HOOFDSTUK II: Talen en automaten

Een hiërarchie van talen met bijbehorende hiërarchie van berekeningsmechanismen of grammatica's en bijbehorende hiërarchie van test en generatie procedures is een Chomsky hiërarchie.

Een string over een alfabet Σ is een opeenvolging van nul, één of meer elementen van Σ.

Een taal L over een alfabet Σ is een verzameling van eindige strings over Σ.

Σ^* is aftelbaar ∞ groot: er zijn ∞ veel strings, maar geen ∞ strings

Een alfabet is een eindige verzameling symbolen, karakters en tekens van Σ.

Gegeven 2 talen L_1 en L_2 over hetzelfde alfabet Σ, dan noteren we de concatenatie van L_1 en L_2 als $L_1 L_2$ en definiëren we: $L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$

L^* - de Kleene ster van een taal L: $L^* = \bigcup_{n=0}^{\infty} L^n$

Een reguliere expressie E over een alfabet Σ is van de vorm

- ϵ
- ϕ
- $a \in \Sigma$
- $(E_1 E_2)$ met E_1 en E_2 reguliere expressies over Σ
- $(E_1)^*$ met E_1 een reguliere expressie over Σ
- $(E_1 \mid E_2)$ met E_1 en E_2 reguliere expressies over Σ.

Een taal die door een reguliere expressie wordt bepaald is een reguliere taal.

Er zijn aftelbaar oneindig veel reguliere expressies over Σ. De verzameling talen bepaald door een reguliere expressie is ook aftelbaar oneindig groot (= aantal reguliere talen)

2.

Twee niet-deterministische toestandsautomaten (NFA's) worden equivalent genoemd als ze dezelfde taal bepalen.

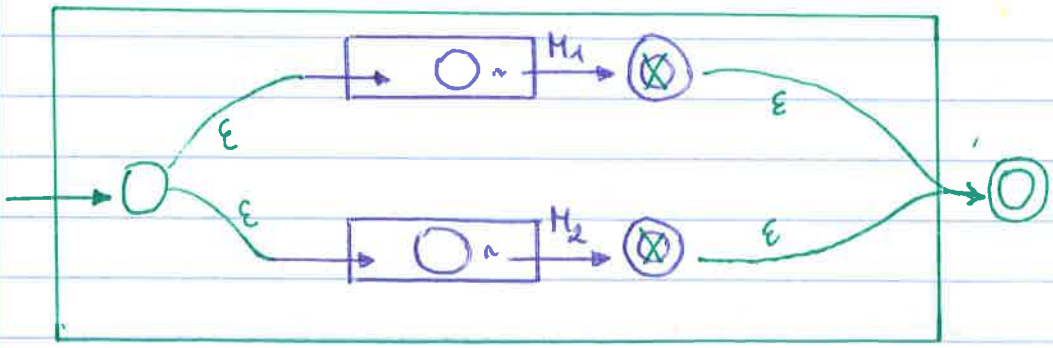
Een niet-deterministische eindige toestandsautomaat is een 5-tal $(Q, \Sigma, \delta, q_s, F)$ waarbij

- Q een eindige verzameling toestanden is
- Σ is een eindig alfabet
- δ is de overgangsfunctie van de automaat, $\delta: Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$
- q_s is de starttoestand (en is een element van Q)
- $F \subseteq Q$: F is de verzameling eindtoestanden.

Een string s wordt aanvaard door een NFA $(Q, \Sigma, \delta, q_s, F)$, indien s kan geschreven worden als $a_1 a_2 \dots a_n$ met $a_i \in \Sigma_\varepsilon$ en er een rij toestanden $t_1 t_2 \dots t_{n+1}$ bestaat zodat:

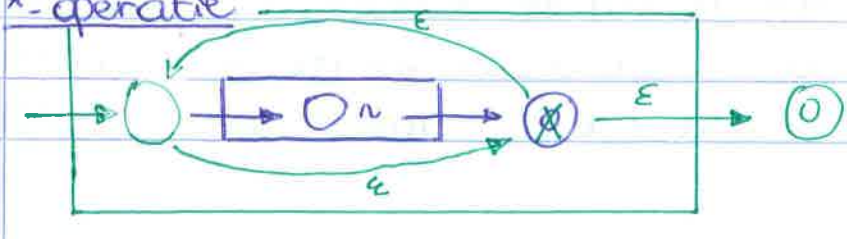
- $t_1 = q_s$
- $t_{i+1} \in \delta(t_i, a_i)$
- $t_{n+1} \in F$

Unie van 2 NFA's



Concatenatie van 2 NFA's

*-operatie



Een GNFA is een eindige toestandsautomaat met de volgende wijzingen en beperkingen:

- Er is slechts 1 eindtoestand en die verschilt van q_s
- Er is juist 1 boog van q_s naar elke andere toestand, maar er komen geen pijlen aan
- Er is juist 1 boog naar de eindtoestand van elke toestand, maar er vertrekken geen pijlen.
- Tussen elke 2 andere toestanden is er juist 1 boog in beide richtingen
- Er is juist 1 boog van elke toestand naar zichzelf
- De bogen hebben een reguliere expressie als label

Algoritme om van een NFA een RE te maken:

① Maak van de NFA een GNFA
② Reduceer de GNFA: verwijder de inwendige knopen tot 0
③ Bepaal RE: $0 \xrightarrow{RE} 0$

$n+2$ knopen $= O(n)$
om 1 knoop te verwijderen: $O(n^2)$ aanpassingen
$\Rightarrow O(n^3)$

Deterministische eindige toestandsmachine: slechts 1 mogelijke overgang per symbool en geen ε-overgangen

Als je een DFA M hebt: $L_M \in$ Reghan, $\forall L \in$ Reghan : $\exists M \in$ DFA : $L_M = L$

Algoritme om gegeven de NFA $(Q_n, \Sigma, \delta_n, q_{s_n}, F_n)$ de DFA $(Q_d, \Sigma, \delta_d, q_{s_d}, F_d)$ te construeren zodat $L_{NFA} = L_{DFA}$

$Q_d = \mathcal{P}(Q_n)$
$F_d = \{S \mid S \in Q_d, S \cap F_n \neq \emptyset\}$
$\delta_d : (\mathcal{P}(Q_n) \times \Sigma) \longrightarrow \mathcal{P}(Q_n)$
Definiëren we de afbeelding $eb : Q_n \to \mathcal{P}(Q_n)$

$$eb(q) = \delta_n^*(q, \varepsilon) \cup \{q\}$$

Voor $Q \in \mathcal{P}(Q_n)$: $eb(Q) = \bigcup_{q \in Q} eb(q)$

4.

Vervolgens definiëren we δ_d als:

$\delta_d(Q,a) = eb(\delta_m(Q,a))$ voor $Q \in Q_d$

Tenslotte definiëren we $q_{s_d} = eb(q_{s_m})$

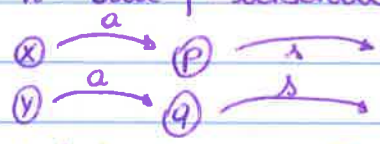
Twee toestanden p en q zijn f-gelijk indien

$\forall w \in \Sigma^* : \delta^*(p,w) \in F \Leftrightarrow \delta^*(q,w) \in F$

Algoritme om sets van f-gelijke toestanden te vinden

INIT: $P \leftarrow \{(p,q) \mid p \in F, q \notin F\}$

L: alle f-verschillen

LUS: 

→ x en y zijn niet als p en q ook f-verschillend

zoek (x,y) en $a \in \Sigma$: $\delta(x,a)$ f $\delta(y,a)$ en $(\delta(x,a), \delta(y,a)) \in P$

$\rightarrow (x,y) \rightarrow P \in P$

TOT er geen (x,y) zijn

NA LUS: P van f-verschillende toestanden (x,y)

- graaf met Q krijgen

Als $DFA_1 = (Q_1, \Sigma, \delta_1, q_s, F_1)$ een machine is zonder onbereikbare toestanden, dan bestaat er geen machine met strikt minder toestanden die dezelfde taal bepaalt.

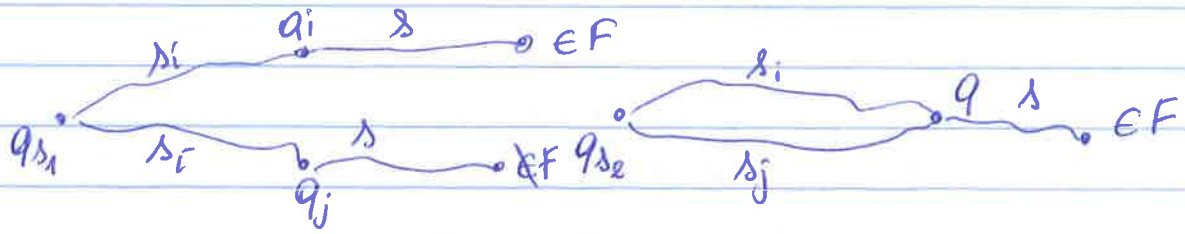
Bewijs: Stel: DFA_2 met $|Q_2| < |Q_1|$

voor $q_i \in Q_1$: s_i : $\delta_1^*(q_{s_1}, s_i) = q_i$

$\delta_2^*(q_{s_2}, s_i) \in Q_2$

$\Rightarrow \exists i, j, i \neq j : \delta_2^*(q_{s_2}, s_i) = \delta_2^*(q_{s_2}, s_j)$

$\Rightarrow DFA_1$ en DFA_2 bepalen niet dezelfde taal:



$DFA_1(Q_1, \Sigma, \delta_1, q_{s_1}, F_1)$ is isomorf met $DFA_2(Q_2, \Sigma, \delta_2, q_{s_2}, F_2)$ indien er een bijectie $b: Q_1 \to Q_2$ bestaat zodat

- $b(F_1) = F_2$
- $b(q_{s_1}) = q_{s_2}$
- $b(\delta_1(q,a)) = \delta_2(b(q), a)$

Een partitie P_1 is fijner dan P_2 indien elk element van P_1 vervat zit in een element van P_2.

reach(q) is de verzameling strings die je van de begintoestand in toestand q brengen.

$$x \sim_{DFA} y \Leftrightarrow \delta^*(q_s, x) = \delta^*(q_s, y)$$

Myhill-Nerode relaties voor L, zijn equivalentie-relaties die voldoen aan:

1. $\forall x,y \in \Sigma^*, a \in \Sigma : x \sim_{DFA} y \to xa \sim_{DFA} ya$ ($\sim_{DFA}$ is rechts congruent)
2. $\sim_{DFA}$ verfijnt $\sim_L$ of: $x \sim_{DFA} y \to x \sim_L y$
3. Het aantal equivalentieklassen van $\sim_{DFA}$ is eindig

Gegeven een taal L over Σ en een MN(L)-relatie $\sim$ op Σ^*, dan definieert $(Q, \Sigma, \delta, q_s, F)$ een DFA die L bepaalt, waarbij

- $Q = \{x_\sim \mid x \in \Sigma^*\}$: elke equivalentieklasse is een toestand
- $q_s = \varepsilon_\sim$: de starttoestand bereik je met ε
- $F = \{x_\sim \mid x \in L\}$: eindtoestanden worden met strings uit L bereikt
- $\delta(x_\sim, a) = (xa)_\sim$

De overgang van DFA naar MN.-relatie, en van daar naar een DFA, zijn elkaars isomorfen (op een DFA-isomorfisme na).

$x \sim_{sup} y$ is de transitieve sluiting van $(x \sim_1 y) \vee (x \sim_2 y)$
Het supremum is de fijnste relatie die groffer is dan beide relaties en die omvat.

6.

Het supremum van 2 MN(L)-relaties is ook een MN(L)-relatie:
als $\sim_1$ en $\sim_2$ MN(L)-relaties zijn, dan is het supremum $\sim_{sup}$ van $\sim_1$ en $\sim_2$ ook een MN(L)-relatie.

Bewijs

1. $x \sim_{sup} y \equiv \exists z_i : ((x \sim_1 z_1) \vee (x \sim_2 z_1)) \wedge ((z_1 \sim_1 z_2) \vee (z_1 \sim_2 z_2)$
$\wedge \ldots \wedge ((z_m \sim_1 y) \vee (z_m \sim_2 y))$
$\Rightarrow \exists z_i : \forall a \in \Sigma : (xa \sim_1 z_1 a) \vee (xa \sim_2 z_1 a) \ldots$
$\Rightarrow \forall a \in \Sigma : xa \sim_{sup} ya$
$\Rightarrow \sim_{sup}$ is rechtscongruent

3. Stel $x, y \in L$ en $x \sim_{sup} y$
dan: $\exists z_i : ((x \sim_1 z_1) \vee (x \sim_2 z_1)) \wedge \ldots$
$\Rightarrow z_1 \in L \wedge \exists z_i : ((z_1 \sim_1 z_2) \vee (z_1 \sim_2 z_2)) \wedge \ldots$
$\Rightarrow z_2 \in L \wedge \exists z_i : ((z_2 \sim_2 z_3) \vee (z_2 \sim_2 z_3)) \wedge \ldots$
$\Rightarrow y \in L$ (idem voor $\bar{L}$)
$\Rightarrow \sim_{sup}$ verfijnt $\sim_L$

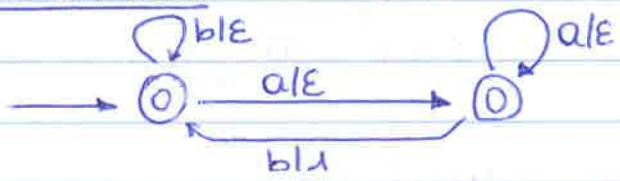
3. Het aantal equivalentieklassen voor $\sim_{sup}$ is niet hoger dan het aantal equivalentieklassen van 2 $\sim_i$, dus eindig.

(Stelling van Myhill en Nerode)
Laat $L \subseteq \Sigma^*$ een taal zijn over Σ, dan zijn deze uitspraken equivalent:
1. L is regulier
2. Er bestaat een Myhill-Nerode relatie voor L
3. Definieer $\sim$ op Σ^*: $x \sim y \Leftrightarrow \forall s \in \Sigma^* : (xs \in L \Leftrightarrow ys \in L)$
en dus heeft de relatie $\sim$ een eindige index

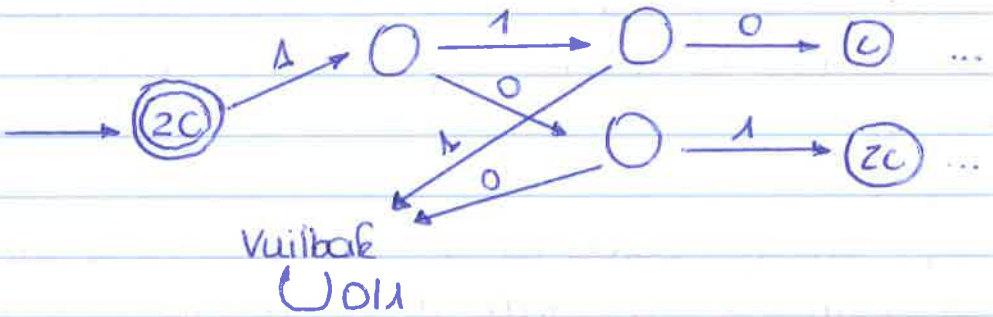
(pompend lemma voor reguliere talen)
Voor een reguliere taal L bestaat een pomplengte d, zodanig dat als $s \in L$ en $|s| > d$, dan bestaat er een verdeling van s in stukken x, y en z zodat $s = xyz$ en 1. $\forall i \geq 0 : xy^i z \in L$
2. $|y| > 0$
3. $|xy| \leq d$

Bewijs: Neem een DFA die L bepaalt en d = # toestanden + 1
Neem dan een willekeurige $s = a_1 a_2 \dots a_m$ met $m \geq d$.
Beschouw de accepterende rij van toestanden voor s.
deze heeft een strikte lengte groter dan d
$\Rightarrow$ bij de eerste d zijn er zeker 2 toestanden gelijk
(er zijn slechts $d-1$ toestanden!)
Stel: q_i en q_j $(i \neq j)$ zijn gelijk met $i < j \leq d$,
dan nemen we $x = a_1 a_2 \dots a_i$ en $y = a_{j+1} \dots a_j$ en z
de rest van de string.
⚠ Niet alle strings kunnen gepompt worden.

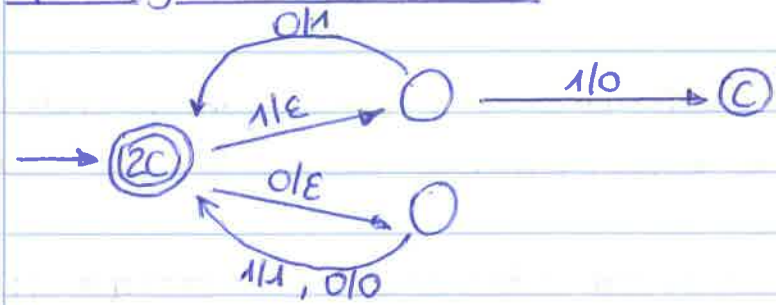
Transducer



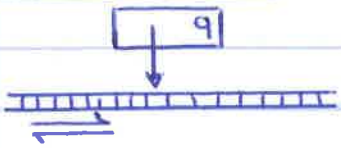
Optelling met een DFA



Optelling met transducer



Two-way DFA

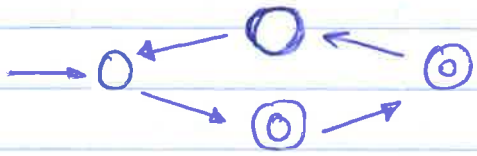


$\delta \; x \; q \; h, R\}$

8.

Büchi-automaten

Een oneindige string s wordt aanvaard door een Büchi automaat indien de rij toestanden waarlangs je passeert oneindig vaak een aanvaarde toestand heeft.



Een contextvrije grammatica is een 4-tal (V, Σ, R, S) waarbij

- V een eindige verzameling niet-eindsymbolen is
- Σ een eindig alfabet van eindsymbolen is, disjunct met V
- R een eindige verzameling regels
  Een regel is een koppel van 1 niet-eindsymbool en een string van elementen $V \cup \Sigma_\varepsilon$.
- S is het startsymbool en behoort tot V

Gegeven een CFG (V, Σ, R, S). Een string f over $V \cup \Sigma_\varepsilon$ wordt afgeleid uit een string b over $V \cup \Sigma_\varepsilon$ met behulp van de CFG als er een eindige rij strings $s_0, s_1, \ldots, s_m$ bestaat zodat:

- $s_0 = b$
- $s_m = f$
- s_{i+1} verkregen wordt uit s_i $(i < m)$ door in s_i een niet-eindsymbool X te vervangen door de rechterkant van een regel waarin X links voorkomt.

De taal L_{CFG} bepaald door een CFG (V, Σ, R, S) is de verzameling strings over Σ die kunnen afgeleid worden van het startsymbool S.

Een taal L is contextvrij indien er een CFG bestaat zodat $L = L_{CFG}$

Twee contextvrije grammatica's CFG_1 en CFG_2 zijn equivalent indien $L_{CFG_1} = L_{CFG_2}$.

Omdat er voor een string meerdere afleidingsbomen kunnen bestaan, is een string ambigu.

Een CFG heeft de Chomsky Normaal vorm als elke regel een van deze vormen heeft: 1. $A \to BC$
2. $A \to \alpha$
3. $S \to \varepsilon$

waarin α een eindsymbool is, A een niet-eindsymbool en B en C niet-eindsymbolen verschillend van S, het startsymbool.

Voor elke CFG bestaat er een CFG' die equivalent is en in Chomsky NV staat.

Bewijs

1. Als S het startsymbool is in de grammatica, vervang het dan door een nieuw niet-eindsymbool X en voeg de regel $S \to X$ toe
2. Stel dat we een regel $\mathcal{E} = A \to \varepsilon$ hebben en een regel $\mathcal{R} = B \to \gamma$ waarbij A voorkomt in γ, dan definiëren we de verzameling regels $V(E,R)$ als de verzameling regels van de vorm $B \to \eta$ waarin η verkregen wordt uit γ door alle voorkomens van A eruit weg te laten
We transformeren de grammatica: zolang er regels $\mathcal{E} = A \to \varepsilon$ en $R = B \to \gamma$ zijn zodat $V(E,R)$ nieuwe regels bevat, voeg $V(E,R)$ toe aan de grammatica
Vervolgens verwijderen we uit de grammatica alle regels van de vorm $A \to \varepsilon$ behalve als $A = S$
3. We definiëren de regel $U(E,R) = A \to \gamma$: zolang er regels van de vorm $\mathcal{E} = A \to B$ (B is een niet-eindsymbool) en $\mathcal{R} = B \to \gamma$ bestaan en $U(E,R)$ een nieuwe regel is, voeg $U(E,R)$ toe aan de grammatica.
Vervolgens verwijderen we uit de grammatica alle regels van de vorm $A \to B$
4. * regels $A \to \gamma$ waarin γ uit juist 2 niet-eindsymbolen bestaat, blijven
* regels $A \to \gamma$ waarin γ minstens uit 2 niet-eindsymbolen bestaat: we vervangen a door A_a en voegen de regel $A_a \to a$ toe
5. Regels van de vorm $A \to X_1 X_2 \dots X_m$ $(m > 2)$ vervangen we door: $A \to X_1 Y_1$
$Y_1 \to X_2 Y_2, \dots, Y_{m-2} \to X_{m-1} X_m$

10.

Een push-down automaat is een 6-tal $(Q, \Sigma, \Gamma, \delta, q_s, F)$ waarbij
- Q is een eindige verzameling toestanden
- Σ is een eindig inputalfabet
- Γ is een eindig stapelalfabet
- δ is een overgangsfunctie: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$
- q_s is de starttoestand
- $F \subseteq Q$ is de verzameling eindtoestanden

Een string s wordt aanvaard door een PDA indien s kan worden opgesplitst in delen w_i ($i = 1 \dots m$ en $w_i \in \Sigma_\varepsilon$) zodat er toestanden q_j ($j = 0 \dots m$) zijn en stacks $stack_k$ ($k = 0 \dots m$ met $stack_k \in \Gamma_\varepsilon^*$) zodat
- $stack_0 = \varepsilon$
- $q_0 = q_s$
- $q_m \in F$
- $(q_{i+1}, y) \in \delta(q_i, w_{i+1}, x)$ met $x, y \in \Gamma_\varepsilon$ en $stack_i = xt$, $stack_{i+1} = yt$ met $t \in \Gamma_\varepsilon^*$

De taal L bepaald door een PDA bestaat uit alle strings die door de PDA aanvaard worden.

Elke PDA bepaalt een contextvrije taal en elke contextvrije taal wordt bepaald door een PDA.

De constructie van een PDA uit een CFG, levert een PDA die L_{CFG} accepteert

(Pompend lemma voor Contextvrije talen)
Voor een contextvrije taal L bestaat een getal p zodat elke string s van L met lengte minstens p kan opgedeeld worden in 5 stukken u, v, x, y en z uit Σ^* zodat $s = uvxyz$
1. $\forall i \geq 0: uv^i xy^i z \in L$
2. $|vy| > 0$
3. $|vxy| \leq p$

Bewijs: Neem een CFG in Chomsky normaalvorm voor L
laat n het aantal niet-eindsymbolen van de CFG zijn.
Voor een string s uit L bestaat er een afleidingsboom
Als je van deze boom de onderste takken verwijdert heb je
een volledig binaire boom. De hoogte van deze boom is dan
minstens gelijk aan $\log_2 |s|$. Het langste pad vanuit de
wortel bevat dus minstens $\log_2 |s| + 1$ knopen. Als we s lang
genoeg kiezen is $\log_2 |s| + 1 > n$
$\Rightarrow$ op het langste pad moet er dus minstens 1 niet-
eindsymbool X herhaald worden.
Neem de langste X $(= X_2)$ en zijn dichtste herhaling X_1 op dat
pad.

$$S \Rightarrow^* u X_2 z \Rightarrow^* uv X_1 yz \Rightarrow^* uvxyz \quad \circledast$$

waarbij u, v, x, y en z strings uit Σ^* zijn. Bovendien zijn v en y
niet tegelijk leeg. Uit $\circledast$ volgt: $S \Rightarrow^* u X_2 z \Rightarrow^* uxz$
en dus $S \Rightarrow^* uXz \Rightarrow^* uvXyz \Rightarrow^* uvvxyyz$
Als we nu strings nemen die langer zijn dan 2^{n-1}, dan wordt
onze pomplengte p.
vxy wordt afgeleid uit X met een afleidingsboom kleiner dan n
en met dus hoogstens 2^n bladeren die corresponderen met vxy.

Een ambigue taal (> 1 afleidingsboom) kan niet deterministisch zijn.

Chomsky hiërarchie

	grammatica	taal	automaat
	type-0	unrestricted	herkenbaar	turingmachine
polynoom, exponentieel of sneller	type-1	context-sensitief	context-sensitief	lineair begrensd
constante ruimte	type-2	context-vrij	context-vrij	push-down
$O(n^3)$				
$O(n)$	type-3	regulier	regulier	eindig

HOOFDSTUK 3: Talen en Berekenbaarheid

Een **Turing machine** is een 7-tal $(Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$ waarbij Q, Σ en Γ eindige verzamelingen zijn en

- Q is een verzameling toestanden
- Σ is een inputalfabet dat niet # bevat
- Γ is het tape alfabet, $\# \in \Gamma$ en $\Sigma \subset \Gamma$
- q_s is de starttoestand
- q_a is de accepterende eindtoestand
- q_r is de verwerpende eindtoestand $(\neq q_a)$
- δ is de transitiefunctie : $Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$

Een Turing-machine TM **herkent** L_{TM}

Een taal L is **Turing-herkenbaar** indien er een Turingmachine TM bestaat zodat $L = L_{TM}$.

Een Turingmachine TM **beslist** een taal L, als TM L herkent en bovendien $\infty_{TM} = 0$.

Een taal L heet **Turing-beslisbaar** als er een Turingmachine is die L beslist.

Een taal L is **co-herkenbaar/co-beslisbaar** als $\bar{L}$ herkenbaar/beslisbaar is

Als L beslisbaar is, is L ook cobeslisbaar.
Bewijs In de Turingmachine die L beslist, verwissel je de rol van q_r en q_a.

Als L herkenbaar is en co-herkenbaar, is L beslisbaar.

Bewijs: M_1 is de machine die L herkent en M_2 is de machine die $\overline{L}$ herkent.
Laten we nu M_1 en M_2 samen lopen als een nieuwe machine M in parallel: zodra M_1 accepteert, accepteert M en zodra M_2 accepteert, verwerpt M.
M_1 en M_2 kunnen niet samen accepteren en voor elke string zal minstens 1 machine stoppen.

Er bestaat een taal die niet herkenbaar is.

A_{TM} is niet beslisbaar. ($A_{TM} = \{\langle M, s\rangle \mid M$ is een turingmachine en $s \in L_M\}$)

Bewijs: Stel dat er een beslisser B bestaat voor A_{TM}. Dat betekent dat bij input $\langle M, s\rangle$ B accepteert als M bij input s stopt in zijn q_a. We schrijven: $B(\langle M, s\rangle)$ is accept als M s accept en anders: reject.
Construeer nu de contradictiemachine C met eigenschap:
$C(\langle M\rangle)$ = opposite $(B(\langle M, M\rangle))$ voor elke TM M.
Nemen we voor M C zelf krijgen we: $C(\langle C\rangle)$ = opposite $(B(\langle C, C\rangle))$
Als $C(\langle C\rangle)$ accept dan zal $B(\langle C, C\rangle)$ accepteren en dus ontstaat er een contradictie: accept = opposite (accept)
Dus kan C niet bestaan
$\Rightarrow$ B bestaat niet
$\Rightarrow$ A_{TM} is niet beslisbaar

H_{TM} is niet beslisbaar. ($H_{TM} = \{\langle M, s\rangle \mid M$ is een turingmachine die stopt bij input s$\}$)

Bewijs: Stel dat H_{TM} beslisbaar is door turingmachine H, dan construeren we een beslisser B voor A_{TM}: bij input $\langle M, s\rangle$ doet B:
- laat eerst H lopen op $\langle M, s\rangle$
- Als $H(\langle M, s\rangle)$ = accept, laat B M op s lopen en geeft als resultaat wat M geeft
- Als $H(\langle M, s\rangle)$ = reject, dan verwerpt B ook $\langle M, s\rangle$.

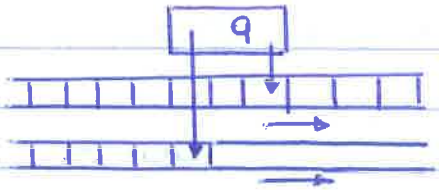
$\Rightarrow$ Er is een beslisser voor A_{TM} = CONTRADICTIE
$\Rightarrow$ H kan niet bestaan.

14.

H_{TM} is herkenbaar
(H is een herkenner voor H_{TM}. H accept zodra H_{TM} stopt)
A_{TM} is herkenbaar.

$\overline{A_{TM}}$ en $\overline{H_{TM}}$ zijn niet herkenbaar.

Enumeratormachine:



$ = scheidingsteken

De taal door een enumerator bepaald is herkenbaar en elke herkenbare taal wordt door een enumerator geënumereerd.

Bewijs
1. Geef een string s aan de herkenner TM en start de enumerator Enu. Telkens Enu in zijn q_e komt, kijkt TM of de laatst geproduceerd string op de outputband van Enu gelijk is aan s. Indien ja, accepteert TM, indien niet: laat TM Enu verderrekenen.
2. laat TM L bepalen. Nu construeren we de Enumerator Enu voor L:
Maak een TM_{gen} die gegeven een getal n de eerste n strings uit Σ^* op de band zet: $s_1, s_2, \ldots, s_n$.
Maak een TM_n die op elk van die strings, n stappen van TM uitvoert. Als daarbij een string s_i aanvaard wordt, schrijf je die op de outputband van Enu.
Maak nu TM_{driver} die de opeenvolgende getallen n genereert en dan TM_{gen} en TM_n oproept.

Een lineair begrensde automaat is een turingmachine die niet leest of schrijft buiten het deel van de band dat initieel invoer bevat.

	BESLISBAAR	HERKENBAAR	NIET-HERKENBAAR
Reghan	"Alles" is beslisbaar $A_{\mathcal{H}}$, All, Empty, E, EQ	\	\
CFL	A, Empty, E	$\overline{EQ_{CFG}}$, $\overline{All_{CFG}}$	EQ_{CFG}, All_{CFG}
CSL	A_{CBA}, H_{CBA}	$\overline{E_{LBA}}$	E_{LBA}
TM	"niets" is beslisbaar	H_{TM}, A_{TM}, [illegible] $IsIn_{TM,Reghan}$	E_{TM}

Een eigenschap P van Turingmachines heet niet triviaal indien:
$pos_P \neq 0$ en ook $neg_P \neq 0$.
Niet alle Turingmachines hebben P, maar $\exists$ TM die P heeft
De eigenschap P heet taal-invariant indien $L_{M_1} = L_{M_2} \Rightarrow P(M_1) = P(M_2)$
Alle machines die dezelfde taal lepalen hebben ofwel P ofwel heeft geen enkele P.

(Stelling van Rice I.)

Voor elke niet-triviale en taal-invariante eigenschap P van Turingmachines geldt dat pos_P niet beslisbaar is.

Bewijs: Stel dat $M_\emptyset$ de eigenschap P niet heeft.
Vermits P niet-triviaal is, bestaat er een taal L_X zodat X een Turingmachine is met eigenschap P.
Stel dat Pos_P beslisbaar is: A beweegt dan als input $\langle M, s \rangle$:
- construeer de machine $H_{M,s}$ die bij input x:
  - laat M lopen op s
  - als M accepteert: laat X lopen op x en accepteer als X x accepteert
- laat nu de beslisser B voor Pos_P lopen op $H_{M,s}$
  Als B $H_{M,s}$ accepteert, dan: accept, anders: reject

A accepteert $\langle M, s \rangle$ als en slechts als B $H_{M,s}$ accepteert, als en slechts als $H_{M,s}$ de eigenschap P heeft, als en slechts als $H_{M,s}$ L_X accepteert, als en slechts als M accepteert s.
$\Rightarrow$ A is een beslisser voor A_{TM} = CONTRADICTIE
$\Rightarrow$ B kan niet bestaan
$\Rightarrow$ Pos_P is niet beslisbaar.

16.

Een functie f heet Turingberekenbaar indien er een Turingmachine M bestaat die bij input s uiteindelijk stopt met $f(s)$ op de band

Taal L_1 over Σ_1 kan naar taal L_2 over Σ_2 gereduceerd worden indien er een afbeelding f met signatuur $\Sigma_1^* \to \Sigma_2^*$ bestaat zodat $f(L_1) \subseteq L_2$ en $f(\overline{L_1}) \subseteq \overline{L_2}$ en zodanig dat f turingberekenbaar is. We noteren $L_1 \leq_m L_2$.

Als $L_1 \leq_m L_2$ en L_2 is beslisbaar, dan is L_1 beslisbaar.
herkenbaar herkenbaar.
L_1 is niet herkenbaar, dan is L_2 niet-herkenbaar

Er bestaat een orakelmachine $O^{A_{TM}}$ die E_{TM} beslist.
Bewijs: We construeren $O^{A_{TM}}$: bij input $\langle M \rangle$ doet $O^{A_{TM}}$:
- construeer een turingmachine P die bij input ω
  - laat M lopen op alle strings van Σ^*
  - als M een string accepteert: accept
- vraag aan het orakel voor A_{TM} of $\langle P, x \rangle \in A_{TM}$
- als het orakel ja antwoord: reject, anders: accept

Als $L_M \neq \emptyset$ dan accepteert P elke input en zeker input x.
$\Rightarrow O^{A_{TM}}$ moet verwerpen
Als $L_M = \emptyset$ moet $O^{A_{TM}}$ accepteren

Een taal A is Turingreduceerbaar naar een taal B, indien A beslisbaar is relatief ten opzichte van B: er bestaat een O^B die A beslist
We schrijven $A \leq_T B$.

Indien $A \leq_T B$ en B is beslisbaar, dan is A beslisbaar

Indien $A \leq_m B$ dan is $A \leq_T B$.

Alle functies die je kan maken door te vertrekken van de basisfuncties en door gebruikmaking van compositie en primitieve recursie, noemen we **primitief recursief**.

Basisfuncties:
- nulfunctie: $nul : \mathbb{N} \to \mathbb{N}$, $nul(x) = 0$
- successorfunctie: $suc : \mathbb{N} \to \mathbb{N}$, $suc(x) = x+1$
- projectie: $p_i^m : \mathbb{N} \to \mathbb{N}$, $p_i^m(x_1, \dots, x_m) = x_i$

compositie: gegeven: $g_1, \dots, g_m$ functies $\mathbb{N}^k \to \mathbb{N}$ en $f: \mathbb{N}^m \to \mathbb{N}$
$\to h: \mathbb{N}^k \to \mathbb{N}$, $h(\bar{x}) = f(g_1(\bar{x}), \dots, g_m(\bar{x}))$

primitieve recursie: gegeven: $f: \mathbb{N}^k \to \mathbb{N}$ en $g: \mathbb{N}^{k+2} \to \mathbb{N}$
$\to h: \mathbb{N}^{k+1} \to \mathbb{N}$, $h(\bar{x}, 0) = f(\bar{x})$
$h(\bar{x}, y+1) = g(\bar{x}, y, h(\bar{x}, y))$

Ackermann: $Ack(0,y) = y+1$
$Ack(x+1, 0) = Ack(x, 0)$
$Ack(x+1, y+1) = Ack(x, Ack(x+1, y))$

Recursieve functies verkrijg je uit de basisfuncties en toepassen van primitieve recursie, compositie en onbegrensde minimisatie. Die functies worden ook μ-recursie genoemd.

HOOFDSTUK 4: Herschrijfsystemen

Een voorkomen van een variabele x is vrij in de expressie E indien
- $E == x$
- $E == (AB)$ en het voorkomen van x is vrij in A of B
- $E == (\lambda y.\ A)$ en $x \neq y$ en x komt vrij voor in A

$A_+\ C_n\ C_m = C_{n+m}$
$A_*\ C_n\ C_m = C_{n*m}$
$A_{exp}\ C_n\ C_m = C_{n^m}$ voor $m > 0$

(Church-Rosser I)
Indien $E_1 \overset{*}{\leftrightarrow} E_2$, dan bestaat er een expressie E zodat $E_1 \xrightarrow{*} E$ en $E_2 \xrightarrow{*} E$

(Uniciteit van de normaalvorm)
Geen expressie kan geconverteerd worden naar 2 verschillende normaalvormen.
Bewijs: Stel $E \overset{*}{\leftrightarrow} E_1$ en $E \overset{*}{\leftrightarrow} E_2$ waarbij E_1 en E_2 in normaalvorm staan, dan is $E_1 \overset{*}{\leftrightarrow} E_2$ en dus bestaat er een expressie F: $E_1 \xrightarrow{*} F$ en $E_2 \xrightarrow{*} F$, maar gezien E_1 en E_2 geen redexen (deeltermen die gereduceerd kunnen worden) bevatten, moet $E_1 = F = E_2$

De normaalorde bepaalt dat de meest linkse, buitenste redex eerst moet behandeld worden.

(Church-Rosser II)
Indien $E \xrightarrow{*} N$ en N staat in normaalvorm, dan bestaat er een reductierij in normaalorde van E naar N.

(28/09/2010) # Automaten en Berekenbaarheid

HOOFDSTUK 1: Talen en automaten

een automaat: Rekent: input → output

functie: domein → bereik (moet niet injectief zijn)

= PROGRAMMA: • niet-deterministisch programma's (domein = hele wereld)

• oneindig lopende programma's (bereik ∪ ⊥)

! een algoritme stopt altijd

DOMEIN vb * ℕ ONEINDIG

* {t,f} = boolean — EINDIG
* {sudokupuzzels}
* ~~ℝ ONEINDIG niet-aftelbaar~~
* ~~ℚ ONEINDIG aftelbaar~~

BEREIK vb * ℕ

* {t,f,m}
* {ingevulde sudokupuzzels}
* ~~ℝ~~
* ~~ℚ~~

4 soorten functies:

~~eindig~~ → eindig

~~eindig~~ → oneindig

oneindig → eindig

oneindig → oneindig

ONEINDIG = aftelbaar

{t,f} → beschouwen we voor eindige bereiken

{t,f,m} } → niet nodig vb welke dag van de week is het beste om te studeren!

{m,d,...,z} en {ja,nee}

↳ implementeren als functies met 2 waarden

{t} = CONSTANTE functie: heb je niets aan!

Alle programma's die slechts een eindig domein aanvaarden, zijn TRIVIAAL.

Een string over een alfabet Σ is een opeenvolging van nul, één of meer elementen van Σ.
Een taal over een alfabet Σ is een verzameling van eindige strings over Σ.

$\mathbb{N} \to \mathbb{N}$ | → decimale notatie voor $\mathbb{N}$
$\mathbb{N} \to \{0,1\}$ | → opeenvolging van tekens uit een alfabet Σ
waarbij Σ een eindige verzameling van tekens is.

→ We bekijken de functies van strings die je kan maken van Σ = Σ^* naar {true, false}.

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

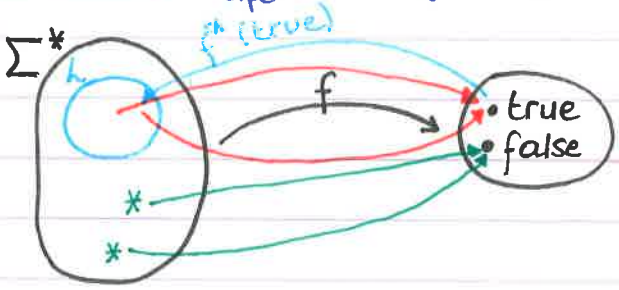
$\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, \ldots\}$

Hoe groot is Σ^*?

Er zijn oneindig veel eindige strings

Er zijn geen oneindige strings (≠ aanwaarbaar voor de computer)

→ aftelbaar : $\bigcup_{\text{aft}}$ eindig = aftelbaar



= verdeelt Σ^* in 2 delen met $f(L)$ = true.

→ L noemen we een taal, waarbij L een deelverzameling is van Σ^* (met $\Sigma = \{0,1\}$).

vb Σ^*: L = strings met een lengte ↝ loop constructie

Bestaat dit? ja. constructief bewijs (makkelijk te implementeren)

L = {0} ↝ if-then-else constructie

We beschouwen verschillende soorten talen met de machinerie die nodig is om die taal te kunnen programmeren.

vb beslissing of string s in L zit.

aantal functies dat L beslist: 1: functie moet true teruggeven voor elk element in L en false anders.

aantal algoritmen dat L beslist: aftelbaar oneindig

bestaat er een kleinste/kortste algoritme? ja: de verzameling van algoritmen die de lengte van strings van L beslissen ≠ ϕ.

java-programma's Σ = eindig alfabet, javac Σ^* = aftelbaar

```
{int i = 0
}
```

Bestaat er een algoritme dat de kleinste beslisser voor L vindt ?

h

'tel alle strings af' → $s \in \Sigma^*_{java}$ ↝ oneindige lus
'test of s een javaprogramma is' = stukje van de compiler
'test of s f_L implementeert'
↳ IMPLEMENTATIE voor elke $p \in \Sigma^*$ h ↝ oneindige lus
test of $s(p) \equiv f_L(p)$

h

h

Er bestaat een kortste programma, maar we kunnen het niet op een constructieve manier vinden noch herkennen.

(29/09/2010) 1. Algebra van talen

Een alfabet is een eindige verzameling van symboden, karakters, tekens en Σ.

Een string ("tekening") over Σ is een eindige rij tekens $\in \Sigma$.
Een verzameling van strings = Σ^*.
Een taal L over $\Sigma \subseteq \Sigma^*$.
Σ^* is aftelbaar oneindig.
Als L een taal is over Σ : $L \in \mathcal{P}(\Sigma^*)$
$\emptyset \in \mathcal{P}(\Sigma^*)$

- deelverzameling van dat tussen haakjes
- volle taal
- lege taal

Hoeveel talen over Σ ?
↕
Hoeveel elementen in $\mathcal{P}(\Sigma^*)$?

→ minstens oneindig veel
→ overaftelbaar

De cardinaliteit van een verzameling V < cardinaliteit $\mathcal{P}(V)$

$\#\mathbb{N} < \#\mathcal{P}(\mathbb{N}) = \mathbb{R}$: $0, n_1, \dots n_i \in \mathbb{R}$ met $n_1, \dots, n_i \in \mathbb{R}$

interesse in:

① taal L = beschrijving van L

vb $\Sigma = \{a,b\}$ $\quad L_e = \{s \in \Sigma^* \mid |s| \bmod 2 = 0\}$

→ $\exists$ algoritme dat beslist of $s \in L_e$

② algoritme of procedure om L te beslissen

$s \in L$

$f: \Sigma^* \to \{0,1\}$

$f(s) = 0$ indien $s \notin L$

$f(s) = 1$ indien $s \in L$

→ deze functie bestaat altijd.

Algebra van talen: verzameling V met daarop operaties

- binair: L_1, L_2 dan $L_1 \cup L_2$, $L_1 \cap L_2$

$L_1 L_2 = \{s_1 s_2 \mid s_1 \in L_1,\ s_2 \in L_2\}$

- unair: L (over Σ) dan $\bar{L} = L^c$.

$L^2 = LL$ of $L^n = L L^{(n-1)}$ $\quad n \in \mathbb{N}_0$.

$L^* = \bigcup_{n \in \mathbb{N}} L^n \subseteq \Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$ = kleene sluiting

→ zeker een opeenvolging van tekens uit Σ^*

→ eindig opeenvolging

2. Reguliere expressies

BASISGEVALLEN:
- ε
- ϕ
- $\forall c \in \Sigma,\ c$

REGELS om nieuwe reguliere expressies te maken uit de oude:

stel E_1 en E_2 zijn regExp:
- $(E_1 E_2)$
- $(E_1)^*$
- $(E_1 | E_2)$

Hoeveel reguliere expressies over Σ zijn er?

~~eindig~~

oneindig — aftelbaar

~~overaftelbaar~~

① stel: $\Sigma = \{a,b,c\}$
→ $a \in$ regExp
$(aa) \in$ regExp
$((aa)a)\,((aa)a)\ldots)\ldots \infty$

② stel: $\Sigma = \{a,b,c\}$ en alfabet: $\Sigma \cup \{(,),|,*,\varepsilon,\phi\} = A$
→ regExp over Σ is een taal over A
regExp over $\Sigma \subseteq A^*$ met A^* aftelbaar
⇒ aantal regExp over Σ kan hoogstens aftelbaar zijn

Een taal over Σ bepaald door een regExp_Σ.
→ inductief definiëren:

regExp →	talen
• ε	• $\{\varepsilon\} = L_\varepsilon$
• ϕ	• $\{\} = L_\phi$
• $c \in \Sigma$	• $\{c\} = L_c$
• $(E_1 E_2)$	• $L_{E_1} L_{E_2}$
• $(E_1)^*$	• $L_{E_1}^*$
• $(E_1 \mid E_2)$	• $L_{E_1} \cup L_{E_2}$

Als $E \in$ regExp, dan is L_E de taal door E bepaald

→ complement $\bar{L}$
doorsnede $\cap$
$A \cup B = \overline{(\bar{A} \cap \bar{B})}$

Hoe groot is de verzameling van talen bepaald door een reguliere expressie over Σ?

- ~~eindig~~
- oneindig
  - aftelbaar
  - ~~overaftelbaar~~

- 1 RegExp bepaalt 1 taal
- 2 verschillende RegExp bepalen:
  - 2 talen
    vb $(a(bc)) \to \{abc\}$
    $((ab)c)$ ↗
  - $\{\}$
    ϕ ← $(\phi|\phi)$
- oneindig veel RegExp bepalen 1 taal: $((R|\phi)|\phi)\ldots$

6.

Reguliere taal $L \triangleq \exists E \in RegExp \;\&\; L = L_E$.

Hoeveel reguliere talen zijn er?

Reghan = verzameling reguliere talen

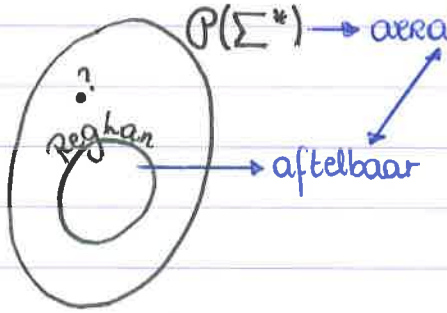
aantal reguliere talen = oneindig aftelbaar

①	RegExp
a	a
aa	aa
aaa	(aa)a
⋮	⋮

→ ONEINDIG

② aantal reguliere talen ≤ aantal reguliere expressies = aftelbaar

Bestaat er een taal die niet regulier is?



$\mathcal{P}(\Sigma^*)$ → overaftelbaar = strikt meer
⇒ de meeste talen zijn niet reguliere

Enkele uitdrukkingen

① Reghan $\subseteq \Sigma$: is ofwel fout (typefout) of correct
set van slt van strings slt van tekens

② Reghan $\subseteq \Sigma^*$ → ERROR

③ Reghan $\subseteq \mathcal{P}(\Sigma)$ → ERROR

④ Reghan $\subseteq \mathcal{P}(\Sigma^*)$ → correct

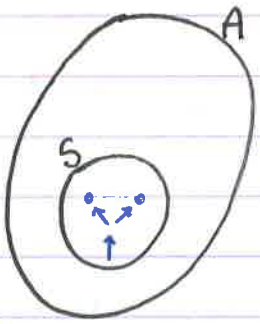
⑤ Reghan $\subseteq \mathcal{P}(\mathcal{P}(\Sigma^*))$ → typefout

⑤' Reghan $\in \mathcal{P}(\mathcal{P}(\Sigma^*))$ → $x \in y \Rightarrow$ y is slt van type x
⇒ correct

⑬ indien $x \in$ Reghan en $y \in x$, dan $y \in \mathcal{P}(\Sigma^*)$.

Reghan $\subset \mathcal{P}(\Sigma^*)$ = alle talen
↳ vormt een algebra voor operaties als: concatenatie, unie, $\cap$, $-$, $*$, $L_1 \setminus L_2$ (verschil), $\wedge$ (omkering: $s = a_1 \dots a_m \to \hat{s} = a_m \dots a_1$), shift ($s_1 = a_2 \dots a_m a_1$)

Is Reglan een subalgebra?



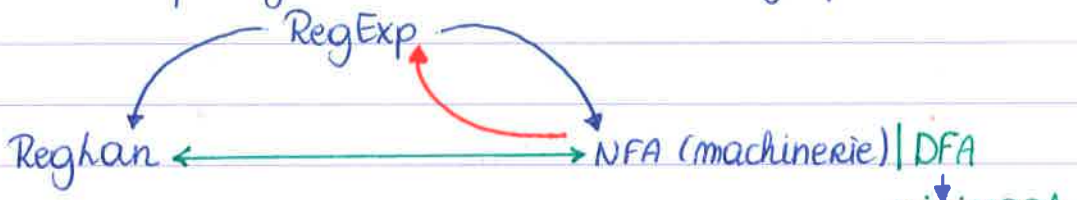
L_1 en L_2 $\in$ Reglan met $L_1 = L_{E_1}$ en $L_2 = L_{E_2}$

- $L_1 L_2 \in$ Reglan, want RegExp: $(E_1 E_2)$
- $L_1 \cup L_2 \in$ Reglan, want RegExp: $(E_1 | E_2)$
- $L_1^* \in$ Reglan, want RegExp: $(E_1)^*$
- $\overline{L_1} \in$ Reglan → geen constructiemethode
- $L_1 \cap L_2 \in$ Reglan

→ Hoe bewijs je dat $L \in$ Reglan?

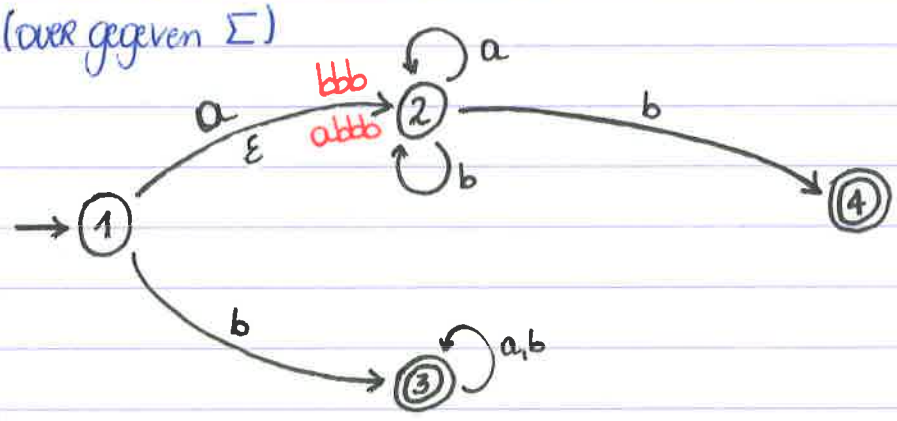
= Zoek of bewijs het bestaan van een $E \in$ RegExp, zodat $L = L_E$.

(5/10/2010)



3. Eindige toestandsautomaten

(over gegeven Σ)



→ knopen van de graaf = toestanden

→ ① is de starttoestand

Je kan dit gebruiken om een string te genereren.

Je kan nagaan of de string door de machine wordt geaccepteerd.

= manier om een string te herkennen, maar hiermee zijn fouten mogelijk!

vb abbb wordt op minimaal 2 manieren aanvaard.

Strings die door de automaat worden opgebouwd
strings die herkend worden door de automaat

Bewijs het pad om een string op te bouwen
⇕
het pad om dezelfde string af te breken. ∎

Zo een machine noemen we een NFA (non-deterministic finite automaton)
↳ Σ eindig aantal toestanden

L_M = taal over Σ aanvaard door $M \subseteq (\Sigma^*)$

Hoe groot is de verzameling van talen aanvaard door een NFA?

~~eindig~~
oneindig < aftelbaar
~~overaftelbaar~~

$NFA = (Q, \Sigma, \delta, q_s, F)$
toestanden | start ∈ Q | eindtoestanden ∈ Q
alfabet

δ beschrijft de bogen : $\delta : Q \times Q : \Sigma \cup \{\varepsilon\}$ (RELATIE)
↳ Σ_ε

$\delta : Q \times Q \to \mathcal{P}(\Sigma_\varepsilon)$ = FUNCTIE

$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$

→ je kan een tabel van δ maken = TRANSITIETABEL.
grootte = $|Q|^2$

δ beschrijft ① —a/ε→ ②

$\delta^*(1, ab) = \delta(\delta(1,a), b)$ herhaaldelijk toepassen

$\delta^*(q_s, s) \cap F \neq \phi$ → s wordt geaccepteerd door de NFA.

→ voor vaste Q en vaste Σ : aantal NFA = eindig
⇒ unie van eindig mogelijke Q : aftelbaar.

Een string s van Σ^* wordt aanvaard door NFA $(Q, \Sigma, \delta, q_s, F)$
↔ $s = a_1 \dots a_m$ met $a_i \in \Sigma_\varepsilon$.
∃ toestand $q_j \in Q$
$q_0 = q_s$ en $q_m \in F$
$a_{i+1} \in \delta(q_i, q_{i+1})$

NFA ~ RegExp algebra - concatenatie
~ Reghan algebra

Een NFA vormt ook een algebra.

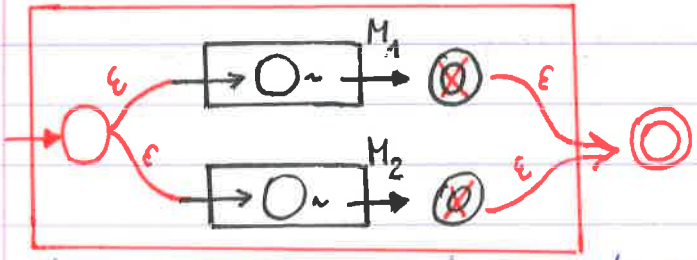
① Concatenatie

Voor elke NFA M_1 bestaat een NFA M_2 zodat $L_{M_1} = L_{M_2}$ en M_2 heeft 1 eindtoestand

$L_{CONCAT(M_1,M_2)} = L_{M_1} L_{M_2}$ → ALGEBRAÏSCHE OVEREENKOMST

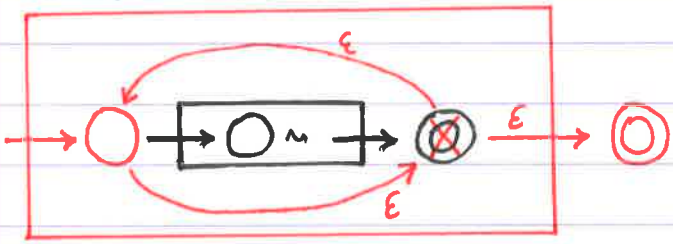
vb $s = yx$ met $y \in L_{M_1}$ en $x \in L_{M_2}$.

② Unie



$L_{M_1|M_2} = L_{M_1} \cup L_{M_2}$ $\quad L_{(E_1|E_2)} = L_{E_1} \cup L_{E_2}$

③ *-operatie



↳ kleene sluiting

NFA M ⟶ RegExp E

$L_M = L_E$

Variant van NFA : GNFA

NFA ∈ GNFA

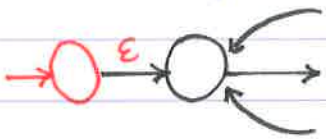
↘ NFA' ↗

→ op boog Σ_ε en RegExp

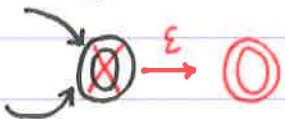
→ 1 begintoestand, 1 eindtoestand

→ Er zijn bogen van en naar elke knoop (behalve begin- en eindknoop)

* enkel uit q_s vertrekken

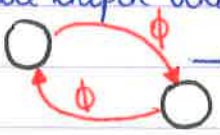


* q_f



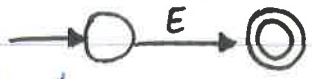
10.

* alle andere knopen verbonden in 2 richtingen



→ deze bogen kan je eigenlijk niet nemen

→ Een GNFA met slechts 2 toestanden



alle strings die voldoen aan de RegExp E, worden aanvaard.

$L_M = L_E$

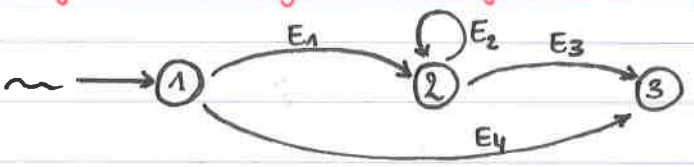
NFA → GNFA → GNFA met slechts 2 toestanden → E

(*) bij elk van deze stappen blijft de taal behouden.

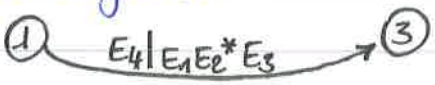
knopen 1 voor 1 verwijderen

Hoe kan je een inwendige knoop weghalen terwijl de taal hetzelfde blijft?

Vb

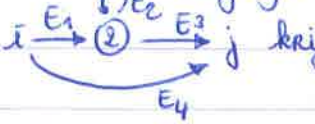


2 knopen weghalen:



→ dit voor alle buren van ② tegelijk doen!

→ Voor alle knopen $i, j \neq ②$ in 1 stap



krijg je $i \xrightarrow{E_4|E_1E_2^*E_3} j$

$L_M = L_{M \setminus ②}$

ALGORITME NFA → RegExp

① NFA → GNFA

② verwijder inwendige knopen tot 0

③ output: $\bigcirc \xrightarrow{E} \textcircled{\bigcirc}$

CORRECTHEID

EINDIGHEID

COMPLEXITEIT $m+2$ knopen

kost om 1 knoop te verwijderen: $O(m^2)$ aanpassingen nodig

$O(m)$ keer

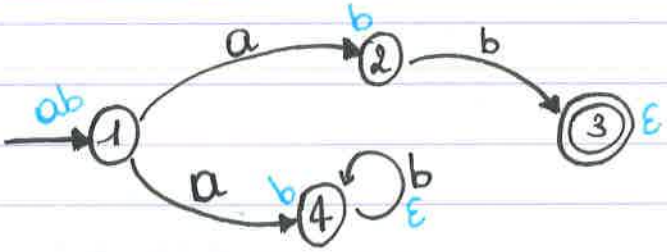
$= O(m^3)$

16/10/2010) # 4. Deterministische eindige automaat DFA

$DFA = (Q, \Sigma, \delta, q_s, F)$

↳ $Q \times \Sigma \to Q$

Als je een DFA-machine hebt: $L_M \in$ Reghan.
$\forall L \in$ Reghan: $\exists M \in$ DFA: $L_M = L$.



→ alle keuzes die je op dat ogenblik kon maken bijhouden.

NFA	DFA
Q	$\mathcal{P}(Q)$
n toestanden	2^n toestanden
$\lvert Q \rvert = n$	$\delta_d : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$
	geen ε?

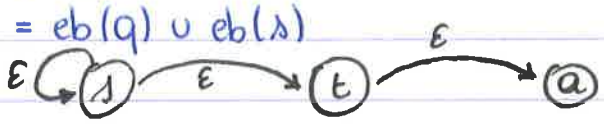
$\begin{cases} q_{s,n} \to q_{s,d} \\ F_n \to F_d \end{cases}$

→ We definiëren de functie eb voor een gegeven $NFA(Q_n, \Sigma, \delta_n, q_{p_n}, F_n)$

$eb: Q_n \to \mathcal{P}(Q_n)$

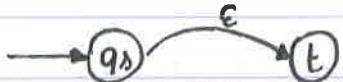
$eb: \mathcal{P}(Q_n) \to \mathcal{P}(Q_n)$

$eb(\{q, s\}) = eb(q) \cup eb(s)$



$\Rightarrow q_{s_d} = eb(q_{s_n})$

NFA:



DFA: → $\{q_s, t\}$

$\Rightarrow F_d =$ deelverzameling van $Q_n \cap F_n \neq \phi$

$\delta_d : \mathcal{P}(Q_n) \times \Sigma \to \mathcal{P}(Q_n)$ en $Q_d = \mathcal{P}(Q_n)$

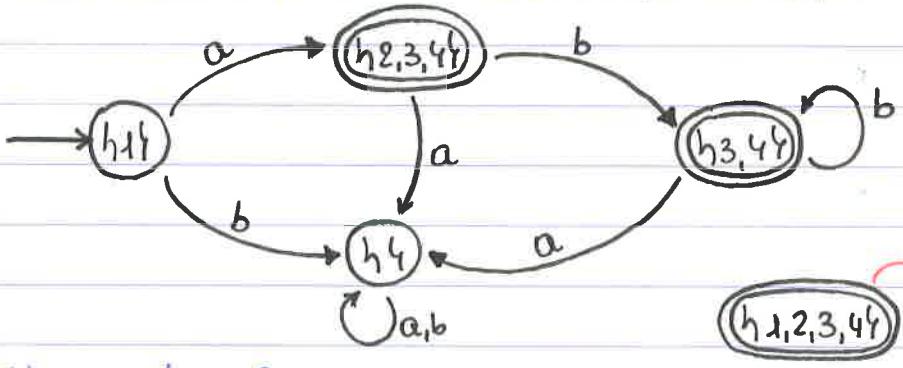
$\delta_{d_j}(\{q_0 \dots q_j\}, a) = eb\left(\bigcup_{i=1} \delta_n(q_i, a)\right)$

$\delta_d(\{1\}, b) = eb(\delta_n(\{1\}, b)) = eb(\{2\}) = \{2\} \in \mathcal{P}(Q_n)$

Hoeveel toestanden zijn er in Q_d?

$Q_d = 2^{|Q_m|} \; (= 2^4 = 16)$

$q_{s,d} = eb(\{1\}) = \{1\}$ $\qquad$ $S_d(\{1\}, a) = \{2,3,4\}$



We hebben een transformatie van NFA → DFA die de taal behoudt.

⊕ geen keuze

⊖ veel meer toestanden

→ Complexiteit van het herkennen van een string s met lengte $|s|$

NFA	DFA
$O(2^{|s|})$	$O(|s|)$
Q	$2^{|Q|}$ komt in praktijk weinig voor

We moeten $2^{|Q|}$ keer δ_d berekenen $(+\Sigma, \delta_m, eb)$

$\Rightarrow O(2^{|Q|}|Q||\Sigma|)$. $\qquad$ $|\Sigma|$ 1 $|Q|$

Toch maken veel tools de overgang van NFA → DFA. Dit is om achteraf tijd te winnen (zeker als de DFA vaak gebruikt wordt).

NFA → DFA

explosie $|Q|$

aantal toestanden DFA ↓

Minimale DFA M_2

- $L_{M_1} = L_{M_2}$: M_1 en M_2 zijn equivalent
- M_2 heeft een minimaal aantal toestanden.

Bestaat zo een M_2?

Ja. Er is altijd een machine met een minimaal aantal toestanden

Is M_2 uniek?

Nee. Twee DFA's kunnen hetzelfde uitzien, maar niet hetzelfde zijn

vb → (1) a

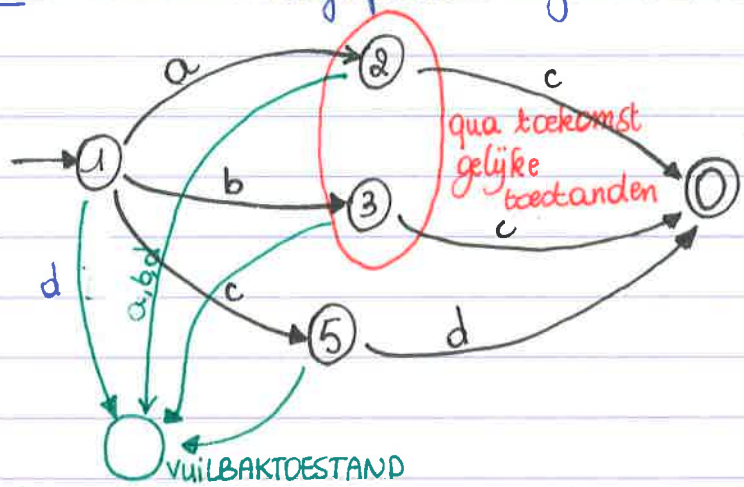
→ (2) a

} niet hetzelfde!

Isomorfe DFA: de toestand van een machine is een hernoeming van de andere, maar de overgangen blijven gelijk.

$M_1(Q_1, \Sigma, \delta_1, q_{s_1}, F_1)$ is isomorf met $M_2(Q_2, \Sigma, \delta_2, q_{s_2}, F_2)$
indien: $b: Q_1 \to Q_2$ met b een bijectie
$\delta_2(b(q), a) = b(\delta_1(q, a))$.

DFA: toestanden kwijtspelen = weghalen van onbereikbare toestanden



⚠ De machine weet niet hoe je in een toestand bent geraakt.

② en ③ noemt met f-gelijke toestanden

Voor elke string s: $\delta^*(②, s) \in F \Leftrightarrow \delta^*(③, s) \in F$

ALGORITME

① Zoek een verzameling van f-gelijke toestanden

② Neem samen

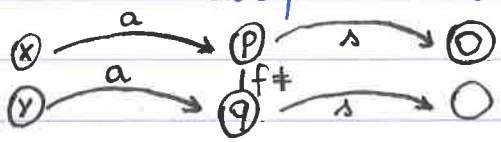
{alle toestanden zijn f-verschillend}
{alle toestanden zijn bereikbaar}

p, q zijn f-verschillend $\equiv \exists s: \delta^*(p, s) \in F$ en $\delta^*(q, s) \notin F$ of omgekeerd

$p \in F$, $q \notin F$ } p is f-verschillend van q

initialisatie $P \leftarrow \{(p,q) \mid p \in F, q \notin F\}$
L: alle f-verschillen

Lus:

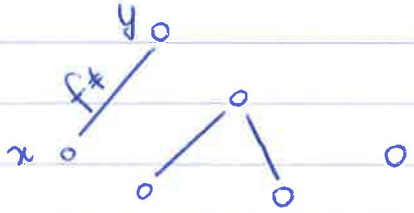


} ⇒ x en y zijn ook f-verschillend

→ zoek (x, y) en $a \in \Sigma$: $\delta(x,a)$ f≠ $\delta(y,a)$ en $(\delta(x,a), \delta(y,a)) \in P$
→ $(x, y) \to P \in P$ tot er geen (x, y) zijn

Na lus: P van f-verschillende toestanden (x, y) = graaf met Q knopen

14.



f-gelijk is een transitieve RELATIE

Levert dit een minimale DFA?

(12/10/2010) Uitkomst van het algoritme:

- elke toestand is bereikbaar vanuit q_s

  $\forall q \in Q, \exists s \in \Sigma^* : \delta^*(q_s, s) = q$

- elke 2 toestanden zijn f-verschillend

  $\forall p,q \in Q, p \neq q, \exists s \in \Sigma^* : \delta^*(p,s) \notin F$ en $\delta(q,s) \in F$ (of omgekeerd)

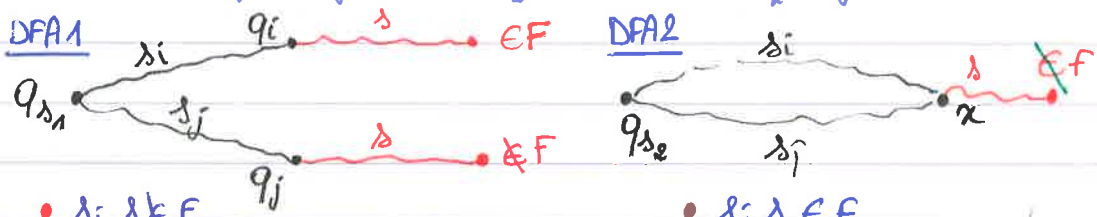
DFA_1 met: • alle toestanden zijn bereikbaar
• $\forall p,q \in Q$: p is f-verschillend van q
dan bestaat er geen DFA met minder toestanden voor dezelfde taal

Bewijs stel: DFA_2 met $|Q_2| < |Q_1|$

voor $q_i \in Q$: s_i : $\delta_1^*(q_{s_1}, s_i) = q_i$

$\delta_2^*(q_{s_2}, s_i) \in Q_2$

$\Rightarrow \exists i,j, i \neq j : \delta_2^*(q_{s_2}, s_i) = \delta_2^*(q_{s_2}, s_j)$



- $s_j s \notin F$
- $s_i s \in F$

- $s_j s \in F$
- $s_i s \notin F$

$\Rightarrow DFA_1$ en DFA_2 bepalen niet dezelfde taal ∎

5. Algebra van Reguliere talen

→ COMPLEMENT van $L \in Reglan \Rightarrow \exists DFA\ (Q, \Sigma, \delta, q_s, F)$

$\Rightarrow \overline{DFA} = (Q, \Sigma, \delta, q_s, \bar{F})$

$\bar{L} \in Reglan$

De doorsnede van reguliere talen is ook regulier!

$L_1 \cap L_2 = \overline{\bar{L_1} \cup \bar{L_2}}$

DFA_1 voor L_1 : $(Q_1, \Sigma, \delta_1, q_{s_1}, F_1)$
DFA_2 voor L_2 : $(Q_2, \Sigma, \delta_2, q_{s_2}, F_2)$
x
DFA : $(Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, q_{s_1} \times q_{s_2}, F)$

$(\delta_1 \times \delta_2)(q_1 \times q_2, a) = \delta_1(q_1, a) \times \delta_2(q_2, a)$

① $F = F_1 \times F_2$

$\delta^*(q_{s_1} \times q_{s_2}, s) \in F$: $\delta_1^*(q_{s_1}, s) \in F_1$
$\delta_2^*(q_{s_2}, s) \in F_2$

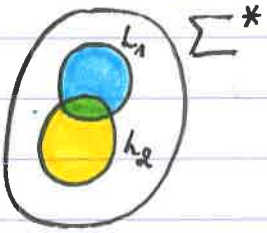
→ $L_1 \cap L_2$

② $F = F_1 \times Q_2 \cup F_2 \times Q_1$

→ $L_1 \cup L_2$

③ $F = F_1 \times \overline{F_2}$

→ $L_1 \cap \overline{L_2} = L_1 \setminus L_2$



NFA : $(Q, \Sigma, \delta, q_s, F) : L \Rightarrow \exists$ pad : $q_s \xrightarrow{s} F$
$(Q, \Sigma, \delta, q_s, \overline{F}) : \overline{L}$ ~~$\forall$ paden $q_s \xrightarrow{s} F$~~
! ZO WERKT EEN NFA NIET

→ Andere semantiek wegens niet-determinisme!

Complexiteit om een DFA te minimaliseren = polynomiaal
↕ ↕
complexiteit om een NFA te minimaliseren = NP-hard

6. Myhill - Nerode

Partities van een verzameling V en equivalentierelaties op V.
↳ opdeling van $V = \{a, b, c, d\}$ in disjuncte stukken die samen V vormen
vb $\{\{a, b\}, \{c, d\}\}$ zonder ϕ.

→ partitie $\{V_1 \dots V_k\}$ $V_i \cap V_j = \phi$ $i \neq j$
$\cup V_i = V$

→ equivalentierelatie op V : $\sim_p$: $x, y \in V$; dan $x \sim_p y$
$\Leftrightarrow \exists i : x, y \in V_i$

OMGEKEERD: gegeven $\sim$ op V
↳ equivalentierelatie

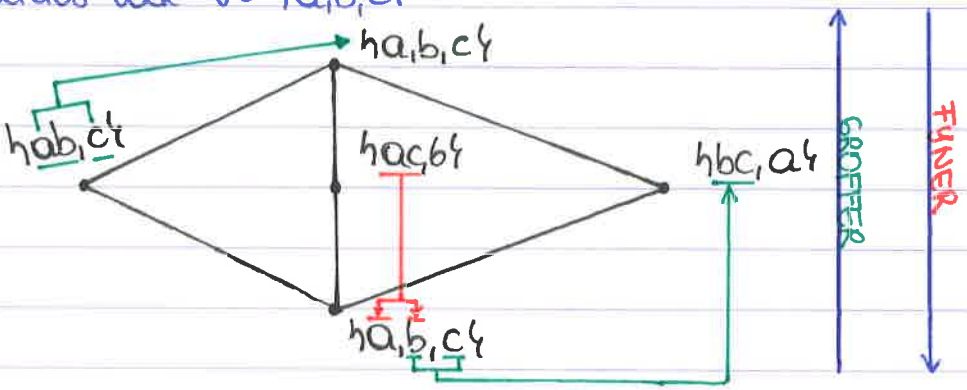
$\Rightarrow p_\sim = \{V_1 \dots\}$

$x \in V$: $\{y \mid y \sim x\} = V_i$

→ alle partities van V zitten in traliet T = PARTIEEL GEORDENDE RELATIE
↳ orde gegeven door "GROFFER"

→ Hesse-diagram

vb partities van $V = \{a,b,c\}$



2 partities: p_1 en p_2: in het diagram kunnen we naar het SUPREEM zoeken.

supreem (p_1, p_2) = fijnste die groffer is dan p_1 en p_2.

Als $V = \Sigma^*$ dan zijn er niet-aftelbaar oneindig veel partities.

→ Gegeven een reguliere taal L en een DFA (waarbij dus alle toestanden bereikbaar zijn) voor L $(Q, \Sigma, \delta, q_s, F)$.

Voor $q \in Q$: $\{s \mid s \in \Sigma^*, \delta^*(q_s, s) = q\} = \text{reach}(q)$

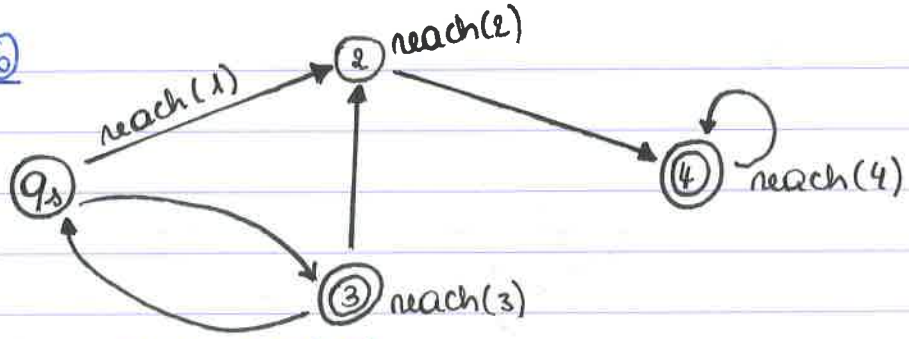
① $p \neq q \in Q \Rightarrow \text{reach}(p) \cap \text{reach}(q) = \phi$
② $\text{reach}(q) \subseteq \Sigma^*$
③ $\bigcup_{q \in Q} \text{reach}(q) = \Sigma^*$
→ $\{\text{reach}(q) \mid q \in Q\}$ = partitie van Σ^*
④ partitie is eindig: $\{V_1 \dots V_k\}$ met $V_k = \text{reach}(q_k)$.
⑤ $\sim_{DFA}$: 2 strings zijn gelijk als ze in dezelfde reach(q) zitten
→ RECHTSCONGRUENT: $s_1 \sim_{DFA} s_2 \Rightarrow \forall a \in \Sigma : s_1 a \sim_{DFA} s_2 a$
⑥ $\{\text{reach}(q)\}$ is fijner dan $\{L, \bar{L}\}$

Bewijs voor ⑤ $s_1 \sim_{DFA} s_2 \Rightarrow \delta^*(q_s, s_1) = \delta^*(q_s, s_2) = q$

$\delta(\delta^*(q_s, s_1), a) = \delta(\delta^*(q_s, s_2), a)$

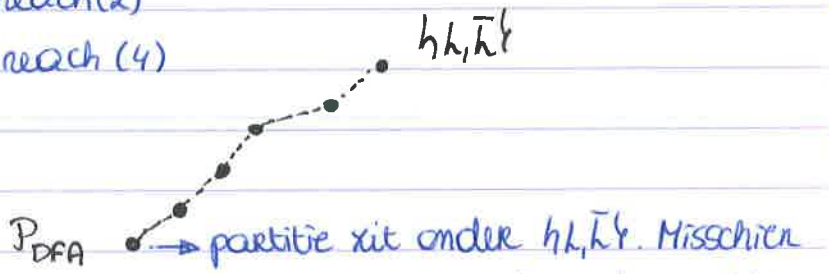
$\Rightarrow \delta^*(q_s, s_1 a) = \delta^*(q_s, s_2 a) \Rightarrow s_1 a \sim_{DFA} s_2 a$ ∎

uitleg bij ⑥



$\bar{L}$ = reach(1) ∪ reach(2)

L = reach(3) ∪ reach(4)

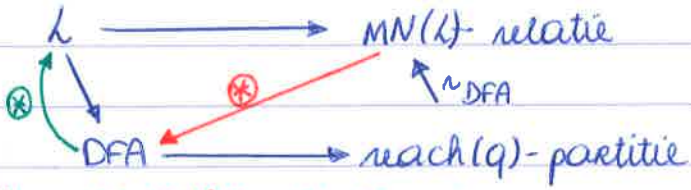


→ partitie zit onder $\{L, \bar{L}\}$. Misschien niet direct, maar de partitie is wel fijner

Myhill-Nerode-relatie over L

MN(L)-relatie op Σ^* is:

- $\sim$ is rechtscongruent: $s_1 \sim s_2 \Rightarrow \forall a \in \Sigma : s_1 a \sim s_2 a$
- $\sim$ heeft een eindige index | de partitie is eindig
- $\sim$ verfijnt $\{L, \bar{L}\}$



Hoeveel MN(L)-relaties zijn er voor een gegeven taal?

~~eindig~~
oneindig < aftelbaar
~~onaftelbaar~~

→ DFA met n toestanden → partities met n elementen
+ aantal DFA's = aftelbaar = aantal MN(L)-relaties

⊛ Gegeven: een bepaalde relatie MN(L) op Σ^*
→ Construeer een DFA voor L: $(Q, \Sigma, \delta, q_s, F)$.

- Q = partitie van MN(L)-relatie $\sim$ = EINDIG
- q_s is de equivalentieklasse die ε bevat

- F: $q \subseteq L \Rightarrow q \in F$
- $\delta : Q \times \Sigma \to Q : \delta(q, a) = p$ we nemen een element x uit q
  $\Rightarrow$ beschouw $xa \in p$.

⊛ $\delta^*(q_s, s) \in F \Leftrightarrow s \in L$

DFA_1 voor L ⟶ $MN(L)$ ↓

$MN(L)$ ↓ ⟷ DFA_2 voor L (niet IDENTIEK, wel ISOMORF!)

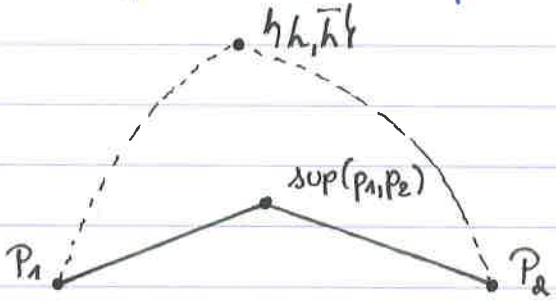
(13|10|2010) **stel**: DFA_1 en DFA_2 die niet isomorf zijn

$|Q_1| = |Q_2|$ = MINIMAAL | L.

→ $MN(L)_1 \sim 1$ $\quad$ $MN(L)_2 \sim 2$

↓ $\quad$ ↓

$|\text{partitie}_1| = |Q_1| = |\text{partitie}_2|$



- $P_1, P_2 = MN(L)$, dan is het $\sup(P_1, P_2)$ dat ook
- $|P_1| \geq |\sup(P_1, P_2)|$
- $|P_2| \geq |\sup(P_1, P_2)|$
- $|Q_1| \geq |Q_{sup}|$

DFA_{sup} is ook bepaald voor L.

→ $|\sup(P_1, P_2)| = |P_1| = |P_2|$

⇒ $P_1 \equiv P_2 \equiv \sup(P_1, P_2)$

$P_1 = \{V_1 \dots V_n\}$ $\quad$ $V_i \cap V_j = \phi$ voor $i \neq j$

$P_2 = \{W_1 \dots W_k\}$ $\quad$ $\cup V_i = V$

$\sup(P_1, P_2) = \{K_1 \dots K_m\}$ → moet groffer zijn dan P_1 en P_2

$\forall V_i : \exists K_j : V_i \subseteq K_j$

$\forall W_i : \exists K_j : W_i \subseteq K_j$

vb $P_1 = \{\{a\}, \{b,c\}, \{d\}\}$

$P_2 = \{\{a,b\}, \{c,d\}\}$

→ $\{\{a,\underline{b}\}, \{\underline{b}, c, d\}\}$ OVERLAPPEN

→ $\{a,b,c,d\}$

$P_1 \sim_1$

$P_2 \sim_2$

$x \sim_{sup} y$ indien $x \sim_1 y$ of $x \sim_2 y$ en transitieve sluiting

⇒ $x \sim_{sup} y$ indien $x \sim_1 a_1$ en $a_1 \sim_2 a_3$ en $a_3 \sim_1 y$.

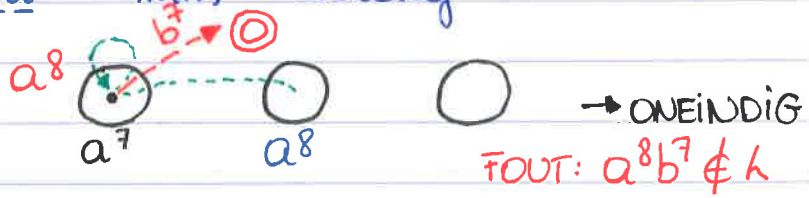
MN-stelling: **L is een reguliere taal a.s.a. $\exists$ MN(L).**

↳ gebruik: $\Sigma = \{a,b\}$

$L = \{a^m b^m \mid m \in \mathbb{N}\}$

→ voor L bestaat geen MN(L)-relatie:

stel $\exists$ MN(L) → eindig



FOUT: $a^8b^7 \notin L$

7. Complexiteit van het herkennen van een reguliere taal

L: algoritme $\begin{cases} s \in L & \text{ja} \\ s \notin L & \text{nee} \end{cases}$ $= O(n^3)$

Als je een DFA gebruikt zal je in het slechtste geval heel de string afgaan. $= O(n \log|Q|) \approx O(n)$

↳ Om 2 toestanden te kunnen vergelijken (elke toestand die wordt voorgesteld kost tijd).

Hiërarchie van Chomsky

	MACHINE	COMPLEXITEIT
Reghan	DFA / NFA	$O(n)$.
context free		
context sensitive		
willekeurig		

8. Pompend lemma voor reguliere talen

$L \in$ Reghan, dan bestaat er altijd een minimale pomplengte l: $\forall s \in L: |s| > l$ en $\exists x,y,z: s = xyz$ met $|y| > 0$ zodat $xy^iz \in L$ voor $i \in \mathbb{N}$

Is elke reguliere taal oneindig?

Nee: $\Sigma = \{a\}$ en $L = \{aaa\}$ → eindig en regulier.

ELKE EINDIGE TAAL IS REGULIER.

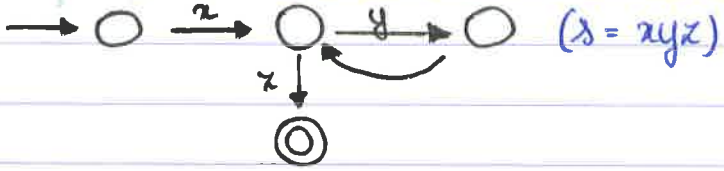
Bewijs van het pompend lemma :

gegeven : $L \in$ Reglan

→ Neem een DFA voor L : $|Q|$

Neem string $|s| > |Q|$ met $s \in L$

⇒ $|s|+1$ toestanden om s te herkennen



⇒ xy^iz wordt herkend. ∎

Gebruik van het pompend lemma voor reguliere talen

- ~~L is regulier?~~

⚠ Het lemma zegt niet dat alle strings kunnen gepompd worden

⇒ L = regulier

Sommige niet-reguliere talen kunnen gepompd worden.

- L is niet-regulier

$L = \{a^m b^m \mid m \in \mathbb{N}\}$

→ $\neg \exists p : \forall |s| > p$ met $s = xyz$ en $xy^iz \in L$:

$\forall p : \exists |s| > p$ en $\forall$ opdelingen $s = xyz$

$\exists i : xy^iz \notin L$.

vb stel: p = pomplengte

$s = a^p b^p \rightsquigarrow xyz$ met $|y| \neq 0$

→ $y = a^i$ of $y = a^i b^i$ of $y = b^i$

⇒ gelijk wat je pompt, je vindt geen geaccepteerde string.

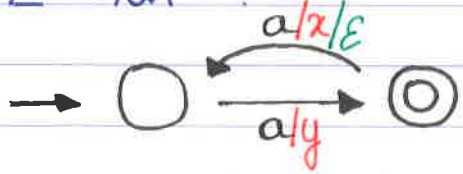
⇒ $L = \{a^m b^m \mid m \in \mathbb{N}\}$ is niet-regulier

(19/10/2010) 9. Varianten van DFA

---

① Transducer

vb $\Sigma = \{a\}$



= accepteert een oneven aantal a's.

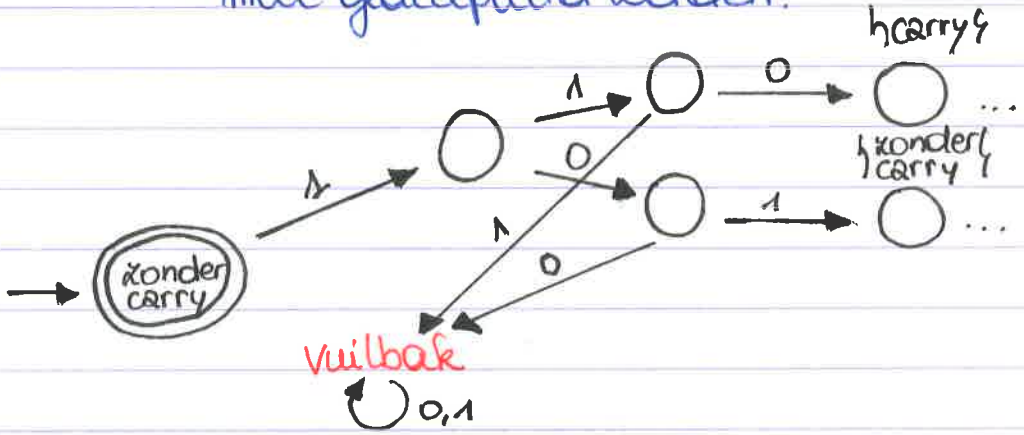
aaa $\xrightarrow{\text{output}}$ yxy

→ Er bestaat een output voor elke geaccepteerde input
↳ symbolen op de gevolgde bogen.

→ Is deze outputtaal ε Reghan?
Je mag ook ε als output nemen.
aaa $\xrightarrow{\text{output}}$ yy = een deling door 2.

② Optelling met een DFA

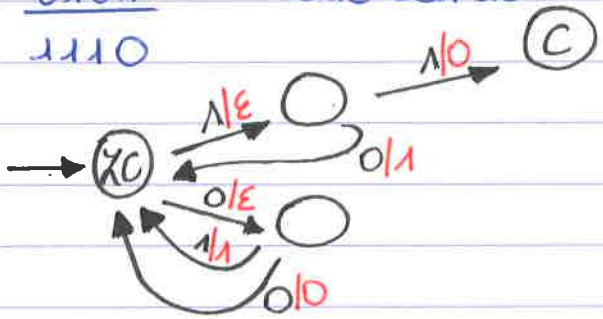
= manier om een DFA te gebruiken
→ nakijken of een optelling correct is $\frac{9}{5}$ ~~13~~ 14
→ Optelling uitvoeren
⇒ Output moet in een juist formaat gezet worden.
vb $\Sigma = \{0,1\}$ 1001
0101 zorgen dat alle getallen
1110 even lang zijn.
→ 110 001 011 101 = string in Σ^*
= moet geaccepteerd worden.



Essentieel heb je slechts een eindig geheugen nodig.
Je hebt niets van het verre geheugen van de optelder nodig.

③ Optelder met een transducer

1001
0101 110 001 10
1110



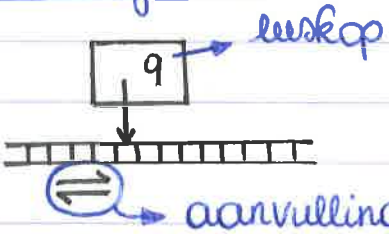
Kan men hiermee vermenigvuldigen ?

→ tussenbewerkingen met te onthouden resultaten
= $O(n^2)$ met n het aantal cijfers.

```
  ***
 ***
 ***
 ***
+***
```

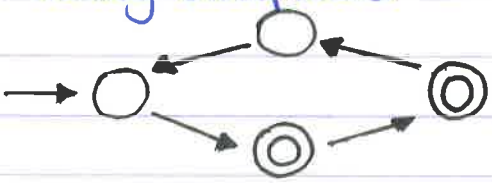
⇒ gaat niet !

④ Two-way DFA



→ aanvulling van de transitietabel : $\delta \times q \times \{L, R\}$

⑤ Büchi-automaat — deterministisch
∼ DFA — niet-deterministisch
strings = $\omega \in \Sigma^*$
ω strings = ω proces
→ string accepteerd



string s wordt geaccepteerd door een Büchi-automaat als s oneindig vaak een ◎-knoop passeert.
$s = s_1 s_2$

10. Reguliere expressies en lexicale

RegExp = specificatie van het lexicon van een programmeertaal.
vb getal : (1|...|9)(0|...|9)*|0
id : (a|...|z)(a...z|0...9)*
afkortingen vb digit → (0|...|9), letter → (a|...|z)
number → digit (digit)*
id → letter (letter|digit)*

RegExp → NFA → DFA → minimaal
(tools jflex)

javaprogramma

Grammatica voor term

term → term ① + term | term ② - term | int ⑤
int → dig + ③
dig → 0|...|9 ④

vb 3 is een term
7+5-3 is een term

Hoe maak je een nieuwe term?

term → ② term - term → ① term - term + term
→ ③,④ term - int + term → * 17 - 3 + 20

Afleiding van een string uit grammatica (V, Σ, R, S)
→ reeks van een niet-eindsymbool / startsymbool
V: niet-eindsymbool, links, term int dig
Σ: eindsymbolen ± 0...9
R: niet-eindsymbool → (eindsymbool | niet-eindsymbool)*
↳ hieraan voldaan ⇒ CONTEXTVRIJ CFG
S: startsymbool : term

sommige strings s hebben een afleiding: $s \in L_{CFG}$
CFG: gebruiken om $s \in L_{CFG}$ te genereren / herkennen?

Bestaat er een contextvrije taal (CFL) die niet-regulier is?

$\Sigma = \{a,b\}$
$L = \{a^n b^n \mid n \in \mathbb{N}\}$
$L \notin$ Reglan
→ ja!

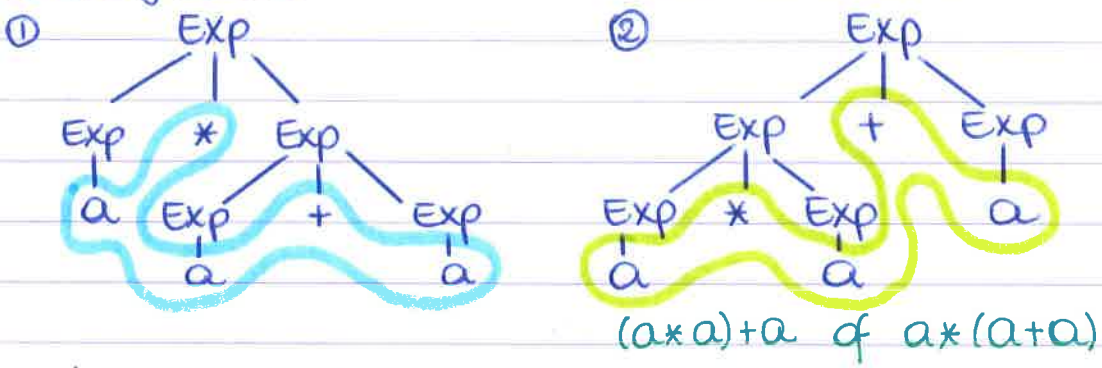
CFG = $(\{S\}, \{a,b\}, \{S \to aSb, S \to \varepsilon\}, S) = G$
L_G

① **Ambiguïteit**

Exp → Exp * Exp
Exp → Exp + Exp
Exp → a

vb • Exp → Exp * Exp → Exp * Exp + Exp → a * a + a ①
• Exp → Exp + Exp → Exp * Exp + Exp → a * a + a ②

Afleidingsboom



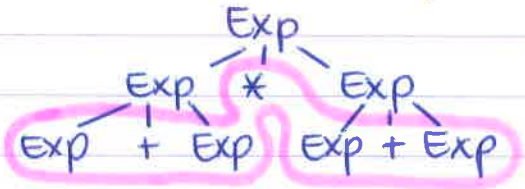
(a*a)+a of a*(a+a)

Voor 1 string: 2 of meer verschillende afleidingsbomen.
⇒ grammatica is ambigu.

Hoe kan je een grammatica niet-ambigu maken?

Exp → (Exp * Exp)
Exp → (Exp + Exp)
Exp → a

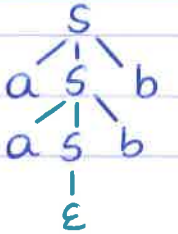
vb • Exp → (Exp * Exp) → ((Exp+Exp) * Exp)
→ ((Exp+Exp)*(Exp+Exp)) → ((a+a)*(a+a))
• Exp → (Exp * Exp) → (Exp * (Exp+Exp)) ↕ GELIJK
→ ((Exp+Exp)*(Exp+Exp)) → ((a+a)*(a+a))
→ 2 verschillende afleidingen, maar dezelfde afleidingsboom

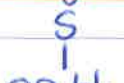


⇒ grammatica is niet-ambigu.
→ steeds meest linkse afleiding kiezen.

$G_1 \sim G_2$ (equivalent) $L_{G_1} = L_{G_2}$
$G \rightarrow L_G$
stel: G is ambigu : ∃s met 2 verschillende afleidingsbomen
soms: ∃G' voor L_G met G' niet-ambigu
als dit niet zo is: (∀G voor L_G zijn ambigu), dan
is L_G inherent ambigu.

vb S → aSb
S → ε
S → aabb





⚠ De meeste grammatica's voor programmeertalen zijn ambigu

② Grammatica's in speciale vorm

↳ Chomsky normal form.

$S \to AB \quad A,B \in V$

$S \to \alpha \quad \alpha \in \Sigma$

$S \to \varepsilon$ enige regel die ε bevat

Als G in Chomsky NF staat, dan $\varepsilon \in L_G$ en $S \to \varepsilon$ $\in$ regels.

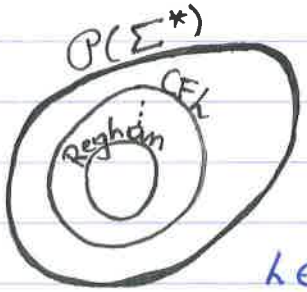
$s \in \Sigma^*$ } → je kent de lengte van de afleiding
$s \in L_G$ }

(26/10/2010) Contextvrije grammatica CFG = (V, T, R, S) met 1 term → $(V|T)^*$

→ De regels zelf vormen een taal die regulier is.

vb $S \to aSb$

$S \to \varepsilon$



$L \in Reghan \Rightarrow L \in CFL$

11. Chomsky normaalvorm (voor een CFG)

---

① $C \to AB$

② $A \to a, a \in T$

③ enige regel die ε afleidt : $S \to \varepsilon$

④ S staat nergens rechts

• Als een CFG in CNF staat: $\varepsilon \in L_{CFG}$ - beslisbaar

• Een afleiding van $s \in L_{CFG}$, $s \neq \varepsilon$: CFG is in CNF.

lengte van de afleiding = $2|s|-1$

lengte van de afleiding van $\varepsilon = 1$

ALGORITME $s \in L_{CNF}$: genereer alle $2|s|-1$ afleidingen → s afgeleid

Voor elke contextvrije grammatica BESTAAT een CFG' die equivalent is en in Chomsky normaalvorm staat.

Constructie :

④ Nieuwe start S' : $S' \to S$

③ Stel: $A \to \varepsilon$

zoek alle regels $B \to \alpha A \beta$

$\downarrow$

$B \to \alpha\beta$

→ wat met $B \to \alpha A \beta A \gamma$

$\downarrow$ ∀ combinaties van A's: vervang A door ε.

$B \to \alpha\beta\gamma$
$B \to \alpha A \beta\gamma$
$B \to \alpha\beta A\gamma$

} (door 1 $A \to \varepsilon$)

→ We bepalen nog steeds dezelfde taal

	elimineer $A \to \varepsilon$	elimineer $B \to \varepsilon$
$A \to \varepsilon$	X	
$A \to B$	$A \to B$	$A \not\to \varepsilon$
$B \to A$	$B \to \varepsilon$	

⇒ ε is weg

⇒ ③ is voldaan.

①en② regels van de vorm $A \to B$

⇒ elke regel is van de vorm : $A \to a$

$A \to (V|T)^{2+}$

$S \to \varepsilon$

vb $A \to aBcDe$

nieuwe niet-terminale: $X_a \to a$

$X_c \to c$

$X_e \to e$

⇒ $A \to X_a B X_c D X_e$

$A \to XBYDZ$

↳ nieuwe niet-terminale $T_1 \to BYDZ$

$T_2 \to YDZ$

$T_3 \to DZ$

⇒ ① en ② zijn voldaan!

Elk prologprogramma is equivalent met een programma waarbij elke clause 2 doelen heeft.

PROLOG ⟷ CONTEXTVRIJE GRAMMATICA

12. Pompend lemma voor contextvrije talen

→ 2 substeings vinden en tegelijk herhalen

$L \in$ Reghan → $L \in$ CFL

→ $\exists p : \forall s \in L : |s| > p$

⇒ $\alpha\beta\delta\gamma\eta$ zodat $\alpha\beta^i\delta\gamma^i\eta \in L$

⇒ $|\beta\delta| > 0$

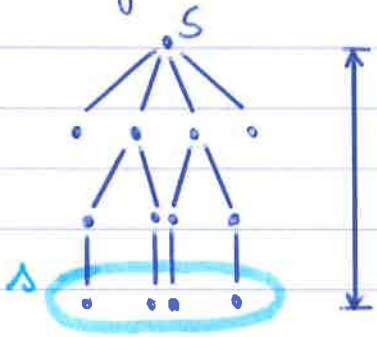
⚠ β en δ kunnen niet tegelijk de lege steing zijn.

→ We vermoeden dat Reghan ⊂ CFL

Het pompend lemma voor regulieren talen is sterker, van daaruit kan je namelijk het andere afleiden.

→ CFG = CNF

Voor steing s uit de taal ($s \in L$) maken we de afleidingsboom



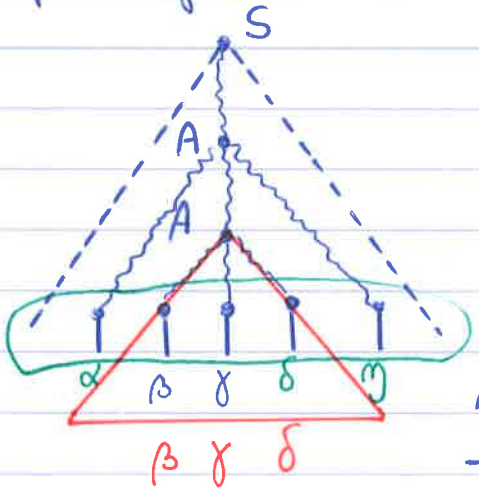
diepte ~ log $|s|$

afleiding = $2|s| - 1$

↳ zegt iets over het aantal talen in de boom

$|V|$ = aantal niet-terminalen

⇒ als diepte > $|V|$, dat bevat het langste pad van de afleidingsboom een A die herhaald wordt.



n keer een deelhoek maken:

→ $\alpha\beta^n\gamma\delta^n\eta$

↳ stelen geldige afleidingen voor.

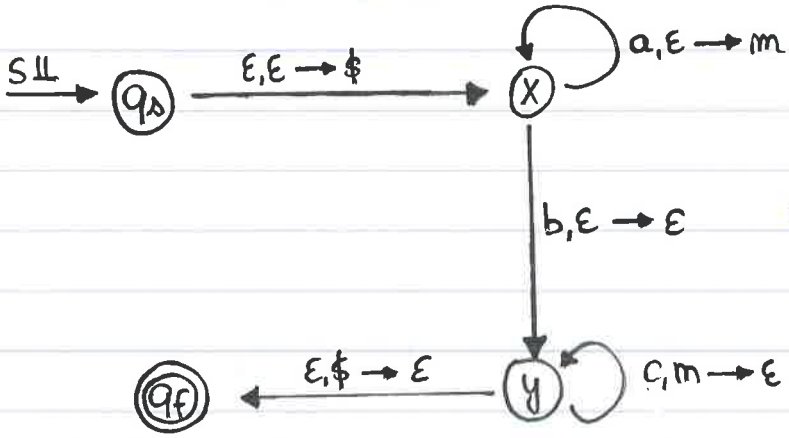
Omdat je nergens ε uit kan halen: $|\beta| > 0$

vb $\{a^m b^m c^m \mid m \in \mathbb{N}\} = L \subset \{a,b,c\}^*$

$a^m b^m c^m$

$\beta \quad \delta$

→ elke opdeling van de steing kan niet gebruikt worden om te pompen: Er bestaat minstens 1 taal buiten CFL.



→ deze automaat bepaalt de taal $a^m b c^m$.

stapel: initieel leeg

a, x → y

↳ consumeer a uit de steing (= begin van de steing)

VOORWAARDE: x staat nu op de top van de stapel

→ door dit te controleer kaal je hem

eraf: POP(STAPEL)

GEVOLG: voeg y toe: PUSH(STAPEL, y).

vb steing aabcc ∈ L

	stapel	toestand
aabcc	⊥	q_s
aabcc	⊥$	x
abcc	⊥$ m	x
bcc	⊥$ m m	x
cc	⊥$ m m	y
c	⊥$ m	y
ε	⊥$	y
ε	⊥	q_f

13. Push Down automaat (PDA)

Q = toestanden
$F \subseteq Q$ = finale toestanden
q_s = starttoestand
Σ = symbolen van strings
Γ = alfabet stapel
δ = overgangsfunctie (toont wat op de bogen staat)

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow Q \times \Gamma_\varepsilon$ (deterministisch)

$\rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ (niet-deterministisch)

↳ keuze mogelijk

Hoeveel PDA's zijn er?

gegeven: Σ en Γ

~~eindig~~
oneindig < aftelbaar
~~overaftelbaar~~

$|Q|$ ligt vast → eindige PDA

⇒ unie over alle mogelijke $|Q|$ = aftelbaar!

Wanneer aanvaardt een PDA een string s?

- Met lege string in F door δ te volgen.
- stapel? — maakt niet uit. Mag leeg zijn, maar moet niet
  moet leeg zijn, zolang je in F zit
  je moet niet in F zitten, als de stapel maar leeg is.

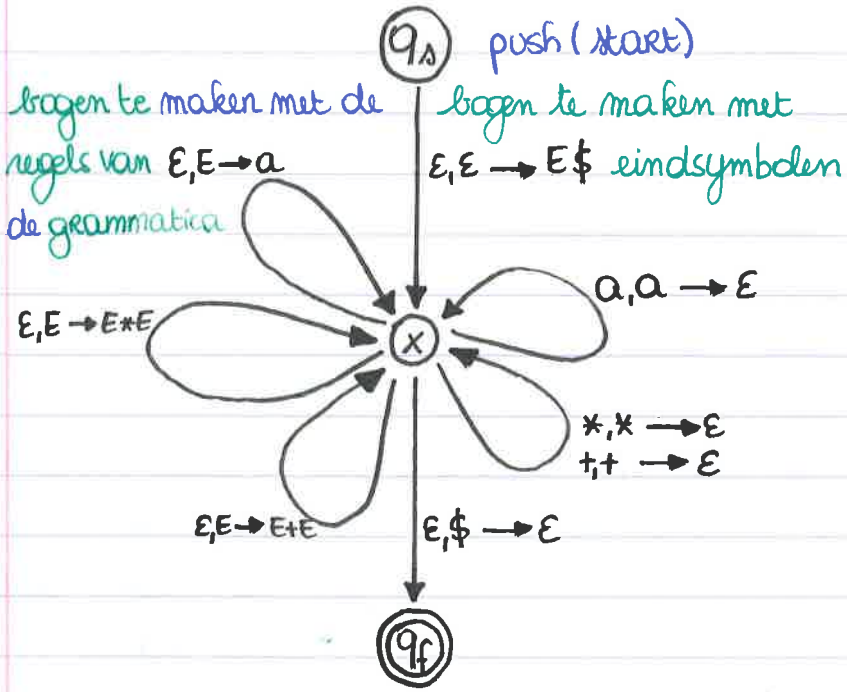
EQUIVALENT (maar noties vallen niet samen)

PDA ⟷ CFG

$\forall$ PDA: $\exists$ CFG : $L_{PDA} = L_{CFG}$ en omgekeerd

vb $E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow a$

$L_{CFG} = L_{PDA}$, want: afleiding is een pad in de PDA.



(27/10/2010)

14. Een algebra van contextvrije talen

Vormt de CFL een algebra ?

$L_1 \cup L_2 \in CFL$

$\begin{cases} S_1 \to \dots \end{cases}$ $\begin{cases} S_2 \to \vdots \end{cases}$ $S \to S_1$, $S \to S_2$

* $\begin{cases} S \to \end{cases}$ $S \to \varepsilon$, $S \to SS$

$L_1 L_2$

$\begin{cases} S_1 \to \dots \end{cases}$ $\begin{cases} S_2 \to \vdots \end{cases}$ $S \to S_1 S_2$

Complement/doorsnede $\quad \overline{\overline{A} \cup \overline{B}} = A \cap B$

$\exists L \in CFL \to \overline{L} \notin CFL$

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

$L = \{ss \mid s \in \Sigma^*\}$ → deze taal kan je niet pompen!

$\overline{L}$

⇒ Complement is geen inwendige operatie bij CFL, dus voor de doorsnede ook niet!

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

$L_1 = \{a^m b^m c^m \mid m, m, \dots\}$

$L_2 = \{a^m b^m c^n \mid m, m, \dots\}$

$L_1 \cap L_2 = \{a^m b^m c^m \mid m \in \mathbb{N}\}$

= niet-contextvrij, want je kan het niet pompen.

$\hat{s}$

als $s: a_1 \dots a_m$, dan is $\hat{s} = a_m \dots a_1$

$S \to AB$ = gewoon de volgorde van de strings omdraaien

$S \to BA$

= contextvrij

De unie is zeker een CFL, want: Regular is CFL.

Is de doorsnede regulier of contextvrij?

Regulier: zeker niet voor alle doorsnedes!

→ Als L_1 niet regulier is, kan de doorsnede het ook niet zijn

Contextvrij: product DFA: $Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, q_{s_1} \times q_{s_2}, F$

$F = F_1 \times F_2$

Nu heeft L_1 geen DFA!

⇒ product: PDA $(Q_1, \Sigma, \Gamma, \delta_1, F_1)$

DFA $(Q_2, \Sigma, \delta_2, F_2)$

$\delta_1: Q_1 \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q_1 \times \Gamma_\varepsilon)$

$\delta_2: Q_1 \times \varepsilon \to Q_2$

$\delta: (Q_1 \times Q_2) \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q_1 \times Q_2) \times \Gamma_\varepsilon$

Product van 2 stacks nemen is erg moeilijk

→ alsof je 2 stacks hebt die onafhankelijk van elkaar op en neer gaan

⇒ kan meer dan met 1 stapel

⇒ stack van een DFA is altijd leeg.

⇒ CONTEXTVRIJ!

15. Ambiguïteit en determinisme

---

① Determinisme van een PDA

CFG → PDA $E \to E+E$

$E \to E*E$

$E \to a$

niet-deterministisch

NFA ⟶ DFA

Deze constructie proberen uit te voeren op een PDA.

verteek eegens → kijk waar je kan uitkomen met het input symbool = 1 toestand

$(Q, \Sigma, \Gamma, \delta, F)$ = ND PDA

DPDA: $(\mathcal{P}(Q), \Sigma, \Gamma', \delta', \{P \mid P \in \mathcal{P}(Q), P \cap F \neq \phi\})$

↳ moeilijk vast te leggen.

$\delta' : \mathcal{P}(Q) \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(\mathcal{P}(Q) \times \Gamma_\varepsilon)$

$\delta' : (\{q_1, q_2\} \times a \times t)$ $\quad \delta(q_1 \times a \times t) = \{s_1, s_2\} \times \underline{\underline{p}}$

$\delta(q_2 \times a \times t) = \{x_1, x_2, x_3\} \times \underline{\underline{u}}$

→ 2 verschillende dingen om te pushen

⇒ geen stacksymbool voor

Je kan de constructie van NFA ⟶ DFA niet zomaar overnemen

⇒ Er is geen andere constructie!

Sommige CFL hebben geen DPDA!

= INHERENT NIET-DETERMINISME

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

$L = \{a^n b^n \mid \ldots\} \cup \{a^n b^{2n}\}$

↳ ook CFL? $S \to aSbb, S \to \varepsilon$, ja!

⇒ L is zeker een CFL!

Stel: PDA die die 2 aanvaardt en die deterministisch is.

$a^m b^m c^n \neq$ CFL

⇒ We hadden ze niet van mogen veronderstellen dat de 1ste taal een DPDA had!

→ $\{a^n b^m \mid \ldots\}$ is inherent niet-deterministisch.

De CFG G is ambigu als $\exists s$ met 2 verschillende afleidingsbomen.

$L \in$ CFL : L is ambigu als $\forall G$ voor L ambigu is

L is inherent ambigu.

Stel: je hebt een taal L die ambigu is.
kan die dan ook deterministisch zijn (of andersom)?
FA: ND = D ⟷ PDA: ND ≠ D

allebei zijn ze niet leeg en voor de reguliere talen vallen ze samen.

Verband tussen afleiding en PDA

in de afleiding is keuze mogelijk
→ moet in die PDA ook zijn
⇒ kan dus niet-deterministisch zijn!

RegExp ⟶ DFA_{min}
jflex compiler: • lexicon RE
• parsing/syntax (vb haakjes) CFG
• stat ⟶ if (condition) then statement;
stat ⟶ if (condition) then statement;
else statement;
stat ⟶ (statement)*

antlee

16. Contextsensitieve grammatica

aXb ⟶ aDfGb
aXa ⟶ aXbXa
→ Context in rekening brengen. Hoeft niet hetzelfde te zijn als je in een andere situatie zit!
→ CFL ⊂ contextsensitieve taal

Chomsky-hiërarchie

Reg	RegExp	FSA
CFL	CFG	PDA
CSL	CSG	LBA ⊛
...	unrestricted grammatica	TM

⊛ lineair begrensde automaat!

34.

Hoe zit het met algebraïsche eigenschappen, determinisme en complexiteit?

Regulier	$O(n)$
CFL	$O(n^3)$
CSL	constante ruimte
TM	unrestricted

- polynoom
- exponentieel
- alles wat slechter is.

HOOFDSTUK 2 : Talen en berekenbaarheid

1. De Turingmachine als herkenner en beslisser

$(Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$ — q_a: accept, q_r: reject

Σ: alfabet voor input (# & ε)

Γ: tapealfabet → dingen die geschreven kunnen worden ($\# \in \Gamma$)

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

$s \in \Sigma^*$ wordt aanvaard door TM $\quad \delta^*(s, q_s) = q_a$.

Machine M: $L_M = \{ s \in \Sigma^* \mid s \text{ wordt aanvaard door M} \}$

M herkent L_M — geef M $s \notin L_M$ → stop in q_r : nee; $s \in L_M$ → ∞ : stopt niet

→ M herkent $\overline{L_M}$ niet

Een taal L is herkenbaar als er een touringmachine M bestaat zodat $L = L_M$.

Hoeveel turingmachines zijn er?

Q, Σ en Γ liggen vast ⇒ EINDIG

→ Unie

⇒ AFTELBAAR ONEINDIG veel turingmachines

Hoeveel herkenbare talen zijn er?

∀ herkenbare taal : ∃ TM ⇒ zeker niet meer herkenbare talen dan er TM's zijn

⇒ AFTELBAAR ONEINDIG veel.

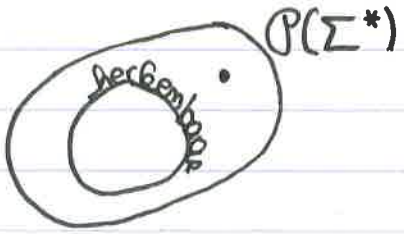
Alles dat regulier is → turingmachine die de DFA herkent → ONEINDIG AFTELBAAR veel

Hoeveel talen zijn er?

$\mathcal{P}(\Sigma^*)$ met Σ^* oneindig aftelbaar

⇒ ONEINDIG OVERAFTELBAAR veel

impliciet geldt: er bestaat een niet-herkenbare taal



De meeste talen zijn zelfs niet herkenbaar.

Een taal L is beslisbaar als er een turingmachine M bestaat zodat $L = L_M$ en M stopt altijd.

$s \in \overline{L_M}$ ~~$\not\in$~~ ~~∞~~ $\xrightarrow{q_r}$ M beslist L

Als L beslisbaar is, is L herkenbaar.
Als L beslisbaar is, is $\overline{L}$ beslisbaar. ($= q_a$ en q_r van plaats wisselen)
Als L herkenbaar is en $\overline{L}$ ook, dan is L beslisbaar.

Constructie

L herkent M_L
$\overline{L}$ herkent $M_{\overline{L}}$
→ Maak een beslisser voor L: L laat M_L en $M_{\overline{L}}$ parallel lopen
= stopt altijd en het antwoord kan je gebruiken om de taal L te beslissen.
→ Minstens 1 van beide machines (mogelijk ook allebei) stoppen, maar dit moet niet. De eerste die stopt, zegt wat je moet weten.

* Unie van 2 herkenbare talen

$L_1 - M_1$ $\quad s \to M_1 // M_2 \quad a_1 / a_2$
$L_2 - M_2$
$L_1 \cap L_2$

M_1	M_2
$q_1 a$	$q_2 s$
accept	start

⇒ HERKENBAAR!

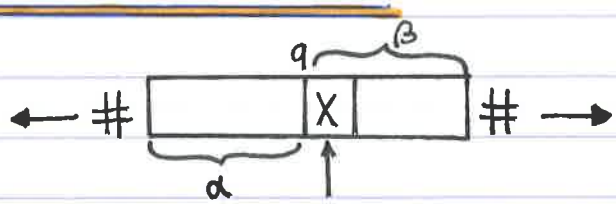
Je mag overal herkenbaar vervangen door beslisbaar!

(3/11/2016)

(p. 88) • Elke string is herkenbaar : $\forall s \in \Sigma^* : s \in$ herkenbare talen $\subseteq \mathcal{P}(\Sigma^*)$
• Elk string is beslisbaar :
⇒ beide uitspraken zijn fout ! $s \subseteq \mathcal{P}(\Sigma^*)$
WEL CORRECT : $\forall s \in \Sigma^* : \{s\} \in$ herkenbare talen en $\{s\} \in$ Reglan.

2. Berekening door TM-voorstelling

configuratie :



$\alpha q \beta \xrightarrow{\delta} \alpha' q' \beta'$

$\delta(q,a) = q' \times a' \times R$ stel : $\beta = \alpha\gamma$: $\alpha q \beta \rightarrow \alpha a' q' \gamma$ ⊕ minstens 1 rijmbad

strings die configuraties voorstellen : $(\Sigma \cup \Gamma)^* Q (\Sigma \cup \Gamma \cup \{\#\})$

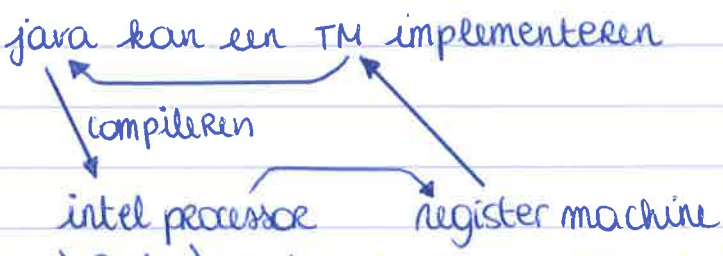
rij van configuraties : $C_1 \$ C_2 \$ C_3 \ldots \$ C_m$ = COMPUTATION HISTORY TM

$C_1 = q_s i$
$C_j \xrightarrow{\delta} C_{j+1}$

EINDIG : $C_m = \alpha q_\alpha \beta$: i aanvaard
$C_m = \alpha q_m \beta$: i niet aanvaard
geen einde (TM in lus)

De verzameling van eindige computation history strings is beslisbaar.

Bewijs : controleer of $C_1 = q_s i$ en of elke $C_j \xrightarrow{\delta} C_{j+1}$
⇒ BEREKENBAAR



Java \ Pedag \ ... (bijna elke programmeertaal) is Turing Compleet
NIET vb SQL (= niet recursief)
Er zijn ook talen die garanderen dat elk programma stopt. vb Datalog
↳ kunnen niet Turing Compleet zijn

3. Niet-deterministische turingmachine

$\delta : Q \times (\Sigma \cup \Gamma) \rightarrow \mathcal{P}(Q \times (\Sigma \cup \Gamma) \times \{L, S, R\})$

Geeft dit meer kracht dan determinisme ?

Nee : in java kan je een NDTM simuleren, je helt er computationeel niet meer aan.

4. Eigenschappen en talen

$\begin{cases} L \subseteq \Sigma^* \\ L \text{ is beslisbaar of herkenbaar} \end{cases}$

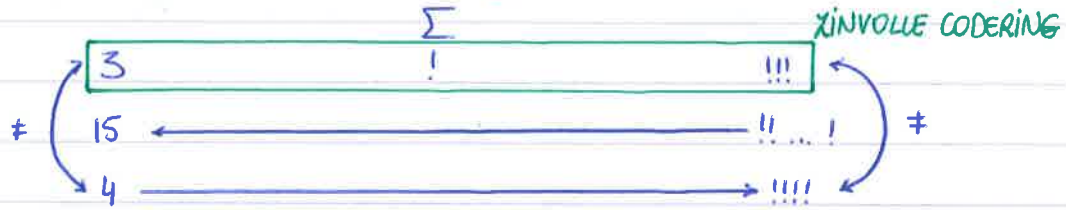
eigenschap van Priem zijn = behoren tot de verzameling priemgetallen

5. Encodering

graaf → input voor TM beschrijven = CODERING

→ VRAAG = Hoe stel ik het me voor ?

Wat zijn goede of slechte encoderingen ? Bestaan slechte coderingen wel ?



vb buurmatrix met dimensie 4

!!!! 0 1 0 ... $ = 1 manier om grafen voor te stellen

16

De encodering (van een graaf) moet niet uniek zijn !

⚠ sommige dingen die op het eerste zicht op een encodering lijken, zijn dat niet

	Σ	
3	!@	!!
4		!@@

Is er een criterium om te bepalen of een encodering beter is dan een andere ?

~ linear speed-up theorem

$|\Sigma| \times |Q|$

Aan de encodering zelf kan je weinig zien, omdat je niet weet welke probleem eemee dient opgelost te worden.

Slechte codering

vb N : priemgetallen : $\Sigma = \{ n, p, ! \}$ p!!! 3 = priemgetal. n!!!!

→ geen goede codering : ① redundantie

② moeilijk te berekenen

③ bedrieglijke strings vb p!!!!

≠ triviale berekening
→ er zit extra informatie in waar je op moet controleren.
⚠ Niet alle informatie die je er extra bijsteekt, zorgt voor een slechte encodering

Een TM encoderen

$Q \$ \Sigma \$ \Gamma \$ \delta \$ q_s \$ q_a \$ q_r$

alfabet <M> : $\$ |Q|_{01} \$ |\Sigma|_{01} \cup \Gamma | \$$ $\$ 3 \$_{01}$ ...

δ = transitietabel
→ gemakkelijk lijn per lijn erin zetten

TM-encodering als input voor andere TM of zelf voor zichzelf.

Is de verzameling van turingmachines herkenbaar/beslisbaar?

Je kan δ nakijken
→ voor een gegeven string kan je eenvoudig bepalen of het een TM kan voorstellen.
⇒ turingmachines zijn beslisbaar.

Vormen de turingmachines een beslisbare taal?

Ja: <M> als input voor TM
<M, s> = concatenatie van <M> en <s> met $s \in \Sigma_M^*$.
⇒ je kan een simulator schrijven voor <M, s> die antwoordt wat M(s) is.
→ Universele turingmachine (UTM)
↳ wat je er niet van mag verwachten:
* zelfde complexiteit als M
* zelfde aantal stappen als M
↳ wat je er wel van mag verwachten:
* zelfde antwoord (= loping gedrag)

grootte UTM: $|Q| \times |\Sigma|$
vb 5x7 - machine (Minski)
2x3 - machine (= 6 entry's in de transitietabel)
⚠ 2x2 is niet universeel!

6. Niet-beslisbare en niet-herkenbare talen

---

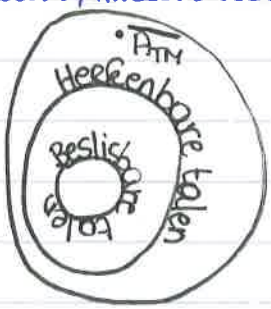
① $A_{TM} = \{<M, s> | M \text{ accepteert } s \text{ of } s \in L_M\}$. = ACCEPTANCE PROBLEEM

Is elke turingmachine M een herkenner?

ja: ze herkennen altijd de taal waarvoor ze stoppen in een accept-toestand.

→ A_{TM} is niet-beslisbaar, maar wel herkenbaar

CONSEQUENTIES:



⇒ $\overline{A_{TM}}$ is niet-herkenbaar en niet-beslisbaar.

A_{TM} is herkenbaar:

UTM $\langle M, s \rangle$ → ja / ∞ / nee → M(s)

⇒ UTM is de herkenner voor A_{TM}.

② $H_{TM} = \{\langle M, s \rangle \mid M \text{ eindigt op } s\}$ = HALTING PROBLEEM.

↳ is niet-beslisbaar, maar wel herkenbaar.

Verband tussen H_{TM} en A_{TM}

stel er bestaat een beslisser B voor H_{TM}

⇒ maak beslisser C voor A_{TM}

C($\langle M, s \rangle$) eerst B($\langle M, s \rangle$) laten uitvoeren ← ja / nee

→ is B($\langle M, s \rangle$) nee: C($\langle M, s \rangle$) is ook nee

→ is B($\langle M, s \rangle$) ja: C laat UTM($\langle M, s \rangle$) uitvoeren ← ja / nee

→ is UTM($\langle M, s \rangle$) nee: C($\langle M, s \rangle$) is ook nee

→ is UTM($\langle M, s \rangle$) ja: C($\langle M, s \rangle$) is ook ja

⇒ Er is geen beslisser B voor H_{TM}.

A_{TM} is niet-beslisbaar.

Bewijs stel A_{TM} is beslisbaar met beslisser B voor A_{TM}

→ B($\langle M, s \rangle$)
- ja als $s \in L_M$
- nee als $s \notin L_M$
- geen lus

→ contradictie-machine C: C($\langle M \rangle$) is omgekeerde van B($\langle M, M \rangle$)

CONTRADICTIE C($\langle C \rangle$) is omgekeerde van B($\langle C, C \rangle$)

uitkomst van C op zichzelf

⇒ B bestaat niet! ∎

Is $H_{TM} \cap A_{TM}$ beslisbaar?

stel L_1 en L_2 zijn niet beslisbaar → $L_1 \cap L_2$?

→ $L_1 \cup L_2$?

(9/11/2010)

turingmachine die f simuleert
→ n invoeren in f : $f(n)$ < 1: ja / ? / lus: nee

⇒ je hebt een herkenner

$A_{TM} = \{\langle M, s\rangle \mid s \in L_M\}$

$H_{TM} = \{\langle M, s\rangle \mid M \text{ stopt op } s\}$

TM : turing
- berekenbare gehele getallen 0.739...
- ontwerp de enumeratormachine

Intermezzo : Collatz conjunctuve

---

vb $7 \xrightarrow{3+1} 22 \xrightarrow{/2} 11 \xrightarrow{3+1} 34 \xrightarrow{/2} 17 \xrightarrow{3+1} 52 \xrightarrow{/2} 26 \xrightarrow{/2} 13 \xrightarrow{3+1} 40 \xrightarrow{/2} 20 \xrightarrow{/2} 10 \xrightarrow{/2} 5 \xrightarrow{3+1} 16 \xrightarrow{/2} 8 \xrightarrow{/2} 4 \xrightarrow{/2} 2 \xrightarrow{/2} 1$

```
f(n) {
  while (n ≠ 1) {
    if (even(n)) then n ← n/2
    else n ← 3n + 1
  }
}
```

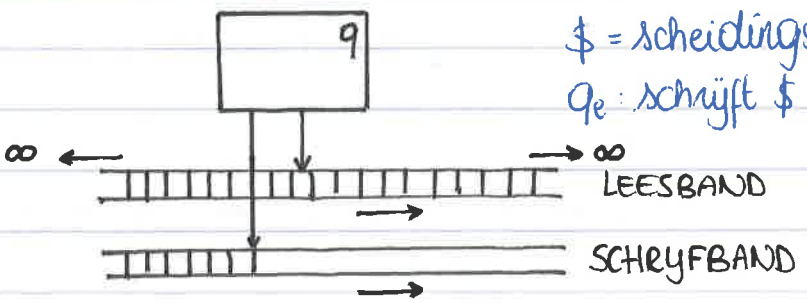
⇒ We kunnen niet zeggen of $f(n)$ stopt voor willekeurige invoer n.

Kunnen we zeggen of $f(n)$ stopt voor een gegeven n?

Ja: Er bestaat een turingmachine 1 die met 'ja' antwoordt en er bestaat een turingmachine 2 die met 'nee' antwoordt. Eén van beide is correct, alleen weten we niet welke.

Voor 1 waarde of voor een eindig aantal waarden bestaat er altijd een beslisser.

7. Enumeratormachine



$ = scheidingsteken
q_e : schrijft $ → q_s verder

Start: enumerator in starttoestand q_s
→ lege input
→ verzameling strings tussen $
= taal geënumereerd door de enumeratormachine

ETM — eindig / stopt in q_a of q_s
∞ — steeds meer $
nooit meer $ — meer output
geen extra output

1/3
1/2
⋮
$x \in \mathbb{Q}$ en $x < 1$
} berekenbare reële getallen

$\frac{\sqrt{2}}{2}$
π
e

→ De getallen die je kan berekenen, zijn TRANSCENDENT.
⇒ En zijn ONEINDIG veel berekenbare reële getallen
En zijn AFTELBAAR ONEINDIG veel enumeratormachines
⇒ En zijn AFTELBAAR ONEINDIG veel berekenbare reële getallen
⇒ De meeste reële getallen zijn niet te berekenen.

① Herkenbaar versus enumereerbaar

L is enumereerbaar als en slechts als L herkenbaar is.

Bewijs ① L is enumereerbaar, $E: L = L_E$
→ Maak herkenner M voor L
→ M krijgt s: M start de enumerator E telkens in enumeratortoestand q_E.
⇒ controleren of de laatste string == s
stopt / accept — doorzoek E

② L is herkenbaar
Maak een enumeratoe E voor L.
= naief: E: ∀s aan M: accept, dan zet E s op de tape
↳ wat mis? $s_1 \ldots s_i \ldots$ ⇒ potentieel oneindige lus

⇒ VARIANT: for $n = 0$ to ∞
do • genereer de eerste n strings van Σ^*
• voor elk van deze strings:
laat M n stappen doen op de string s
gemiddeld ∞
M → accept: schrijf s $
M → reject
M → mag niet klaar

= ITERATIVE DEEPENING

② Aftelbaar versus enumereerbaar

↳ $\exists$ bijectie met $\mathbb{N}$ ↳ effectief aftelbaar: $\exists$ TM die de bijectie implementeert

= STERKER
= herkenbaar

Elke taal $\subseteq \Sigma^*$ = aftelbaar
$\exists$ niet-herkenbare talen: $\overline{H_{TM}}$
A_{TM} en H_{TM} zijn herkenbaar maar niet beslisbaar.
Elke reguliere taal is beslisbaar: $\{\langle s \rangle \mid s$ gegeven door een RegExp$\}$

$\Updownarrow$

$\begin{cases} A_{DFA} = \{\langle D, s \rangle \mid D = \text{DFA en } s \text{ wordt geaccepteerd door } D\} \\ \quad \text{N} \\ \quad \text{RegExp} = \text{acceptance-probleem} \\ H_{DFA} = \{\langle DFA, s \rangle \mid \text{DFA stopt op } s\} \\ \quad \text{N} \\ \text{RegExp} = \text{halting-probleem} \end{cases}$

Je kan heel eenvoudig een DFA simuleren met een TM
⇒ je hebt een beslisser voor de taal A_{DFA}
voor elke string weet je dat de DFA ooit zal eindigen,
alleen moet je controleren of er de juiste encodering in zit.
→ A_{NFA} / H_{NFA} = eerst naar een DFA omzetten

	BESLISBAAR	HERKENBAAR	NIET-HERKENBAAR
Reghan	"alles" is beslisbaar → A/H, All, Empty, E, EQ	\	\
CFL	A, Empty, E, H	$\overline{EQ_{CFG}}$, $\overline{All_{CFG}}$	EQ_{CFG}, All_{CFG}
CSL	A_{CBA}, H_{CBA}	$\overline{E_{LBA}}$	E_{LBA}
TM	"niets" is beslisbaar (WEL: triviale dingen)	H_{TM}, A_{TM}, EQ_{TM}, $IsIn_{TM,Reghan}$	E_{TM}

① DFA

* $All_{DFA} = \{ \langle DFA \rangle \mid L_{DFA} = \Sigma^* \}$ is isomorf met
  → minimaliseer de DFA ≅ → 

* $Empty_{DFA} = \{ \langle DFA \rangle \mid \varepsilon \in L_{DFA} \}$
  → simulatie doen
  ⇒ eindigt altijd
  ⇒ is beslisbaar
* $E_{DFA} = \{ \langle DFA \rangle \mid L_{DFA} = \phi \}$
  → minimaliseer de DFA ≅ → 

* $EQ_{DFA} = \{ \langle DFA_1, DFA_2 \rangle \mid L_{DFA1} = L_{DFA2} \}$
  → $DFA_1 \Delta DFA_2$ symmetrisch verschil
  → indien leeg: DFA_1 en DFA_2 bepalen dezelfde taal

② CFL

* $A_{CFG} = \{ \langle CFG, s \rangle \mid s \in L_{CFG} \}$
  → ∃ afleiding die op s eindigt
  ↓ Chomsky NF
  lengte van de afleiding van $s = 2|s| - 1$

$A_{PDA} = \{ \langle PDA, s \rangle \mid s \in L_{PDA} \}$
↓
CFG → Chomsky normaalvorm
⇒ rekening houden met non-determinisme
⇒ bij simulatie kan je in een lus gaan waarbij de stack groter wordt.

* $H_{PDA} = \{ \langle PDA, s \rangle \mid PDA \text{ stopt op } s \}$
* $Empty_{CFG} = \{ \langle CFG \rangle \mid \varepsilon \in L_{CFG} \}$
  ↳ omvormen tot Chomsky NF : enige regel : $S \to \varepsilon$
* $E_{CFG} = \{ \langle CFG \rangle \mid L_{CFG} = \phi \}$
  → gegeven grammatica G omvormen tot G' :
  $L_G \neq L_{G'}$ maar als L_G niet leeg is, is $L_{G'}$ ook niet leeg.
  → zoek regels : $A \to \alpha, \alpha \in \Sigma^*$
  alle regels : $A \to$ iets $\to$ weg
  alle regels : $B \to Ay$ vervangen door : $B \to xxy$
  ~~① $S \to a$: taal is niet leeg~~
  ② stopt : taal is leeg
  vb ~~$A \to ab$~~ ~~$B \to aAb$~~
  ~~$A \to Xyx$~~ $B \to aabb$
* $EQ_{CFG} = \{ \langle CFG_1, CFG_2 \rangle \mid L_{CFG_1} = L_{CFG_2} \}$
  → niet beslisbaar
  $\overline{EQ_{CFG}}$ ⇒ genereer 1 voor 1 alle strings s
  ↑ $s \leq \begin{matrix} CFG_1 \\ CFG_2 \end{matrix}$ } zelfde antwoord

  indien het antwoord verschilt, zijn CFG_1 en CFG_2 niet equivalent.
  ⇒ $\overline{EQ_{CFG}}$ is herkenbaar
  ⇒ EQ_{CFG} is niet herkenbaar
* $All_{CFG} = \{ \langle CFG \rangle \mid L_{CFG} = \Sigma^* \}$
  ↳ PDA ⇒ geen minimalisatiemethode
  ⇒ niet-herkenbaar
  $\overline{All_{CFG}}$ is herkenbaar

(10/11/2010) ③ TM

* EQ_{TM} → REDUCTIE
* $E_{TM} = \{ \langle M \rangle \mid L_M = \phi \}$ is niet-beslisbaar ↔ $Is In_{TM, \{\phi\}}$

Bewijs : stel : er bestaat een beslisser E voor E_{TM}
→ maak een beslisser B voor A_{TM}.
→ B krijgt $\langle M, s \rangle$:

46.

① maak een machine M_s die input w krijgt en controleert:
- $w \neq s \Rightarrow$ reject
- $w = s \Rightarrow$ M laten lopen op s $\begin{cases} \text{accept} \\ \infty \\ \text{reject} \end{cases}$

② laat E lopen op M_s $\begin{cases} \text{accept} \\ \text{reject} \end{cases}$
indien - accept: $L_{M_s} = \phi \Rightarrow$ M aanvaardt s niet
- reject: $L_{M_s} \neq \phi$, $L_{M_s} = \{s\} \leftrightarrow$ M aanvaardt s

$\Rightarrow$ En is een beslisser voor A_{TM} terwijl A_{TM} niet beslisbaar is (= CONTRADICTIE)

* $\overline{E_{TM}} = \{\langle M \rangle \mid \exists s$: M accepteert $s\}$
E_{TM} is niet beslisbaar
$\Rightarrow \overline{E_{TM}}$ is niet beslisbaar, maar wel herkenbaar
(1 van beide moet ~~beslisbaar~~ herkenbaar zijn)

* $IsIn_{TM,S} = \{\langle M \rangle \mid L_M \in S\}$
verzameling van talen

→ Is de taal van de turingmachine regulier? $TM_{REGULIER}$

$TM_{REGULIER}$ is niet beslisbaar

Bewijs stel: R is de beslisser voor $IsIn_{TM,Reglan}$
→ Bouw een beslisser B voor A_{TM}
→ B krijgt $\langle M, s \rangle$
① maak een machine M_s die input w krijgt en controleert:
- $w = 0^n 1^n \Rightarrow$ accept
- $w \neq 0^n 1^n \Rightarrow$ laat M lopen op s $\begin{cases} \text{accept} \\ \infty \\ \text{reject} \end{cases}$

② laat R lopen op s
* R accepteert $s \leftrightarrow L_{M_s}$ is regulier
$\leftrightarrow L_{M_s} = \Sigma^* \leftrightarrow$ M accepteert s
* R verwerpt $s \leftrightarrow L_{M_s} = \{0^n 1^n\}$
$\leftrightarrow$ M verwerpt s

$\Rightarrow$ B is een beslisser voor A_{TM} terwijl A_{TM} niet beslisbaar is (= CONTRADICTIE)

Voor welke S is $IsIn_{TM,S}$ beslisbaar?

→ EXTREME MOGELIJKHEDEN VOOR S OVERLOPEN

vb $S = \phi$: $IsIn_{TM,S} = \{\langle M\rangle \mid L_M = \phi\}$ ⇒ accepteert nooit

$S = \mathcal{P}(\Sigma^*)$: $IsIn_{TM,S} = \{\langle M\rangle \mid L_M \in \mathcal{P}(\Sigma^*)\}$ ⇒ triviaal aan voldaan

S = herkenbare talen : $IsIn_{TM,S} = \{\langle M\rangle \mid L_M \in$ herkenbare talen$\} =$ ALLE TM

↳ als je er 1 uithaalt is de verzameling niet meer beslisbaar.

④ CSL

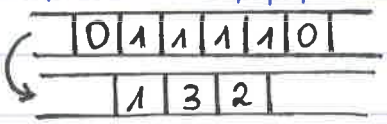
→ lineair begrensde automaat

TM : niet buiten de input lezen/schrijven

∃C : nuttige ruimte ≤ C.|input| (voor input groot genoeg)

$\Sigma \to \Sigma \times \Sigma$

$\{0,1\} \to \{0,1,2,3\}$



Kan je herkennen of een gegeven turingmachine een LBA is?

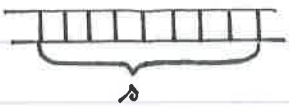
→ is $\{\langle M\rangle \mid M$ gaat nooit buiten zijn invoer$\}$ beslisbaar?

↳ het complement is herkenbaar en gezien het niet beslisbaar is, is deze verzameling niet-herkenbaar.

$\{\langle M\rangle \mid L_M \in CSL\} = IoIn_{TM,CSL}$ (niet-beslisbaar)

Je kan van een TM een CBA maken.

* A_{LBA}, H_{LBA} } configuratie van LBA voor s : lengte $= |s|+1$



$qs : |\alpha q' \beta| \leq |s|+1$

⇒ eindig aantal configuraties voor $s = N$

⇒ simuleer (mbv een TM) een LBA tot maximaal N stappen

① s is al geaccepteerd : $s \in L_{LBA}$

② s is al verworpen : $s \notin L_{LBA}$

③ zelfde configuratie is ondertussen opgedoken

= LBA zit in oneindige lus ⇒ $s \notin L_{LBA}$ (loop detectie mogelijk)

⇒ Beslisbaar ∎

48.

(16/11/2010) * $E_{LBA} = \{\langle M\rangle \mid M = LBA \text{ en } L_M = \emptyset\}$ is niet beslisbaar

↔ E_{CFG} is beslisbaar

Bewijs: Als E_{LBA} beslisbaar is, dan is A_{TM} beslisbaar

⇒ Computation history M voor s.

= opeenvolging van configuraties $ ... $

Met een LBA kan bepaald worden of 'iets' (= eindig) een accepteerende computation history is voor een gegeven M en s.

$LBA_{M,s}$ → zet <s> <M> op de tape

→ check heel de string

⇒ je kan nagaan of iets in een accepterende computation history zit.

Construeer beslisser B voor A_{TM}

→ input $\langle M, s\rangle$

- construeer $L_{M,s}$
- beslisser voor E_{LBA} op $L_{M,s}$ laten lopen:
  - reject ⇒ M accepteert s
  - geen string geaccepteerd door $LBA_{M,s}$
    ⇒ M op s is geen accepterende computation history

⇒ Er is een beslisser voor A_{TM}, terwijl A_{TM} niet beslisbaar is (= CONTRADICTIE)

8. Stelling van Rice

---

① Niet-triviale eigenschappen van turingmachines

→ niet alle TM hebben eigenschap P, maar ∃TM die P heeft

Als een eigenschap niet triviaal is, is $\overline{P}$ ook niet triviaal.

vb $TM_3 = \{\langle M\rangle \mid M \text{ heeft 3 toestanden}\}$

= niet-triviaal

② Taalinvariante eigenschappen van turingmachines
(= $IsIn_{TM,\{\}}$)

→ M_1 en M_2 met $L_{M_1} = L_{M_2}$ dan $P(M_1) = P(M_2)$

vb1 M_1 heeft 3 toestanden en $L_{M_1} = L_{M_2}$
HAAR dit betekent niet dat M_2 ook 3 toestanden heeft
$\Rightarrow TM_3 = \{\langle M\rangle \mid M$ heeft 3 toestanden$\}$
= niet-taalinvariant

vb2 $P_L(M) = (L_M \equiv L)$ is niet-triviaal maar wel taalinvariant
$\{\langle M\rangle \mid L_M = L\}$ (en L is herkenbaar)
$L_{M_1} = L_{M_2} \Rightarrow (L_{M_1} = L \Leftrightarrow L_{M_2} = L)$

vb3 $P(M) \triangleq$ (aantal L_M is oneindig) is niet triviaal maar wel taalinvariant
→ $IsIn_{TM,S}$ met $S = \{L \mid L \in \mathcal{P}(\Sigma^*)$ en aantal L = oneindig$\}$
↳ overaftelbaar groot en slechts aftelbaar veel turingmachines
→ $S = \{L \mid L \in \mathcal{P}(\Sigma^*)$ en aantal $L = \aleph_0\} \cap$ herkenbare talen

Elke niet-triviale en taalinvariante eigenschap P is van de vorm $IsIn_{TM,S}$ voor een gegeven S.

Stel: M bepaalt L en P(M) als eigenschap
$\Rightarrow L \in S$
M_2: $L_{M_2} = L \Rightarrow P(M_2)$ is true
$\Rightarrow S = \{L_M \mid \exists M: P(M) = \text{true}\}$

Stelling van Rice
Elke niet-triviale en taalinvariante eigenschap P van turing machines is niet-beslisbaar.

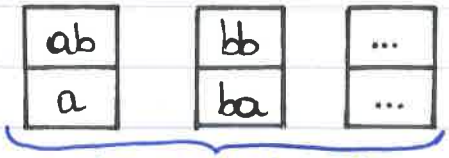
↳ $IsIn_{TM,S}$

- $EQ_{TM} = \{\langle M_1, M_2\rangle \mid L_{M_1} = L_{M_2}\}$
→ via constructie kan je de stelling van Rice toepassen
- $E_{TM} = \{\langle M\rangle \mid L_M = \phi\}$
→ Stelling van Rice eenvoudig toe te passen.

9. Post-correspondence probleem

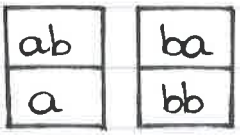
overeenkomst

→ Spelletje met dominoblokjes



eindig aantal blokjes

⇒ beslis of



...

→ string boven
→ string onder
} gelijk zijn

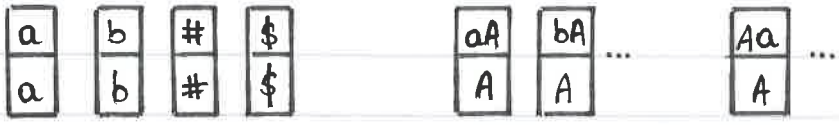
Post bewees: ① het spel is niet beslisbaar
② het spel is turingcompleet

→ Construeer voor een gegeven turingmachine M het postspel dat M kan simuleren op S.

Tabel met δ: Σ ⊢ A z

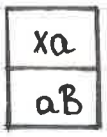
ab x start

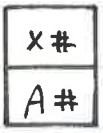
symbool x:



regels in de transitietabel:

①	xa	BaR
③	x#	A#S
⋮		

① 

③ 

} 1 blokje per regel δ

→ er zijn ook regels die onafhankelijk zijn van de turing machine.

stel: input ab

= speciale tegel voor de invoer.

⇒ je kan een computation history van een gegeven turingmachine op een input s simuleren.

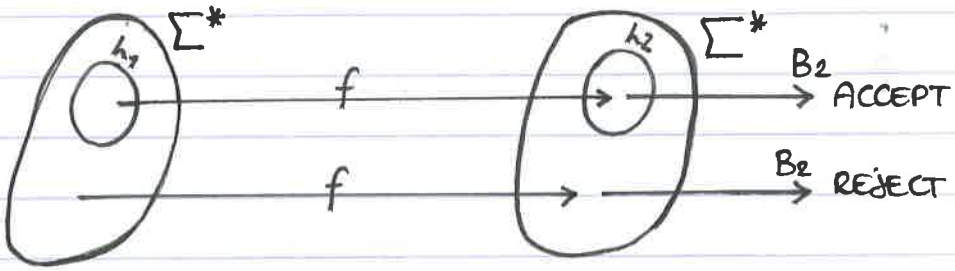
$	Xa	b	...
xab	aB	b	...

= deterministische turingmachine die stopt als je in een acceptantietoestand zit

Beslissen of het spel een oplossing heeft is equivalent met beslissen of M s accepteert. Dit is dan weer equivalent met A_{TM} dat niet beslisbaar is.

⇒ het postspel is turingcompleet

10. Reductie van problemen



$L_1 \leq_m L_2$ — many-to-one

L is reduceerbaar tot L_2 indien f turingberekenbaar is (∃TM die f berekent) en $f(L_1) \leq L_2$, $f(\bar{L}_1) \leq \bar{L}_2$.

Indien $L_1 \leq_m L_2$ en L_2 is berekenbaar (herkenbaar), dan is L_1 berekenbaar (herkenbaar).

Bewijs: ~~Beslissen~~ herkennen B_2 voor L_2.

→ maar ~~beslissen~~ herkennen B_1 voor L_1

B_1 s? $L_1 \rightarrow B_1(s) = B_2(f(s))$

→ werkt: f wordt geïmplementeerd door een turingmachine

⇒ we hebben 2 na elkaar geschakelde turingmachines

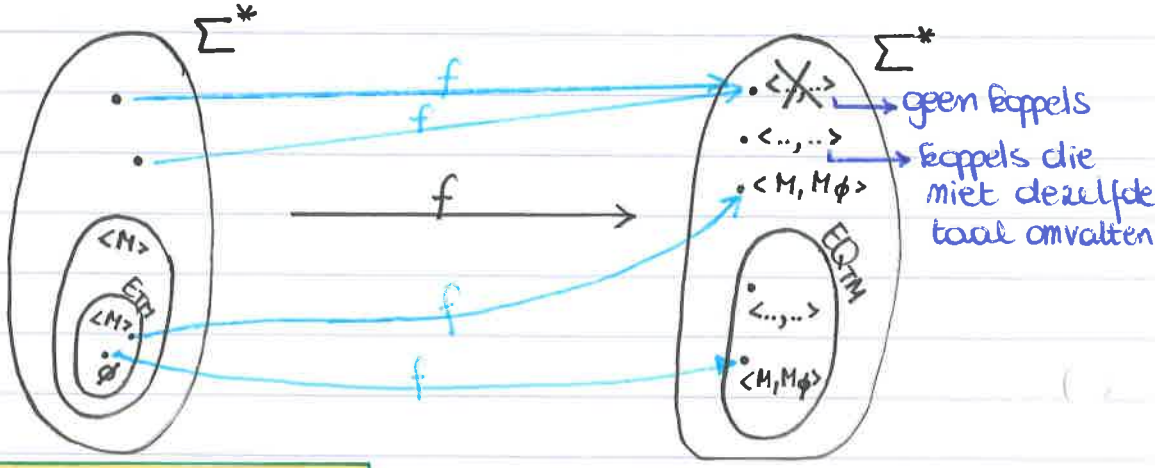
- $EQ_{TM} = \{\langle M_1, M_2\rangle \mid L_{M_1} = L_{M_2}\}$? niet-beslisbaar

$E_{TM} = \{\langle M\rangle \mid L_M = \phi\}$ is niet beslisbaar + $E_{TM} \rightarrow EQ_{TM}$: $\langle M\rangle \downarrow \rightarrow \langle M_1, M_2\rangle$

(17/11/2010)

$E_{TM} \leq_m EQ_{TM}$ → E_{TM} is niet beslisbaar
$L_{TM} = \phi$ → $\Rightarrow EQ_{TM}$ is niet beslisbaar

$f: \langle M \rangle \longrightarrow \langle M, M_\phi \rangle$
$\Sigma^* \longrightarrow \Sigma^*$
$f(\overline{E_{TM}}) \leq \overline{EQ_{TM}}$



Als $L_1 \leq_m L_2$ dan $\overline{L_1} \leq \overline{L_2}$

A_{TM} om te bewijzen dat EQ_{TM} niet herkenbaar en ook niet coherkenbaar (complement is (niet) herkenbaar) is.

	BESLISBAAR	HERKENBAAR	COHERKENBAAR
CFL Reghan	+	+	+
A_{TM}	−	+	−
$\overline{A_{TM}}$	−	−	+
EQ_{TM}	−	−	−

$\Rightarrow A_{TM} \leq_m \overline{EQ_{TM}}$ en $\overline{A_{TM}} \leq EQ_{TM}$

$A_{TM} \leq_m \overline{EQ_{TM}}$ → $\Rightarrow EQ_{TM}$ is niet herkenbaar

$\overline{A_{TM}} \leq EQ_{TM}$ → $A_{TM} \leq \overline{EQ_{TM}}$ → $\Rightarrow \overline{EQ_{TM}}$ is niet herkenbaar

① $f: A_{TM} \longrightarrow \overline{EQ_{TM}}$
$\langle M, s \rangle \longrightarrow \langle M_1, M_2 \rangle$
$\exists M_s$ die elke string accepteert als en slechts als M s accepteert: M_s accepteert $s : M_{\Sigma^*}$
of
M_s accepteert s niet : M_ϕ
$\Rightarrow$ je kan M_s construeren, maar je weet niet welke machine juist is: $\langle M_s, M_\phi \rangle$

② $f: A_{TM} \longrightarrow EQ_{TM}$
$\langle M, s \rangle \longrightarrow \langle M_s, M_{\Sigma^*} \rangle$
→ L_1 en L_2 zijn niet herkenbaar
$L_1 \leq_m L_2$ als beide polymorfe talen zijn (uitgezonderd ϕ en Σ^*) dan zijn ze polynoom equivalent.

11. Analyse van code / compilers

→ type correct → je wil dat het type beslisbaar is (dit geldt voor de meeste programmeertalen)

Eigenschappen van stukjes code die de compiler wil nakijken
① gebruik van gedeclareerde variabelen
② Dode code ≠ beslisbaar
└ conditie is altijd waar / onwaar
③ Wordt elke exception opgevangen statisch beslisbaar, maar niet dynamisch beslisbaar
④ Array[i]: ligt i in de ruimte die beschikbaar is voor de Array? ~ eindigen (~ haltingprobleem)
⑤ Eindigen
⑥ optimaliseren:
- loop invariante code naar buiten brengen
- pointer analyse: wijzen *p en *q naar hetzelfde? cond if (*p = *q) moet slagen

} kan de compiler niet perfect nagaan, want dit is niet beslisbaar (turingcompleet).

Datalog ~ Prolog zonder gestructureerde termen (aritmetiek) of andere database queytalen
⇒ minder sterk
⇒ je kan er meer eigenschappen rond bewijzen

12. Orakelmachines

A_{TM} is niet beslisbaar maar er bestaat een ORAKEL voor A_{TM}
= apparaat waar je een string $\langle M, s \rangle$ aan geeft en in één stap naar een accept of reject gaat (en dus een antwoord geeft).

→ geef de turingmachine de mogelijkheid om het orakel voor A_{TM} te raadplegen.
= ORAKELMACHINE $O^{A_{TM}}$
⇒ orakelmachines O^L die een orakel voor L raadplegen.

Hoe sterk zijn orakelmachines?

transformatie

Indien L beslisbaar is, dan is $L O^L = L_{TM}$

⇒ de orakelmachine voor een beslisbare taal geeft geen extra kracht.

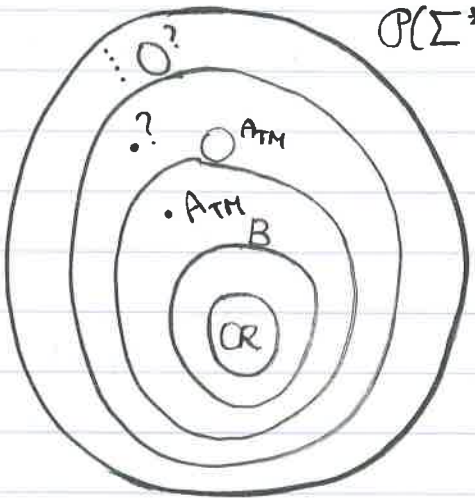
Indien L niet beslisbaar is, dan $L_{TM} \subset L O^L$.

⇒ MEER KRACHT — └ strikte inclusie.

kan met $O^{A_{TM}}$ alles beslist worden?

Er zijn AFTELBAAR ONEINDIG veel A_{TM}'s.
→ in de transitietabel kan je op een eindig aantal plaatsen een orakelmachine plaatsen.
⇒ Nee, want er zijn OVERAFTELBAAR ONEINDIG veel talen.

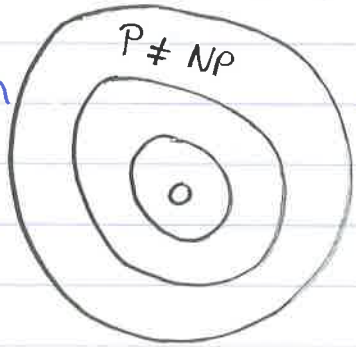
BESLISBAARHEID



⇒ hele hiërarchie van orakelklassen die verschillend zijn
└ oneindig groot

COMPLEXITEIT

→ hiërarchie kan nog in elkaar storten (NP=P)



polynome hiërarchie.

13. Turingreductie

Turingreductie: $L_1 \leq_T L_2$ indien O^{L_2} L_1 kan beslissen.

beslisbaar: $f: \Sigma^* \to \{0,1\}$

$f: \Sigma^* \to \Sigma^*$

$f: \mathbb{N}^m \to \mathbb{N}$

└ NIET AFTELBAAR ONEINDIG zulke functies

Welke functies kunnen door een turingmachine berekend worden?

= AFTELBAAR ONEINDIG veel.

Gödel / Herbrand: klasse van functies

= PRIMITIEF RECURSIEVE FUNCTIES: • mul $\mathbb{N} \to \mathbb{N}$, $i \mapsto 0$

• succesor $\mathbb{N} \to \mathbb{N}$, $i \mapsto i+1$

• projectie $\mathbb{N} \to \mathbb{N}$, $(a_1 \ldots a_n) \mapsto a_i$

Compositie nieuwe $h(\bar{x}) \triangleq f(g_1(\bar{x}), \ldots, g_m(\bar{x}))$

primitieve recursie nieuwe $h: \mathbb{N} \to \mathbb{N}$ $\quad h(0) = \text{constante}$

$h(y+1) = g(y, h(y))$.

⇒ is h een functie (overal gedefinieerd)?

Ja: $g(y, g(y-1, g(y-2, g(y-3, \ldots, g(0,c)) \ldots))$.

⇒ je komt in het gedefinieerd basisgeval.

→ Primitief recursieve functies = eerste voorstel van Gödel.

Ackermann ① functie Ack definiëren

② Ack & Primitief recursieve functies

③ Argumenteer dat Ack berekenbaar is.

Primitief recursieve functies kan je uitrekenen met een for-programma (= niet turing compleet)

② klopt want Ack stijgt sneller dan gelijk welke primitief recursieve functie.

(23/11/2010)

Primitief recursief : overal gedefinieerd : $\mathbb{N}^n \to \mathbb{N}$

Basis : * nul
* successor
* projectie op i^{de} element

Compositie : $f \circ g$

Primitieve recursie : $\begin{cases} f(0) = c \\ f(x+1) = g(f(x)) \end{cases}$

Ackermann : $\begin{cases} Ack(0,y) = y+1 \\ Ack(x+1,0) = Ack(x,1) \\ Ack(x+1,y+1) = Ack(x, Ack(x+1,y)) \end{cases}$

→ te lezen als een Haskell-programma

⇒ te beschouwen als een berekenbare functie

+ Ack stijgt sneller dan elke primitief recursieve functie

⇒ kan geen primitief recursieve functie zijn

14. Onbegrensde minimizatie

$f(x,y)$ h : voor elke y de kleinste x zodat $f(x,y) = 0$ en $f(x,y)$ is gedefinieerd voor $x \leq x_0$.

- $\exists$ nulpunten voor $f(x,y) = 0$
- $\nexists$ nulpunten → h is niet gedefinieerd

vb $f : \mathbb{N}^2 \to \mathbb{N}$

$(x,y) \mapsto 1$

⇒ f is niet gedefinieerd

vb $h(y)$

```
x = 0;
While f(x,y) ≠ 0
    x++;
return x;
}
```

bij het berekenen van f kan je in een lus gaan als f niet gedefinieerd is.

μ-recursieve functies ⟷ TM-functies

* part
* while

Een niet-herkenbare taal is niet μ-recursief

intuïtie : sterk stygende functie ⟷ niet-beslisbaar
beslissingsprobleem ⟷ talen

15. Busy beaver

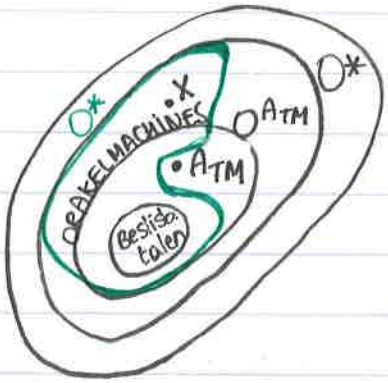
n: beschouw een turing machine met n (= eindig) toestanden
laat deze lopen met de lege string ε
→ stopt — accept : tel de 1'en die geproduceerd worden
\ reject └→ neem het MAXIMUM
→ bb(n) = maximaal aantal 1'en dat de turing machine heeft gezet als hij stopt.

De 'busy beaver'-functie is niet berekenbaar.

De functie bestaat en er is voor elke n een maximum.
⇒ bb(n) is eindig voor elke n.

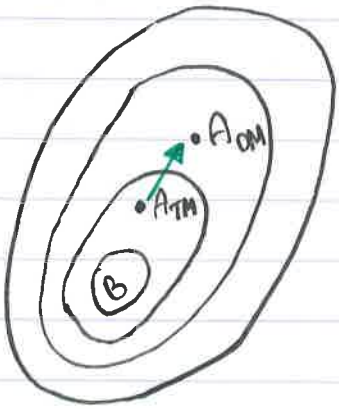
Stel : bb(n) is berekenbaar (~ Haltingprobleem)
We krijgen turing machine M en we willen dat hij stopt op ε.
→ |Q| toestanden
bb(|Q|)
→ We simuleren M
⚠ aantal 1'en ⟷ aantal stappen dat M maximaal moet doorlopen voordat hij stopt.

Afsluiten



$\mathcal{P}(\Sigma^*)$

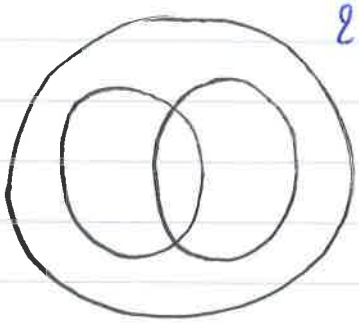
Waarom niet deze vorm?
Moet O^* ook A_{TM} omvatten?



$\mathcal{P}(\Sigma^*)$

A_{OM} is niet beslisbaar (p 36)

Turing jump om van de ene taal een andere te maken.



2 overlappende orakelklassen

transitieve relatie $\leq_T$
MAAR $\leq_T$ is niet unair
$\Rightarrow$ niet elke 2 talen zijn vergelijkbaar

Binnen elke klasse zijn er ook Turing complete problemen
$\forall L \leq L_C$.

HOOFDSTUK 3: Herschrijfsystemen

Plus heeft termen
- bevat alles uit $\mathbb{N}$ (kan oneindig groot zijn)
- $t_1 + t_2 \neq t_3$
- termen hebben GEEN BETEKENIS

en herschrijfregels
- term → term
- plusregels:
  - commutativiteit: $(1+2) \longrightarrow (2+1)$, $(2+1) \longrightarrow (1+2)$
    ⇒ ONEINDIG VEEL
  - associativiteit: $((1+2)+3) \longrightarrow (1+(2+3))$
    ⇒ ONEINDIG VEEL
  - Optelling: $(2+1) \longrightarrow 3$
    ⇒ ONEINDIG VEEL
  - Substitutie: binnen term T: X en $X \longrightarrow Y$
    $T(X) \longrightarrow T(Y)$
    ↳ notie van deeltermen

⚠ lussen mogelijk
- elke regel kan je in de omgekeerde volgorde toepassen

kan $((1+2)+3) \longrightarrow (3+3)$?
↳ lijkt redelijk (Substitutieregel)
MAAR ∄ regel in het systeem die dit toelaat.

→ Een term t kan je herschrijven: $t \longrightarrow$
binnen t is $\longrightarrow$

⇒ t is REDUCEERBAAR

In plus geldt: een term staat in normaalvorm indien er geen '+' in staat.

Bestaat er voor elke term een normaalvorm? Is die uniek?
Bij elke reductie probeer je één '+' kwijt te raken (STRATEGIE)
⇒ Er bestaat een normaalvorm en deze is uniek.

Van plus naar functies x, y, z
↳ termen { 0, $(t_1 + t_2)$, x y z → VERANDERLIJKEN

vb $(((x+x)+y)+3)$: $f : \mathbb{N}^2 \to \mathbb{N}$
↳ toegepast op (3,2)
$t_{x/3, y/2} \xrightarrow{*}$ herschrijven tot normaalvorm

Plus is zwak
welke functies kan je definiëren met Plus?
Lineaire functies met coëfficiënten in N.

(29/11/2010) 1. λ-calculus

→ termen t, reductieregels r en normaalvormen n.
→ synthax : t_1 t_2
f a ~ f toegepast op a
⇒ moduleert de functie.

vb ++ 1
→ we hebben geen constanten of ingebouwde dingen nodig
→ geen datastructuren
vb f a b → (f a) b @@ f (a b) ~ CURRYING
⚠ wat wordt bedoeld? Hoe moeten we gecoperen?
→ Functies hebben slechts één argument.
WE KIJKEN ENKEL NAAR NIET-GETYPEERDE λ-CALCULUS.
vb 1+ , +++ , ...
HIERBIJ VERONDERSTELLEN WE EEN AANTAL INGEBOUWDE FUNCTIES.
vb +3 4 → 7
En bestaan IF-functies en functies met TRUE en FALSE
vb IF TRUE t_1 t_2 → t_1 } meerdere herschrijfregels voor IF
IF FALSE t_1 t_2 → t_2 }
vb IF 1+5 → kan je niet herschrijven (= NORMAALVORM)

① λ-abstractie

vb $(\lambda x. + x\ 1)\ 3 \longrightarrow +\ 3\ 1 \longrightarrow 4$

term1 term2

evaluatie kan tot een nieuwe functie leiden.

*λ-term

<VARIABELE>
<INGEBOUWDE CONSTANTE / FUNCTIE>
<λ-t> <λ-t>
λ <VARIABELE> . <λ-t>
(<λ-t>)

vb $\lambda x. + x\ 1\ 7 \longleftrightarrow (\lambda x. + x\ 1)\ 7$

*λ-regels — η-conversie
— β-conversie — β-reductie
— β-abstractie
} 2 mogelijke richtingen
— α-conversie.

a) β-conversie : β-reductie ($\xrightarrow{\beta}$)

=functie toepassen op 1 argument

$(\lambda x. B(x))\ a$

$\xrightarrow{\beta}$ kopiëer B(x) en vervang x door a: B(x\a)

vb $(\lambda x. + x\ 1)\ 7 \xrightarrow{\beta} (\lambda x. + 7\ 1)$

⚠ NESTING VAN FUNCTIES

f(x) h

...

g(x) h

...

x

...

niet dezelfde

x

...

→ x zowel in f als g gebruikt.

vb $(\lambda x. (\lambda x. + x)\ 2)\ 7 \longrightarrow (\lambda x. + 7)\ 2$

→ Notie van vrije en gebonden variabelen

vb $(\lambda x. + x(y))\ x$

gebonden, vrij

{ ① wat is de expressie waar de variabele in voorkomt?
② wat is het VOORKOMEN van de variabele in die expressie?

vb E == x, dan is x vrij in E.

$E = (A\ B)\ x$ $\qquad$ $E = \lambda x.E$

y → vrij voorkomen van y in E.

vb $\lambda x. + ((\lambda y. + yz)7)\ x = E$

y is gebonden in E, maar is vrij in de deelexpressie
x is vrij in E
x is gebonden in E, maar is vrij in de body van de λ-abstractie

b) β-conversie : β-abstractie (←β—)

vb $3 \longleftarrow +1\ x$

Wat als je niet weet in welke richting de β-conversie is?

$\sim t_1 \leftarrow\!\!\beta\!\!\rightarrow t_2$

c) α-conversie

je mag variabelen hernoemen, maar hierbij mogen geen variabelen gevangen worden.

vb
```
f()h                    f()h
  int x;                  int x; y
  x                       x  y
  ...                     ...
  h                       h
    float x;                float y;
    a                       y  y
    ...                     ...
  }                       }
  ...                     ...
}                       }
```

vb $(\lambda x + x\,1) \xrightarrow{\alpha} \lambda y + y\,1$ ~ SEMANTIEK BLIJFT BEHOUDEN.
~ LYKT NIKS ESSENTIEEL TE DOEN!

vb $\lambda x.\ (\lambda y.\ +\,x\,y) \xrightarrow[x \to y]{\alpha} \lambda y.\ (\lambda y.\ y\,y)$
(E; x vrij in E; y gebonden in E)

$\boxed{\lambda x.\ E \xrightarrow[x \to y]{\alpha} \lambda y.\ E'}$

→ Vrije x in E vervangen door y, waarbij geen variabelen gevangen genomen worden.

d) η-conversie

$\boxed{\lambda x.\ F\,x \xrightarrow{\eta} F}$ waarbij F zelf een functie is.

vb $(\lambda x.\ F\,x)\,a \longrightarrow F\,a\ \forall a$ MAAR x mag niet vrij voorkomen in F

Een deelterm die gereduceerd kan worden, noemt men een redex.

vb $(\lambda x.\ ((\lambda f.\ \lambda x.\ (f\,x))\ x))\ z \to \lambda p.\ B$
(① ② ② ①)

$\xrightarrow{\beta①}$ $((\lambda f.\ \lambda x.\ (f\,x))\ x) \to \lambda p.\ B$
(② ②)

$\xrightarrow{\beta②}$ $\lambda x.\ (x\,x)$

$\xrightarrow{\eta}$ MAG ALS x EEN FUNCTIE IS

(of) $\xrightarrow{\beta②}$ $(\lambda x.\ (\lambda x.\ (x\,x))\ x) \xrightarrow{\beta①} \lambda x.\ x\,x$

⇒ α-conversie om het vangen te vermijden

$\xrightarrow{\alpha}$ $(\lambda x.\ ((\lambda f.\ \lambda y\ (f\,y))\ x))\ z$

$\xrightarrow{\beta}$ $(\lambda x.\ (\lambda y\ (x\,y))\ x)$

(30/11/2010) ② Functies

Om te kunnen rekenen, moet je getallen kunnen voorstellen in λ-calculus ($\mathbb{N}$)

notatie: $F^{0}(M) = M$

$F^{n+1}(M) = F(F^{n}(M))$

C_n = n^{de} getal uit $\mathbb{N}$

$= \lambda f.\ \lambda x.\ f^{n}(x) = \lambda f x.\ f^{n}(x)$

} 1 mogelijke voorstelling van $\mathbb{N}$ in λ-calculus

Bevat dit een Redex? Nee, je kan het niet reduceren

$A_+ = \lambda x\, y\, p\, q.\ x\, p\, (y\, p\, q)$
= om '+' uit te drukken met deze getallen
vb $A_+\ C_3\ C_4 \xrightarrow{*} C_{10}$
⇒ je kan heel wat functies beschrijven met λ-calculus

③ Algoritmen

$\begin{cases} IF = \lambda x\, y\, z.\ (x\, y)\, z \\ TRUE = \lambda x\, y.\ x \\ FALSE = \lambda x\, y.\ y \end{cases}$

vb 1 IF TRUE p q (met p en q willekeurig)
$= (\lambda x\, y\, z.\ (x\, y\, z))\ (\lambda a\, b.\ a)\ p\ q$
(x y z): BODY; x: FORMELE PARAMETER; (λ a b. a): ARGUMENT

$\xrightarrow{\beta} (\lambda y\, z\ ((\lambda a\, b.\ a)\ y\ z))\ p\ q$
$\xrightarrow{\beta} (\lambda z.\ ((\lambda a\, b.\ a)\ p\ z)\ q$
$\xrightarrow{\beta} (\lambda a\, b.\ a)\ p\ q$
$\xrightarrow{\beta} (\lambda b.\ p)\ q$
$\xrightarrow{\beta} p$

als je β-conversie toepast, dan pas je dit toe op de vrije vormen van p

AFHANKELIJK VAN DE STRATEGIE KAN HET AANTAL STAPPEN OM IN DE NORMAALVORM TE KOMEN VERSCHILLEN.

vb 2 IF + 3 4 heeft zin ($\xrightarrow{*}$ normaalvorm)

④ Datastructuren – gelinkte lijsten

Head lijst
Tail lijst
Cons element lijst (toevoegen aan een lijst)

vb $head\ (cons\ e\ l) \xrightarrow{*} e$
$tail\ (cons\ e\ l) \xrightarrow{*} l$
} Invarianten.

wat als l geen lijst IS?
We zitten in een niet-getypeerde context
⇒ maakt niet uit.

① Encodering

$\mathbb{N}$ — unair
$\mathbb{N}$ — binair

→ In λ-calculus moet je iets kiezen als encodering

② Strategie

Om een gegeven λ-expressie om te zetten in zijn normaalvorm

PROGRAMMA = λ-abstractie
└ uitvoeren: input = rij
└ eindigt ⇒ resultaat = $\xrightarrow{*}$ normaalvorm.

Als alles eindigt ↔ we hebben NIET te maken met een Turing Compleet formalisme.

vb $(\lambda x.xx)(\lambda x'.x'x')$ } op α-conversie na dezelfde term.
$\xrightarrow{\beta}$ $(\lambda x'.x'x')(\lambda x'.x'x')$ }

⇒ heeft geen normaalvorm

vb. $(\lambda x.3)((\lambda x.xx)(\lambda x.xx))$
$\xrightarrow{\beta}$ idem ∿ ONEINDIGE LUS ↺
$\xrightarrow{\beta}$ 3 ∿ ONMIDDELLIJK DE NORMAALVORM

SOMS IS DE STRATEGIE CRUCIAAL OM DE NORMAALVORM TE BEREIKEN.

Strategieën: (moet in de specificatie van een taal staan)

① Argument EERST evalueren voor je de functie toepast
= CALL-BY-VALUE (∿ Java)

② Functie EERST toepassen en argumenten invullen indien nodig
= CALL-BY-NEED (luie evaluatie) (∿ Haskell)

Voor Prolog maakt het geen verschil (voor de eindigheid) of het gespecificeerd is als call-by-value of als call-by-need.

③ Onderlinge plaats van 2 redexen in λ-calculus

- 1 redex binnen een andere
- 1 redex links van de andere

→ Als de binnenste van 2 redexen geëvalueerd wordt
= CALL-BY-VALUE

→ Als de buitenste van 2 redexen geëvalueerd wordt
= CALL-BY-NEED

Voor de keuze tussen links en rechts moet je, om een strategie

te hebben die lussen vermijdt, eerst de linkse evalueren
→ REDUCTIE IN NORMAALORDE = meest linkse, buitenste (= call-by-name)

vb $(\lambda x.\ +\ x\ x)\ \ (+\ 3\ 4)$

normaalorde → $+\ (+\ 3\ 4)\ \ (+\ 3\ 4)$
→ $+\ 7\ \ (+\ 3\ 4)$
→ $+\ 7\ \ 7$
→ 14

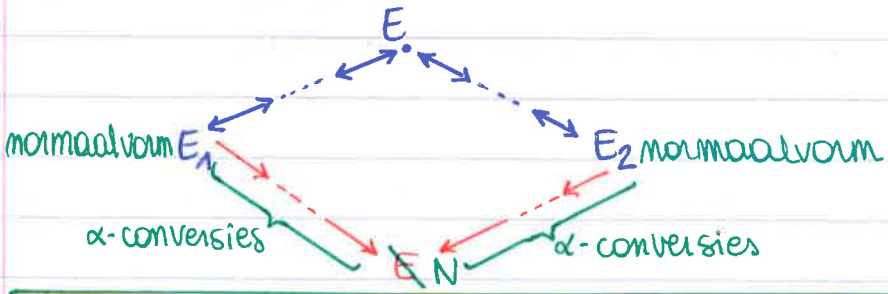
of → $(\lambda x.\ +\ x\ x)\ 7$
→ $+\ 7\ 7$
→ 14

Haskell: kan optimaal zijn (herschrijven van grafe)

MEER STAPPEN
⇒ NORMAALORDE IS NIET OPTIMAAL!

2. Stellingen van Church-Rosser

Church-Rosser I



Indien $E_2 \overset{*}{\longleftrightarrow} E_1$, dan $\exists\ E \ldots\ E_1 \overset{*}{\rightarrow} E$ en $E_2 \overset{*}{\rightarrow} E$.

Gevolg: Als $E \overset{*}{\rightarrow} E_1$ = normaalvorm en $E \overset{*}{\rightarrow} E$ = normaalvorm
dan zijn E_1 en E_2 gelijk (op α-conversie na)
= UNICITEIT VAN DE NORMAALVORM
⇒ λ-CALCULUS IS CONFLUENT

of gelijk welke strategie die eindigt op een normaalvorm, levert dezelfde normaalvorm op α-conversie na.

Church-Rosser II

Reductie in normaalorde brengt je gegarandeerd naar de normaalvorm indien die bestaat.

↳ strategie van Haskell is compleet (die van prolog niet)

(7/12/2010)

Recursie → functie-definitie waarin je de functie zelf oproept
→ In principe is λ-calculus hiervoor te zwak.

vb. FAC = (λm. IF (=m 0) 1 (*m FAC (m-1))) → Geen goede definitie van FAC
(IF vervangen; FAC kan je niet vervangen)

→ RETEER: FAC = (λf.(fm. IF(=m 0) 1 (*m (f (-m 1))))) FAC
H (β-abstractie)

FAC = H. FAC

FAC is een fixed point van H.

Hebben alle functies een vast punt?

Voor elke functie in de λ-calculus kan je een vast punt vinden.

vb niet voor (wel) $f: \mathbb{N}_\infty \to \mathbb{N}_\infty$
$x \mapsto x+1$

→ Vast punt combinator y
= beeldt elke functie X af op een vast punt van X
$\forall X : X(yX) = (yX)$
$\Rightarrow yH = FAC$
└ een FAC, dé FAC bestaat niet.

① y kan in λ-termen
$y = (\lambda h.(\lambda x. h(xx))(\lambda y. h(yy)))$
vb $(yH)3 \xrightarrow{*} 6$ (moet kunnen, maar kost veel werk)

② $\forall X : X(yX) = (yX)$

→ y = λ-magie

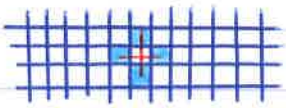
vb Lisp (niet-lui) ⟷ Haskell (lui)

vb In Hava is er 1 constructie die lui is: if-then-else (ook OF en AND)
= lui in het 2de en 3de element
MOET vb if x=0 then y=0 else $y = \frac{1}{x}$

HOOFDSTUK 4: Andere rekenparadigma's

1. Cellulaire automaten

Vb Game of life (John Conway)



• levend, wit = dood

initialisatie init = gen 1

$gen_n \rightarrow gen_{n+1}$

→ DFA's die verbonden zijn (boven-onder-links-rechts)

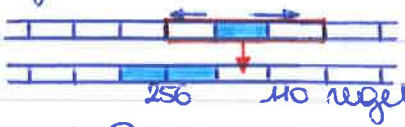
↳ $\delta: Q_l \times Q_r \times Q_b \times Q_o \times Q_s \rightarrow Q_s$

→ Spel is Turing Complet — 1 UTM op de tape

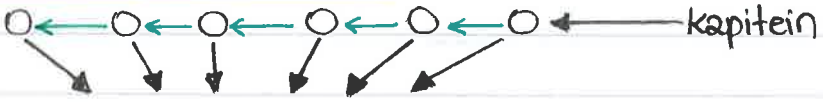
⇒ bijna alle vragen zijn onbeslisbaar.

John Von Neumann ~ zelfreparerende DFA's

Stephen Wolfram ("A new kind of science") = turing Complet

↳ 

Firing Squad Problem (Myhill)



~ SYNCHRONISATIE PROBLEEM

X (slachtoffer)

Elke soldaat (o) is een DFA

⇒ kunnen niet tellen en weten niet met hoeveel ze zijn.

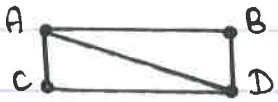
= Aan elkaar gekoppelde DFA's kunnen een krachtig geheel vormen.

2. DNA-computing

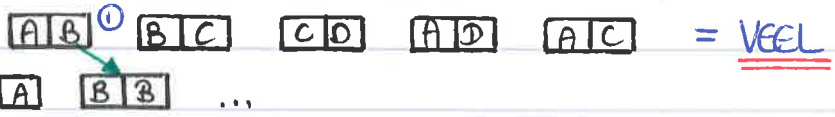
→ Je maakt een fysisch systeem om voor 1 instantie van het probleem een oplossing te geven.

↔ NORMALE PROGRAMMEERAANPAK: 1 programma voor elk generisch probleem.

vb Hamiltoniaans pad = NP-COMPLEET



(∧ Adelmann – RSA)

→ Componenten  = VEEL

① = strings waar we A en B (en alle andere bogen) coderen

OVEERLICHT ⇒ in beide richtingen

⇒ Eigenschap van aan elkaar vastkleven

 vastgehouden door

→ filter: op het einde iets overhouden

⇒ Je hebt een Hamiltoniaanse kring (en die bestaat)

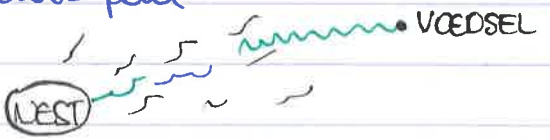
⇒ Er bestaat een kans dat er geen Hamiltoniaanse kring gevonden wordt terwijl er wel een is.

→ Deze kans zo klein mogelijk houden.

3 Ant-Computing

= geïnspireerd door de natuur

vb kortste pad



GEURSPOOR → minder kans op fouten

~ MIEREN HEBBEN GEEN GEHEUGEN

→ random systeem is te moeilijk, systeem past zich aan zonder intelligentie.

vb mier vindt niets → spoor wordt niet versterkt op de terugweg

⇒ zal niet gevolgd worden

mier vindt eten → volgt geurpad terug en versterkt het zodat de andere mieren het ook volgen en versterken

→ ETEN OP?

als ① > ② dan verdwijnt ① sneller

DE KANS OP EEN KORTER PAD STIJGT MET DE TIJD. DE KANS OP HÉT KORTSTE PAD IS KLEIN, MAAR ZAL STERK BENADERD WORDEN.

4. Quantum computing

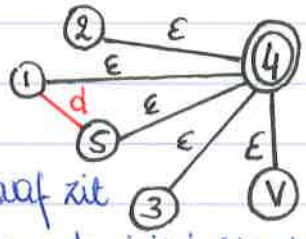
→ NP - complete problemen in polynome tijd oplossen (NPC)
vb priemdeler van een getal bepalen (tot 15)

(14/12/2010) Vragenles:

HB p. 32: Bewijs algoritme 12.2

In repeat: $\exists(p_a, q_a, \_)$
↳ mag eender welk label zijn

Bewijs ⇒



1e geval: $(p,q,\varepsilon) \in V$
als (p,q,ε) in de graaf zit
weet je zeker dat die van de initialisatie komt.
$S(s,d) = V$
→ heb je nu toegevoegd

2e geval: $(p,q,\lambda) \in V$
$(\delta(p,x), \delta(q,x), ?)$ zaten reeds in de graaf!
⇒ dit kan je terugbrengen tot het basisgeval.

Het verschil tussen aftelbaar en effectief aftelbaar

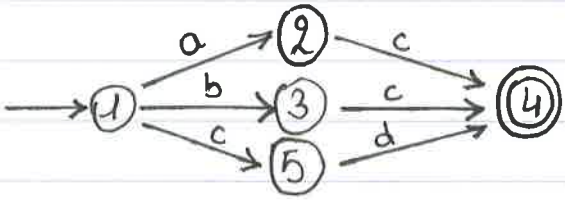
V = aftelbaar ∃ bijectie $b: V \to \mathbb{N}$
effectief aftelbaar: b kan geconstrueerd worden door een TM.
wel aftelbaar, maar niet effectief aftelbaar:
TM s die eindigt op input ε
↳ die bijectie kan je niet effectief construeren
(= er is geen algoritme voor)

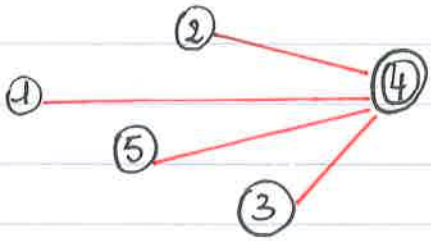
HB p. 96: Bewijs A_{TM} ≠ beslisbaar

C(⟨c⟩) accept c → B(⟨c,c⟩) = accept
↳ beslisser voor A_{TM}
$B(\langle M, s\rangle)$ = accept ⇔ M(⟨s⟩) accepteert

Je gebruikt hier C als oppositie
→ geeft tegenstrijdigheid

HB p. 33 : figuur 2.13





initialisatie : trek een boog tussen de finale toestand en elke andere niet-finale toestand.

→ voor toestand 1 en 5 probeer je een karakter x te vinden zodat je al kan beslissen dat 1 ≠ 5

$\delta(1,x)$ } voor $x = d$ vindt je dat deze twee
$\delta(5,x)$ } toestanden f-verschillend zijn.

HB p. 105 : Bewijs stelling H.2 : $REGULIER_{TM}$ is niet beslisbaar

H_S is niet enkel invoer voor R, loopt zelf niet.
↳ accepteert heel Σ^* of alleen strings van de vorm 0^m1^m.

Als R → Beslisser B voor A_{TM} : B(<M,s>)

EERST : L : H_S — $\Sigma^* \in$ Reghan ⇔ M accepteert S.
— $0^m1^m \notin$ Reghan ⇔ M accepteert S niet

⇒ R kan enkel beslissen in welk geval we zitten

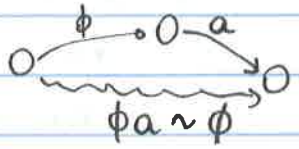
B : (<M,s>) maakt H_S → zet die op de tape
→ accepteert iets regulier als M s accept
→ accepteert iets niet-regulier als M s verwerpt.

laat R(H_S) — accept ⇔ M accepteert s
— reject ⇔ M verwerpt s

We bouwen een machine (hier H_S) niet om die te laten lopen maar omdat we daar eigenschappen van kennen die we kunnen gebruiken.

(12/10/2011) Project → ook vermelden met wie je over het project hebt gepraat

⊛ GNFA $\xrightarrow{\varepsilon}$ $\xrightarrow{\phi}$ RegExp die de lege taal bepaald
= BOOG DIE JE NOOIT KAN VOLGEN

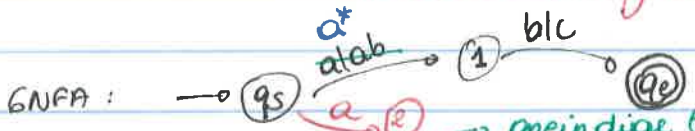


⊛ DFA $= (\Sigma, \delta, q_s, F, Q)$ $\quad \delta : \Sigma \times Q \to Q$

GNFA $= (\Sigma, \delta, q_s, q_e, Q)$ $\quad \delta : (?) \times Q \to P(Q)$

$(?) = \text{RegExp}_\Sigma$ want NIET-DETERMINISTISCH

in principe is dit een goede signatuur maar een nuttigere is: $\delta : P(\Sigma^*) \times Q \to P(Q)$



transitietabel:

	$\{\varepsilon, a, aa, \ldots\}$	$\{1\}$
q_s	~~$\{a, ab\}$~~	$\{1\}$
1	$\{b, c\}$	$\{q_e\}$
q_s	$\{a\}$	$\{1, 2\}$ verz. van toestanden waar je in kan komen

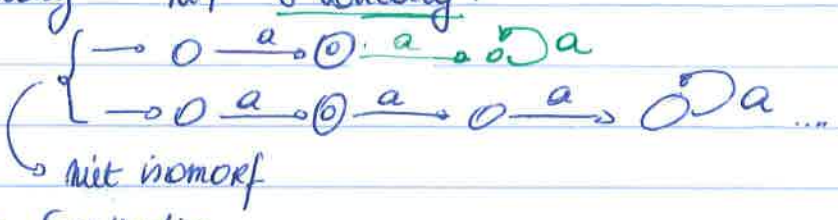
⊛ Deterministisch maken vle NFA → DFA

$\#Q_m \leq \#Q_d$ meestal < als je die niet-bereikbare toestanden weglaat.

$\exists$ NFA m ↑ $\quad \#Q_d = O(2^{\#Q_m})$ via del. algoritme

⊛ p30: Vooruit: gegeven $L \in RE$, bestaan er zo veel DFA's voor L?
→ ja als er onbereikbare knopen zijn.
→ wat als er wel onbereikbare knopen zijn?

Stel: L is eindig hah & volledig?



→ DFA met δ volledig

→ Is er een lus? ja:

DFA heeft een Q die eindig is.

Neem nu string s met $|s| > |Q|$

$\Rightarrow q_s \to q_1 \to q_2 \to \ldots$

minstens 2 dezelfde toestanden

$\Rightarrow \exists$ altijd een lus!

⊛ Hoeveel | Reguliere talen bestaan er? ($\Sigma = \{a,b\}$)

Reguliere exp

NFA's

DFA's

- ~~∅~~
- ~~EINDIG~~
- ONEINDIG — AFTELBAAR
  - ~~OVERAFTELBAAR~~

$\Rightarrow \exists$ niet-reguliere talen.

Hoeveel talen zijn er?

→ $\mathcal{P}(\Sigma^*)$

$\Rightarrow \#\mathcal{P}(\Sigma^*)$

- ~~∅~~
- ~~EINDIG~~
- ONEINDIG — ~~AFTELBAAR~~
  - OVERAFTELBAAR

Aftelbaar: $\exists$ bijectie $b: \mathbb{N} \to RE$

$\updownarrow$

constructie: talen met een bep. grootte enumereren.