 | y_2 | \cdots | y_n$ alle producties zijn voor de variabele B, dan mag je een productie

$$A \rightarrow uBv$$

vervangen door de n producties

$$A\ \rightarrow\ uy_1v \mid uy_2v \mid\ \cdots\ \mid uy_nv$$

waarbij de gegenereerde taal dezelfde blijft

Voorbeeld:

$$S\ \rightarrow\ 0A \mid \lambda,\ \ A\ \rightarrow\ S1$$

kan zo worden vervangen door

$$S\ \rightarrow\ 0S1 \mid \lambda,\ \ A\ \rightarrow\ S1$$

en vervolgens worden vereenvoudigd tot

$$S\ \rightarrow\ 0S1 \mid \lambda$$

84

Toepassen van deze substitutieregel is lang niet altijd een vereenvoudiging

Een variabele A heet **zinloos** als hij nooit meedoet in een afleiding van S naar een woord in de taal, preciezer gezegd:

er bestaat geen afleiding van de vorm

$$S\ \Rightarrow^*\ uAv\ \Rightarrow^*\ w \quad \text{met}\ \ w \in T^*$$

Een productie heet zinloos als er een zinloze variabele in voorkomt

Per definitie verandert het verwijderen van zinloze producties uit een grammatica niet de gegenereerde taal

Het is mogelijk systematisch uit te zoeken welke variabelen zinloos zijn

---

85

Verwijderen van λ-producties

Een productie $A \to \lambda$ heet een **λ-productie**

Een variabele A heet **nullable** als $A \Rightarrow^* \lambda$

Stelling

Voor elke contextvrije taal L met $\lambda \notin L$ is er een contextvrije grammatica G zonder λ-producties met $L(G) = L$

86

Bewijsschets:

Per definitie is er een contextvrije grammatica G met $L(G) = L$; we hoeven alleen nog eventuele λ-producties uit G weg te werken

Bepaal eerst alle nullable variabelen in G met de volgende regels:

- als $A \to \lambda$ dan is A nullable
- als $A \to B_1 B_2 \cdots B_n$ voor $B_1, B_2, \ldots, B_n$ nullable, dan is ook A nullable

Voeg nu voor elke manier waarop je een of meer nullable variabelen in een rechterlid van een productie kunt verwijderen de bijbehorende nieuwe productie toe

Verwijder tenslotte alle λ-producties

Ga na dat $L(G)$ hiermee niet veranderd is

Einde bewijsschets

87

Een productie $A \to B$ met $A, B \in V$ heet een **eenheid-productie** (unit-production)

Ook eenheid-producties kunnen worden verwijderd waarbij met het toevoegen van een stel vervangende producties de taal $L(G)$ niet verandert

Samenvattend:

Voor elke contextvrije taal L met $\lambda \notin L$ is er een contextvrije grammatica G zonder

- zinloze producties
- λ-producties
- eenheid-producties

met $L(G) = L$

88

Er zijn nog meer manieren om contextvrije grammatica's op te schonen:

- **Chomsky normaalvorm**: er zijn alleen producties van de vorm

$$A \to BC \quad \text{en} \quad A \to a$$

- **Greibach normaalvorm**: er zijn alleen producties van de vorm

$$A \to ax \quad \text{met } x \in V^*$$

Voor beide soorten geldt:

Voor elke contextvrije taal L met $\lambda \notin L$ is er een contextvrije grammatica G in $\cdots$ normaalvorm met $L(G) = L$

89

Voor reguliere grammatica's hebben we een automaatbegrip gezien, nl. dfa, en ook nfa, zodanig dat

> Een taal wordt gegenereeerd door een reguliere grammatica precies dan als hij geaccepteerd wordt door zo'n automaat

Bestaat zoiets nou ook voor contextvrije grammatica's?

Preciezer gezegd: is er een soort automaat waarvoor geldt:

Een taal is contextvrij precies dan als hij geaccepteerd wordt door een $\cdots$-automaat

Ja

nl. een **pushdown automaat**

90

Zo'n automaatbegrip moet dan wel krachtiger zijn dan dfa's en nfa's want ook niet-reguliere contextvrije talen zoals

$$\{a^n b^n \mid n \geq 0\}$$

moeten ermee geaccepteerd kunnen worden

De zwakte van dfa's en nfa's is dat de automaat bij het inlezen van een string geen andere vorm van geheugen heeft dan de toestand waarin hij zich bevindt, en daar zijn er maar eindig veel van

In een pushdown automaat is het mogelijk intern een hoeveelheid nog te verwerken informatie op te stapelen

Voor elk stukje informatie op die stapel (stack) zijn er maar eindig veel mogelijkheden, maar er is geen beperking op de grootte van de stapel

---

91

Bij het inlezen van a^n kan dan n keer de informatie dat er een a gelezen is op die stapel gezet worden; bij het vervolgens inlezen van elke b wordt zo'n stukje informatie weer verwijderd

Op een dergelijke manier kan de pushdown automaat herkennen of een string van de vorm $a^n b^n$ is, wat bij dfa's en nfa's niet kan

Ter herinnering:

In een nfa $M = (Q, \Sigma, \delta, q_0, F)$ is

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$$

en betekent $r \in \delta(q, a)$ dat je vanuit toestand q bij input element a naar toestand r kunt

We gaan dit idee nu uitbreiden zodat de interne stapel ook meedoet

---

92

Behalve het **input alfabet** Σ is er nu ook een **stapel alfabet** Γ

De stapel zelf is een rij elementen van Γ, dus zelf een element van Γ^*, waarbij we afspreken dat het bovenste element van de stapel vooraan in de string staat

We willen nu toestaan dat $\delta(\cdots)$ ook afhangt van het bovenste element van de stapel

Zo wordt het domein van δ uitgebreid met Γ (het lezen van het bovenste element van de stapel):

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \cdots$$

93

Verder willen we iets op de stapel kunnen zetten en er iets af kunnen halen

Dit doen we door het bovenste element er altijd af te halen en vervolgens een string van elementen er aan toe te voegen

(als je de stapel niet wilt veranderen voeg je het weggehaalde element gelijk weer toe)

Zo wordt het bereik van δ uitgebreid met Γ^*

(voor de string die je aan de stapel toe wilt voegen)

Dus:

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{Q \times \Gamma^*}$$

Nu betekent $(r, v) \in \delta(q, a, b)$ dat je vanuit toestand q bij input element a en het element b bovenaan de stapel naar toestand r kunt

waarbij je b van de stapel verwijdert en de string v aan de stapel toevoegt

---

94

Wel eisen we dat altijd $\delta(q, a, b)$ eindig is om de keuzemogelijkheid van het non-determinisme eindig te houden

In veel gevallen bestaat $\delta(q, a, b)$ uit hoogstens één element en is de automaat deterministisch

Altijd moet de stapel een bovenste element hebben; daartoe moet ook in het begin de stapel minstens één element bevatten

We spreken af dat in het begin de stapel precies één element bevat:
het **stapelstartsymbool** $z \in \Gamma$

95

Definitie

Een **nondeterministische pushdown accepter (npda)** is een zeventupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$$

waarin

- Q een eindige verzameling interne toestanden is
- Σ een eindig input alfabet is
- Γ een eindig stapel alfabet is
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{Q \times \Gamma^*}$
  de transitiefunctie is, met $\delta(q, a, b)$ eindig voor elke $q \in Q$ en $a \in \Sigma \cup \{\lambda\}$ en $b \in \Gamma$
- $q_0 \in Q$ de begintoestand is
- $z \in \Gamma$ het stapelstartsymbool is
- $F \subseteq Q$ de verzameling eindtoestanden is

96

We moeten nog de transitiestappen in een npda precies definiëren, en daarmee wat de taal $L(M)$ is gegenereerd door een npda M

Op elk moment tijdens het doorlopen van zo'n npda hebben te maken met drie dingen:

- de toestand $q \in Q$
- de nog te lezen string $w \in \Sigma^*$
- de interne stapel $u \in \Gamma^*$

Zo'n drietal $(q, w, u) \in Q \times \Sigma^* \times \Gamma^*$ heet een **instantane beschrijving**

Een transitiestap van de ene instantane beschrijving in de andere geven we aan met het symbool

$\vdash$

97

Als $(r, v) \in \delta(q, a, b)$ dan kun je van q bij input element a naar toestand r waarbij je het bovenste element b van de stapel verwijdert en de string v aan de stapel toevoegt, dus

$$(q, aw, bx) \vdash (r, w, vx)$$

We definiëren nu

$$(q, aw, bx) \vdash (r, w, vx)$$

dan en slechts dan als

$$(r, v) \in \delta(q, a, b)$$

voor $a \in \Sigma \cup \{\lambda\}$

De taal $L(M)$ is nu gedefinieerd als

$L(M) = \{w \in \Sigma^* \mid (q_0, w, z) \vdash^* (r, \lambda, u), r \in F\}$

98

oftewel de verzameling strings w waarbij je vanuit begintoestand q_0 en startstapel z in een aantal $\vdash$-stappen in een eindtoestand terecht kunt komen

Net als bij nfa's en dfa's moet de hele input w afgewerkt zijn; in deze notatie is dat de λ in (r, λ, u)

Over de toestand van de stapel u aan het eind wordt geen enkele eis gesteld

---

99

Nu gaan we een automaat voor $\{a^n b^n | n \geq 1\}$ maken

We kiezen $\Gamma = \{0, 1\}$ en $z = 0$

We maken de automaat zo dat als we vanuit q_0 een a lezen er een 1 op de stapel wordt gezet en we in q_0 blijven

Zo is in toestand q_0 de stapel een soort teller die bijhoudt hoeveel a's er gelezen zijn: als er a^n gelezen is dan is de stapel $1^n 0$

We maken een toestand q_1 waar we zijn als we $a^n b^k$ gelezen hebben voor $1 \leq k \leq n$, de stapel is dan $1^{n-k}0$

Als we dan $a^n b^n$ hebben gelezen is de stapel weer 0 en kunnen we met een lege stap naar de enige eindtoestand q_2

Dus:

$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ met

100

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{0, 1\}$
- $\delta(q_0, a, 0) = \{(q_0, 10)\}$
- $\delta(q_0, a, 1) = \{(q_0, 11)\}$
- $\delta(q_0, b, 1) = \{(q_1, \lambda)\}$
- $\delta(q_1, b, 1) = \{(q_1, \lambda)\}$
- $\delta(q_1, \lambda, 0) = \{(q_2, \lambda)\}$
- $\delta(\cdots) = \emptyset$ voor elke andere $\cdots$
- $z = 0$
- $F = \{q_2\}$

Hiervoor geldt $L(M) = \{a^n b^n | n \geq 1\}$

Zo hebben we

$$(q_0, aaabbb, 0) \vdash (q_0, aabbb, 10) \vdash$$

$$(q_0, abbb, 110) \vdash (q_0, bbb, 1110) \vdash (q_1, bb, 110) \vdash$$

$$(q_1, b, 10) \vdash (q_1, \lambda, 0) \vdash (q_2, \lambda, \lambda)$$

101

Voorbeeld

We gaan een npda maken voor

$$\{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\ \}$$

Onderweg willen we $n_a(w) - n_b(w)$ tellen en in de stapel representeren

Probleempje: dit kan zowel positief als negatief zijn

Oplossing: zorg dat je de stapel

$0^k z$ hebt als $n_a(w) - n_b(w) = k > 0$ en

$1^k z$ hebt als $n_a(w) - n_b(w) = -k < 0$

Kies $Q = \{q_0, q_1\}$ en $F = \{q_1\}$

102

- $\delta(q_0, \lambda, z) = \{(q_1, \lambda)\}$
- $\delta(q_0, a, z) = \{(q_0, 0z)\}$
- $\delta(q_0, b, z) = \{(q_0, 1z)\}$
- $\delta(q_0, a, 0) = \{(q_0, 00)\}$
- $\delta(q_0, b, 0) = \{(q_0, \lambda)\}$

- $\delta(q_0, a, 1) = \{(q_0, \lambda)\}$
- $\delta(q_0, b, 1) = \{(q_0, 11)\}$

en voor de rest weer $\delta(\cdots) = \emptyset$

Deze doet het, zo hebben we

$$(q_0, abbbaa, z) \vdash (q_0, bbbaa, 0z) \vdash$$
$$(q_0, bbaa, z) \vdash (q_0, baa, 1z) \vdash$$
$$(q_0, aa, 11z) \vdash (q_0, a, 1z) \vdash$$
$$(q_0, \lambda, z) \vdash (q_1, \lambda, \lambda)$$

dus $abbbaa \in L(M)$

103

Het verzinnen van een npda voor een gegeven taal is een soort programmeren

Als je dit systematisch wilt doen bedenk je eerst wat voor een eigenschap voor de interne stapel **invariant** is gedurende het hele proces, waarna de definitie voor de bijbehorende transitiestappen daardoor afgedwongen wordt

De automaat is dan zowel een programma als een eenvoudig soort computer; je laat in het midden of het hardware of software is

Grote analogie met 'gewone' programma's, het ontwikkelen daarvan en het redeneren daarover

---

104

Notatie:

Als we een pushdown-automaat met een transitiegraaf willen beschrijven, tekenen we een pijl van q naar r gelabeld met $a[b/v]$ voor elke

$$(r, v) \in \delta(q, a, b)$$

hetgeen betekent:

als ik in q zit, ik lees a van de input en b van de stapel, dan mag ik naar r waarbij b van de stapel wordt verwijderd en vervolgens v aan de stapel wordt toegevoegd

Soms is het handig de symbolen uit Σ zelf op de stapel te zetten

Voorbeeld

$$\{ww^R \mid w \in \{a, b\}^+\}$$

105

Idee:

- Lees eerst w waarbij je in q_0 blijft en de gelezen string op de stapel zet
- Te allen tijde mag je met een lege stap naar q_1

  (dit is non-deterministisch gedrag)
- Vanuit q_1 lees je w^R waarbij je de stapel afbreekt, alleen toegestaan als het de juiste symbolen zijn

Als je zo de stapel helemaal leeg krijgt, heb je precies ww^R gelezen, en mag je naar de eindtoestand q_2

Dus:

$$Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{a, b, z\}$$
$$F = \{q_2\}$$

106

Nodig om w op de stapel te krijgen:

$$\delta(q_0, a, z) = \{(q_0, az)\}$$
$$\delta(q_0, b, z) = \{(q_0, bz)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, a) = \{(q_0, ba)\}$$
$$\delta(q_0, a, b) = \{(q_0, ab)\}$$
$$\delta(q_0, b, b) = \{(q_0, bb)\}$$

Nodig om naar q_1 over te stappen:

$$\delta(q_0, \lambda, a) = \{(q_1, a)\}$$

$$\delta(q_0, \lambda, b) = \{(q_1, b)\}$$

Nodig om de stapel weer af te breken:

$$\delta(q_1, a, a) = \{(q_1, \lambda)\}$$

$$\delta(q_1, b, b) = \{(q_1, \lambda)\}$$

En tenslotte om de eindtoestand te bereiken:

$$\delta(q_1, \lambda, z) = \{(q_2, z)\}$$

107

Zo krijgen we bijvoorbeeld

$$\begin{aligned}
(q_0, abbbba, z) &\vdash (q_0, bbbba, az) \\
&\vdash (q_0, bbba, baz) \\
&\vdash (q_0, bba, bbaz) \\
&\vdash (q_1, bba, bbaz) \\
&\vdash (q_1, ba, baz) \\
&\vdash (q_1, a, az) \\
&\vdash (q_1, \lambda, z) \\
&\vdash (q_2, \lambda, z)
\end{aligned}$$

Wat hebben pushdown-automaten nu met contextvrije talen te maken?

Het blijkt dat je voor elke contextvrije taal een pushdown-automaat kunt maken en omgekeerd

---

108

We gaan eerst bij een grammatica in Greibach normaalvorm een pushdown-automaat maken die precies dezelfde taal beschrijft

Ter herinnering: een grammatica is in Greibach normaalvorm als er alleen producties zijn van de vorm

$$A \to ax \quad \text{met } x \in V^*$$

Als toestanden kiezen we weer

$$Q = \{q_0, q_1, q_2\}$$

met

$$F = \{q_2\}$$

Verder kiezen we

$$\Gamma = V \cup \{z\}$$

opdat we de nog uit te werken variabelen op de stapel kunnen zetten

---

109

Om vast te leggen dat een afleiding begint met S kiezen we

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$$

Dit definiëren we als de enig mogelijke stap vanuit q_0

Om vast te leggen dat we bij een lege stapel naar de eindtoestand mogen kiezen we

$$\delta(q_1, \lambda, z) = \{(q_2, z)\}$$

Dit definiëren we als de enig mogelijke stap naar de eindtoestand q_2

110

Voor de rest speelt alles zich rond q_1 af: we kiezen $\delta(q_1, a, A)$ zodanig dat er geldt dat

$$(q_1, u) \in \delta(q_1, a, A)$$

precies dan als er een productie

$$A \to au$$

is

Deze productie wordt dus afgehandeld door a van de input te lezen, A van de top van de stapel te halen en vervolgens u aan de stapel toe te voegen

Op deze manier krijgen we dat een string geaccepteerd wordt door de automaat precies dan als hij gegenereerd wordt door de grammatica

Voor elke contextvrije taal L met $\lambda \notin L$ is er een contextvrije grammatica G in Greibach normaalvorm met $L(G) = L$ en geeft deze constructie een bijbehorende pushdown-automaat

---

111

Voorbeeld:

$L = \{a^n b^n \mid n > 0\}$ laat zich beschrijven door de contextvrije grammatica

$$S \to ab \mid aSb$$

Greibach normaalvorm:

$$\begin{array}{lcl} S & \to & aB \mid aSB \\ B & \to & b \end{array}$$

$$\begin{array}{rcl} (q_0, aaabbb, z) & \vdash & (q_1, aaabbb, Sz) \\ & \vdash & (q_1, aabbb, SBz) \\ & \vdash & (q_1, abbb, SBBz) \\ & \vdash & (q_1, bbb, BBBz) \\ & \vdash & (q_1, bb, BBz) \\ & \vdash & (q_1, b, Bz) \\ & \vdash & (q_1, \lambda, z) \\ & \vdash & (q_2, \lambda, z) \end{array}$$

112

Als $\lambda \in L$ kun je deze constructie op $L \setminus \{\lambda\}$ toepassen, en definiëren $F = \{q_0, q_2\}$

Conclusie: bij elke contextvrije taal is er een pushdown-automaat die precies die taal accepteert

Omgekeerd is er ook een constructie die bij een willekeurige pushdown-automaat een contextvrije grammatica maakt

Deze is wel iets ingewikkelder: zo moeten er eerst aan een aantal extra eisen worden voldaan, en is er voor elk paar machinetoestanden en elke letter uit het stapelalfabet een variabele voor de grammatica

Conclusie:

De klasse van contextvrije talen is precies de klasse van talen die door een (non-deterministische) pushdown-automaat worden geaccepteerd

---

113

Zijn er nu ook talen die niet contextvrij zijn?

Jazeker

Zo is

$$\{a^n b^n c^n \mid n \geq 0\} =$$
$$\{\lambda, abc, aabbcc, aaabbbccc, \ldots\}$$

niet contextvrij

Hoe kun je zoiets bewijzen?

Net als bij reguliere talen: met een pompstelling

De pompstelling voor contextvrije talen geeft aan dat elke contextvrije taal een bepaalde eigenschap heeft

Als van een taal aangetoond kan worden dat hij niet aan die eigenschap voldoet is daarmee bewezen dat hij niet contextvrij is

---

114

Pompstelling

Laat L een contextvrije taal zijn

Dan is er een getal m zodanig dat elke $w \in L$ met $|w| \geq m$ geschreven kan worden als

$$w = uvxyz$$

met

$$|vxy| \leq m \quad \text{en} \quad |vy| \geq 1$$

zodanig dat voor elke $i \geq 0$ geldt

$$uv^i xy^i z \in L$$

Merk op dat dit anders is dan bij de pompstelling voor reguliere talen:

- Er wordt niet in het beginstuk van de string opgepompt maar ergens in het midden (je weet niet waar)
- Er kunnen twee stukken tegelijkertijd worden opgepompt in plaats van maar één stuk

---

115

Bewijs:

Laat G een contextvrije grammatica zijn met $L(G) = L\backslash\{\lambda\}$, zonder λ-producties en eenheid-producties, met k variabelen

De hoogte van een boom is gedefinieerd als de lengte van het langste pad van de wortel naar een blad

Kies m zo groot dat de hoogte van elke ontleedboom met minstens m bladeren meer dan k is

(bijvoorbeeld: $m = n^{k+1}$ waarbij n het grootste aantal symbolen van een rechterkant van een productie is)

Laat nu $w \in L$ met $|w| \geq m$

Dan is er een ontleedboom voor w waarvan de hoogte meer dan k is

Kies een langste pad van de wortel naar een blad in die boom

116

Omdat er slechts k variabelen zijn is er een variabele A die op de laatste $k+1$ knopen op dat pad twee (of meer) keer voorkomt

Laat x de string zijn van terminal symbolen onder de onderste van die twee A's

Laat vxy de string zijn van terminal symbolen onder de bovenste van die twee A's

Laat $w = uvxyz$ de hele string zijn

Dan geldt $|vxy| \leq m$ omdat de hoogte van de bijbehorende boom $\leq k$ is

Dan geldt $|vy| \geq 1$ omdat er geen λ-producties en eenheid-producties zijn

Er geldt $S \Rightarrow^* uAz$, $A \Rightarrow^* vAy$, $A \Rightarrow^* x$, hiermee is voor elke i een afleiding $S \Rightarrow^* uv^ixy^iz$ te maken

Einde bewijs

117

Voorbeeld

$$L = \{\, a^n b^n c^n \mid n \geq 0 \,\}$$

is niet contextvrij

Stel wel, kies dan $w = a^m b^m c^m \in L$

Vanwege $|vxy| \leq m$ geldt nu dat $n_a(vxy) = 0$ of $n_c(vxy) = 0$

Als $n_a(vxy) = 0$ dan geldt $n_a(uvvxyyz) = m$ en $n_b(uvvxyyz) + n_c(uvvxyyz) > 2m$, dus $uvvxyyz \notin L$

Als $n_c(vxy) = 0$ dan geldt $n_c(uvvxyyz) = m$ en $n_a(uvvxyyz) + n_b(uvvxyyz) > 2m$, dus $uvvxyyz \notin L$

Beide gevallen geven een tegenspraak met de pompstelling, dus is L niet contextvrij

---

118

Met precies hetzelfde bewijs is te bewijzen dat

$$\{\, w \in \{a,b,c\}^* \mid n_a(w) = n_b(w) = n_c(w) \,\}$$

niet contextvrij is

Op een vergelijkbare manier is te bewijzen dat

$$\{\, ss \mid s \in \{a,b\}^* \,\}$$

in tegenstelling tot

$$\{\, ss^R \mid s \in \{a,b\}^* \,\}$$

niet contextvrij is, kies nu $w = a^m b^m a^m b^m$ en $i = 0$

Je moet nu laten zien dat als je uit $w = a^m b^m a^m b^m$ in een stuk van lengte $\leq m$ een

aantal symbolen weghaalt, het resultaat nooit meer van de vorm ss is

---

119

Stelling

Als L contextvrij is dan zijn L^* en L^R dat ook Als L_1 en L_2 contextvrij zijn dan zijn L_1L_2 en $L_1 \cup L_2$ dat ook

Bewijs:

- L^*: voeg toe: $S \to \lambda | S_1 S$ waarbij S_1 het startsymbool van een grammatica voor L is
- L^R: draai alle rechterkanten van producties om
- L_1L_2: voeg toe: $S \to S_1S_2$ waarbij S_i het startsymbool van een grammatica voor L_i is voor $i = 1, 2$
- $L_1 \cup L_2$: voeg toe: $S \to S_1 | S_2$ waarbij S_i het startsymbool van een grammatica voor L_i is voor $i = 1, 2$

Einde bewijs

120

Echter:

Het is niet zo dat $L_1 \cap L_2$ altijd contextvrij is voor contextvrije talen L_1 en L_2

Kies bijvoorbeeld

$$L_1 = \{a^n b^n c^k \mid n \geq 0 \wedge k \geq 0 \}$$

en

$$L_2 = \{a^n b^k c^k \mid n \geq 0 \wedge k \geq 0 \}$$

Dan zijn L_1 en L_2 beide contextvrij, maar

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0 \}$$

is niet contextvrij

Het is ook niet zo dat $\overline{L}$ altijd contextvrij is voor een contextvrije taal L

Stel wel, dan zou $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ contextvrij zijn voor alle contextvrije talen L_1 en L_2 wegens voorgaande stelling, en we hebben net gezien dat dat niet zo is

---

121

Stelling

Als L_1 contextvrij is en L_2 is regulier dan is $L_1 \cap L_2$ contextvrij

Bewijs: zie pushdown-automaat-constructie in het boek

Gevolg:

Als L_1 contextvrij is en L_2 is eindig dan is $L_1 \setminus L_2$ contextvrij

Bewijs: $\overline{L_2}$ is regulier, $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$

Voorbeeld:

$\{a^n b^n \mid n \geq 0 \wedge n \neq 100\}$ is contextvrij

In het bijzonder is $L \setminus \{\lambda\}$ contextvrij voor elke contextvrije taal L

122

Elementaire vragen over talen

Standaardrepresentatie van een reguliere of contextvrije taal: beschrijving met behulp van

- dfa
- nfa
- reguliere expressie
- reguliere grammatica
- contextvrije grammatica
- pushdown-automaat

Elementaire vragen:

- gegeven L in standaardrepresentatie en $w \in \Sigma^*$, geldt $w \in L$?

- gegeven L in standaardrepresentatie, is L leeg?
- gegeven L in standaardrepresentatie, is L eindig?
- gegeven L_1 en L_2 in standaardrepresentatie, geldt $L_1 = L_2$?

123

Voor al deze vragen is er een algoritme te geven dat het antwoord bepaalt, behalve voor gelijkheidstest van contextvrije talen

Omdat we standaardrepresentaties in elkaar kunnen vertalen mogen we steeds de handigste kiezen

Zo doen we $w \in L$ voor reguliere talen met een dfa

$$L_1 = L_2 \iff (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$$

Verrassend:

Je kunt bewijzen dat er geen algoritme bestaat dat twee contextvrije grammatica's als invoer heeft en het antwoord geeft op de vraag of ze dezelfde taal genereren

---

124

Turing machines

We hebben gezien dat eindige automaten een stuk krachtiger worden door ze een onbeperkt geheugen in de vorm van een stapel te geven, resulterend in pushdown-automaten

Maar nog steeds kunnen die eenvoudige talen als $\{a^n b^n a^n\}$ en $\{ww\}$ niet accepteren

Nu kunnen we het begrip automaten in kleine stapjes uitbreiden door bijv. een tweede stapel toe te voegen, maar we gaan nu een grotere stap doen door het onderscheid tussen de invoerstring en het locale geheugen los te laten door een string te hebben waar zowel van gelezen als op geschreven kan worden

In zekere zin wordt het machineconcept hiermee juist weer een stuk eenvoudiger dan dat van pushdown automaten, terwijl het toch veel krachtiger is

---

125

Deze string, de **band** (tape) van de machine, wordt geinspecteerd door een **lees-schrijf-kop** (head) die een symbool van de band leest en hem overschrijft en vervolgens naar links of rechts schuift

Ook hier speelt weer een eindige verzameling toestanden Q mee, en een eindig alfabet: het tape alfabet Γ

De stappen worden weer beschreven door een transitiefunctie δ, deze heeft type:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

$$\delta(q, a) = (r, b, L)$$

betekent dan:

> Als de machine in toestand q is en de kop leest a van de band, dan wordt die a vervangen door b, verschuift de kop een positie naar links en is de machine daarna in toestand r

en net zo met 'links' vervangen door 'rechts' als $\delta(q, a) = (r, b, R)$

126

We willen de band als onbeperkt geheugen kunnen gebruiken, daartoe moet de string op de band ook langer kunnen worden: als de kop het rechteruiteinde van de string bereikt en dan naar rechts schuift, moet daarna op die nog onbeschreven positie van de band geschreven kunnen worden

Uit symmetrie-overwegingen willen we de string aan de linkerkant net zo kunnen uitbreiden

Dit is eenvoudig op te lossen zonder het type van δ te veranderen door de string aan beide kanten met onbeperkt veel symbolen **blank** $\Box$ aangevuld te denken

Door een symbool $\Box$ te lezen en te vervangen door een ander symbool is de string aan beide kanten uit te breiden

---

127

Samengevat: een **Turing machine** M is een zeventupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$$

met

- Q is een eindige verzameling toestanden
- Σ is het input alfabet
- Γ is het tape alfabet
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is de transitiefunctie
- $\Box \in \Gamma$ is het blank symbool, er geldt $\Sigma \subseteq \Gamma \setminus \{\Box\}$
- q_0 is de begintoestand
- $F \subseteq Q$ is de verzameling eindtoestanden

128

Een instantane beschrijving of **configuratie** van een Turing machine is nu een string vqw met $q \in Q$ en $v, w \in \Gamma^*$

Het idee hierbij is dat de string vw de band is en dat de lees-schrijf-kop staat bij het meeste linkse symbool van w

We definiëren dus

$$vqaw \vdash vbrw$$

als $\delta(q, a) = (r, b, R)$, en

$$vcqaw \vdash vrcbw$$

als $\delta(q, a) = (r, b, L)$

Om ervoor te zorgen dat v en w niet leeg zijn in een configuratie vqw mag je altijd v vervangen door $\Box v$ en w vervangen door $w\Box$

129

We staan toe dat δ een **partiële functie** is, dat wil zeggen dat we toestaan dat $\delta(q, a)$ ongedefinieerd is

Een rij opeenvolgende $\vdash$-stappen

$$vqw \vdash^* v'raw'$$

waarvoor $\delta(r, a)$ ongedefinieerd is heet een **berekening**

Deze notie van berekenen is deterministisch: voor elke configuratie c bestaat er maar hoogstens één configuratie c' met $c \vdash c'$, en voor elke beginconfiguratie c is er precies één berekening $c \vdash^* c'$ mogelijk

(dit in tegenstelling tot nfa's en pushdown-automaten)

De taal $L(M)$ geaccepteerd door een Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is gedefinieerd door

$$\{w \in \Sigma^+ \mid \exists q \in F, x, y \in \Gamma^* : q_0 w \vdash^* xqy\}$$

130

We gaan nu een Turing machine M maken waarvoor

$$L(M) = \{a^n b^n \mid n > 0\}$$

Deze taal is niet regulier: we zullen een soort intern geheugen nodig hebben om dit voor elkaar te krijgen

Het enige wat we daarvoor kunnen gebruiken is de string zelf

Met alleen maar één keer de string van links naar rechts te doorlopen lukt het niet

Idee: loop steeds heen weer over de string $a^n b^n$ en vervang daarbij steeds achtereenvolgens een a door een x en een b door een y

Als er uiteindelijk geen a's en b's meer over zijn, moeten er aanvankelijk evenveel a's als b's geweest zijn

---

131

We coderen dit als volgt:

- q_0: lees een a, vervang die door x en ga naar q_1
- q_1: loop naar rechts tot een b gelezen wordt, vervang die door y en ga dan naar q_2
- q_2: loop naar links tot een x gelezen wordt, ga dan terug naar q_0
- als je in q_0 geen a leest maar een y zijn blijkbaar alle a's door een x vervangen, ga dan naar q_3
- q_3: loop naar rechts over alle $y's$ om te kijken of er nog b's zijn overgebleven, zo nee ga dan naar de eindtoestand q_4

Hiermee krijgen we:

$$q_0aaabbb \vdash^*$$

$$xq_0aaybb \vdash^*$$

$$xxq_0ayyb \vdash^*$$

$$xxxq_0yyy \vdash^*$$

$$xxxyyy\Box q_4\Box$$

132

Dit is een stuk ingewikkelder dan met push-down-automaten

Echter: het breidt eenvoudig uit tot niet-contextvrije talen

Zo kun je geheel analoog een Turing machine M maken met

$$L(M) = \{a^n b^n c^n \mid n > 0\}$$

door a's b's en c's respectievelijk door x, y en z te vervangen met

$$q_0aaabbbccc \vdash^*$$

$$xq_0aaybbzcc \vdash^*$$

$$xxq_0ayybzzc \vdash^*$$

$$xxxq_0yyyzzz \vdash^*$$

$$xxxyyyzzz\Box q_4\Box$$

Bij het naar rechts lopen op zoek naar de eerste c hoeven we hierbij alleen onderweg een b door een y te vervangen

---

133

Zolang we alleen kijken naar berekenen, en niet naar het accepteren van een taal, spelen q_0 en F geen rol

Zo kunnen we een Turing machine zien als een eenvoudig soort computer, met een processor die volgens de gegeven spelregels de $\vdash$-stappen uitvoert en de band als geheugen

Nog wel een heel primitief soort geheugen: je kunt alleen maar daar lezen en schrijven waar de kop zich bevindt, bepaald geen RAM

De inhoud van de band aan het begin kun je als invoer van een programma of berekening zien, en de inhoud van de band na afloop als uitvoer

Het is verbazend hoe krachtig deze notie van berekenen is, zo kun je voor elke berekenbare functie f een Turing machine M vinden met berekening

$$q_0w \vdash^* q_f f(w)$$

134

Bijvoorbeeld: als $w \in \{0, 1, 2, \ldots, 9\}^*$ een getal in decimale notatie voorstelt, en $f(w)$ is gedefinieerd als de zeventiendemachtswortel van w tot in zeventien cijfers achter de komma, ook decimaal genoteerd, dan kun je een Turing machine M vinden met berekening

$$q_0 w \vdash^* q_f f(w)$$

Deze Turing machine is dan een machine die in staat is $f(w)$ voor je uit te rekenen voor elke w die je kunt verzinnen

Algemener: alle basisoperaties die een normale computer kent kunnen op een Turing machine worden uitgevoerd, en daarmee ook alle programma's die je daarmee kunt maken

Dit heet **Turing's thesis**, en heeft pas een preciese betekenis als je definieert wat die basisoperaties zijn