Automaten en Berekenbaarheid

Pieter Vanderschueren

Academiejaar 2023-2024

Inhoudsopgave

1 Talen en automaten

1.1 Wat is een taal?

Definitie 1.1: String over een alfabet Σ

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

Toepassing 1.1: ϵ-compressie van een string

Als we uit een string $w \in \Sigma_\epsilon$ elk voorkomen van ϵ schrappen, wat resulteert in een nieuwe string s, dan noemen we s de ϵ-compressie van w.

Definitie 1.2: Taal L over een alfabet Σ

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

1.2 Een algebra van talen

Definitie 1.3: Een algebra- of algebraïsche structuur

Een algebra- of algebraïsche structuur is een verzameling met daarop een aantal inwendige operaties: dikwijls binaire operaties, maar unair of met grotere ariteit kan ook. Zo wordt de verzameling van alle talen over een alfabet Σ een algebra als we als operaties unie, doorsnede, complement, etc. definïeren. Meer concreet: als L_1 en L_2 twee talen zijn, dan is

- de unie ervan een taal: $L_1 \cup L_2$
- de doorsnede ervan een taal: $L_1 \cap L_2$
- het complement ervan een taal: $\overline{L_1}$

Eigenschap 1.1: Concatenatie van twee talen

Gegeven twee talen L_1 en L_2 over hetzelfde alfabet Σ, dan noteren we de concatenatie van L_1 en L_2 als L_1L_2 en definïeren we:

$$L_1L_2 = \{xy \mid x \in L_1,\ y \in L_2\}$$

Eigenschap 1.2: Kleene ster van een taal

De Kleene ster van een taal wordt gedefinieerd als volgt:

$$L^* = \cup_{n \geq 0} L^n$$

Opmerking: $L^+ = LL^*$

Toepassing 1.2: Kleene ster van een alfabet

De verzameling van alle strings over een alfabet Σ is de kleene ster van het aflabet Σ^*, en volgt dus dat

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

1.3 Reguliere expressies en reguliere talen

Definitie 1.4: Reguliere Expressie (RE) over een alfabet Σ

E is een **reguliere expressie** over een alfabet Σ indien E van de vorm is

- ϵ
- ϕ
- a waarbij $a \in \Sigma$
- $(E_1 E_2)$ waarbij E_1 en E_2 reguliere expressies zijn over Σ
- (E_1^*) waarbij E_1 een reguliere expressies is over Σ
- $(E_1 | E_2)$ waarbij E_1 en E_2 reguliere expressies zijn over Σ

Definitie 1.5: Reguliere taal

Een reguliere expressie E bepaalt een **reguliere taal** L_E over hetzelfde alfabet Σ als volgt:

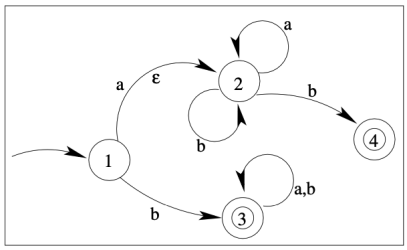
- als $E = a$ (met $a \in \Sigma$) dan is $L_E = \{a\}$
- als $E = \epsilon$ dan is $L_E = \{\epsilon\}$
- als $E = \phi$ dan is $L_E = \emptyset$
- als $E = (E_1 E_2)$ dan $L_E = L_{E_1} L_{E_2}$
- als $E = (E_1)^*$ dan $L_E = L_{E_1}^*$
- als $E = (E_1 | E_2)$ dan $L_E = L_{E_1} \cup L_{E_2}$

1.4 Niet-deterministische eindge toestandsautomaten

Definitie 1.6: Niet-deterministische eindige toestandsautomaat (NFA)

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
- $\delta \subseteq Q \times \Sigma_\epsilon \times Q$ is de overgangsrelatie van de automaat, wat kan worden voorgesteld in een transitietabel
- q_s is de starttoestand
- $F \subseteq Q$ is de verzameling eindtoestanden



Opmerking: In de literatuur wordt δ soms ook als een functie gedefinieerd die een toestand en een symbool afbeeldt op de verzameling van toestanden waarnaar een overgang mogelijk is:

$$\delta(q, a) = \{q' \in Q \mid (q, a, q') \in \delta\}.$$

We schrijven $p \xrightarrow{a} q$ om aan te geven dat (p, a, q) een overgang is in δ.

Definitie 1.7: Aanvaarden van strings in een NFA

Een string s wordt aanvaard door een NFA $(Q, \Sigma, \delta, q_s, F)$ indien er een sequentie $q_s = q_0 \xrightarrow{a_0} \dots \xrightarrow{a_{n-1}} q_n$ van overgangen bestaat met $q_n \in F$ zodat s de ϵ-compressie is van $a_0 \dots a_{n-1}$.

Dus: Voor toestanden p,q en string $w \in \Sigma^*$ schrijven we $p \overset{w}{\rightsquigarrow} q$ indien er een sequentie van overangen $p \xrightarrow{a_0} \dots \xrightarrow{a_{n-1}} q$ bestaat zodat w de ϵ-compressie is van $a_0 \dots a_{n-1}$.

Definitie 1.8: Taal L bepaald door een NFA M

Een taal L wordt bepaald door een NFA M, indien L de verzameling van strings is die M aanvaardt. We noteren de taal van M als L_M.

Definitie 1.9: Equivalentie van twee NFA's

Twee NFA's worden **equivalent** genoemd als ze dezelfde taal bepalen, m.a.w. elke equivalentieklassen van deze equivalentierelatie komt overeen met een taal.

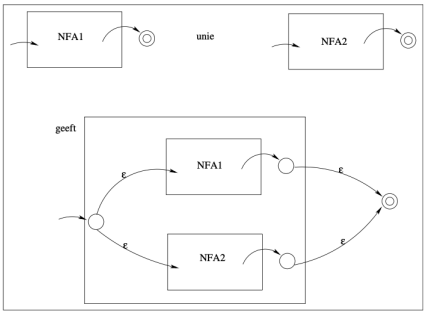
1.5 De algebra van NFA's

Eigenschap 1.3: Unie van twee NFA's

Gegeven: $i \in \{1,2\}: \ \text{NFA}_i = (Q_i, \Sigma, \delta_i, q_{s_i}, \{q_{f_i}\})$

De unie $\text{NFA}_1 \cup \text{NFA}_2$ is de $\text{NFA} = (Q, \Sigma, \delta, q_s, F)$ waarbij

- $Q = Q_1 \cup Q_2 \cup \{q_s, q_f\}$, $F = \{q_f\}$
- δ is gedefnieerd als:
  - $\forall x \in \Sigma : \delta(q_s, x) = \emptyset \ \wedge \ \delta(q_{f_i}, x) = \emptyset$
  - $\forall q \in Q_i \backslash \{q_{f_i}\}, \ x \in \Sigma_\epsilon : \ \delta(q, x) = \delta_i(q, x)$
  - $\delta(q_s, \epsilon) = \{q_{s_1}, q_{s_2}\}$
  - $\delta(q_{f_i}, \epsilon) = \{q_f\}$

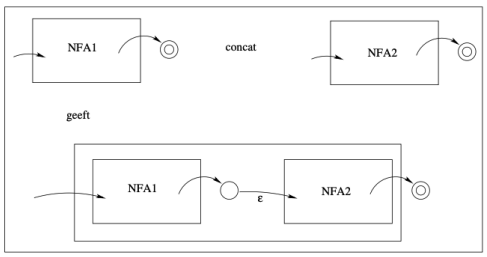


Eigenschap 1.4: Concatenatie van twee NFA's

Gegeven: $i \in \{1,2\}: \ \text{NFA}_i = (Q_i, \Sigma, \delta_i, q_{s_i}, \{q_{f_i}\})$

De concatenatie NFA_1NFA_2 is de $\text{NFA} = (Q, \Sigma, \delta, q_s, F)$ waarbij

- $Q = Q_1 \cup Q_2$, $q_s = q_{s_1}$, $F = \{q_{f_2}\}$
- δ is gedefnieerd als:
  - $\forall x \in \Sigma : \ \delta q_{f_1}, x = \emptyset$
  - $\forall q \in Q_i \backslash \{q_{f_1}\}, \ x \in \Sigma_\epsilon : \ \delta(q, x) = \delta_i(q, x)$
  - $\delta(q_{f_1}, \epsilon) = \{q_{s_2}\}$

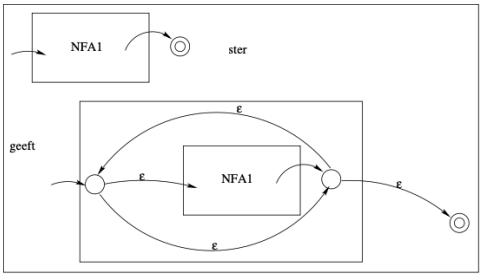


Eigenschap 1.5: Kleene ster van een NFA

Gegeven: $\text{NFA}_1 = (Q_1, \Sigma, \delta_1, q_{s_1}, \{q_{f_1}\})$

De Kleene ster NFA_1^* is de $\text{NFA} = (Q, \Sigma, \delta, q_s, F)$ waarbij

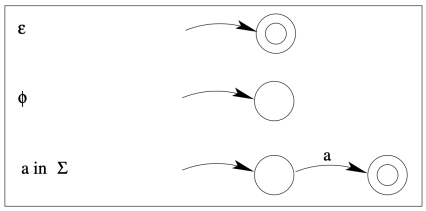
- $Q = Q_1 \cup \{q_s, q_f\}$, $F = \{q_f\}$
- δ is gedefnieerd als:
  - $\forall x \in \Sigma : \delta(q_s, x) = \emptyset \ \wedge \ \delta(q_{f_1}, x) = \emptyset$
  - $\forall q \in Q_1 \backslash \{q_{f_1}\}, \ x \in \Sigma_\epsilon : \ \delta(q, x) = \delta_1(q, x)$
  - $\delta(q_s, \epsilon) = \{q_{s_1}, q_{f_1}\}$
  - $\delta(q_{f_1}, \epsilon) = \{q_s, q_f\}$



1.6 Van RE naar NFA

Definitie 1.10: RE → NFA

We hebben alle ingrediënten om van een reguliere expressie RE een NFA te maken, en zodanig dat de $L_{RE} = L_{NFA}$. Vermits reguliere expressies inductief gedefinieerd zijn zullen we voor elk lijntje van die definitie een overeenkomstige NFA definiëren. De drie basisgevallen ϵ, ϕ en $a \in \Sigma$ zijn triviaal te modeleren als NFA. We illusteren hieronder:



De drie recursieve gevallen beschrijven we als volgt: laat E_1 en E_2 twee reguliere expressies zijn, dan is

- $NFA_{E_1E_2} = \text{concat}(NFA_{E_1}, NFA_{E_2})$
- $NFA_{E_1^*} = \text{ster}(NFA_{E_1})$
- $NFA_{E_1|E_2} = \text{unie}(NFA_{E_1}, NFA_{E_2})$

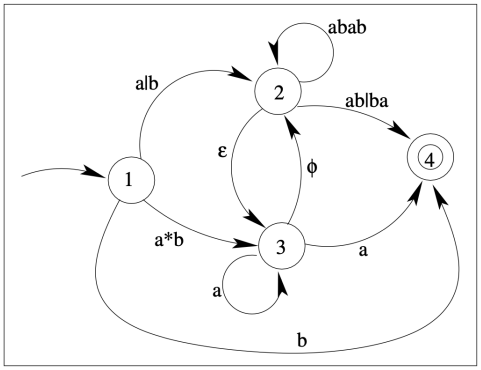
De constructie hierboven bewaart de taal, t.t.z. $L_{NFA_E} = L_E$.

1.7 Van NFA naar RE

Definitie 1.11: GNFA

Een **GNFA** is een eindige toestandsmachine met de volgende wijzigingen en beperkingen:

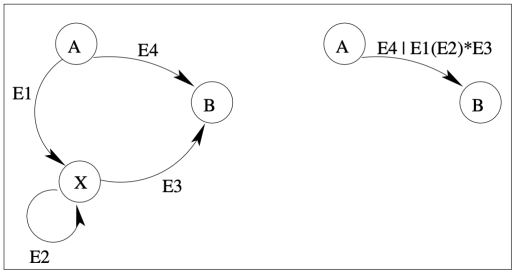
- er is slechts één eindtoestand en die is verschillend van de starttoestand
- vanuit de starttoestand vertrekt er juist één boog naar elke andere toestand; er komen geen bogen aan in de starttoestand
- in de eindtoestand komt juist één boog aan vanuit elke andere toestand; uit de eindtoestand vertrekken geen bogen
- voor paar p,q (let op: $p = q$ is geldig) van andere toestanden (geen start- of eindtoestand) is er juist één boog $p \longrightarrow q$ en één boog $q \longrightarrow p$.
- de bogen hebben als label een reguliere expressie



Algoritme 1.1: NFA → RE

1. Maak van de NFA een GNFA
   - Behoud alle toestanden en bogen van de NFA
   - Als er meerdere bogen zijn tussen twee toestanden gelabeld met symbolen $a_1 \dots a_n$ vervang deze door één boog met als label $a_1 | \dots | a_n$
   - Voer een nieuwe starttoestand in en een ϵ-boog naar de oude starttoestand
   - Voer een nieuwe eindtoestand in en ϵ-bogen vanuit elke oude eindtoestand
   - Voor elke boog die ontbreekt tussen twee toestanden om een GNFA te bekomen, voer een ϕ-boog in
2. Reduceer de GNFA:

   Kies een willekeurige toestand X verschillend van de start- of eindtoestand, ga naar stap 3 als dit niet mogelijk is. Voor elk paar toestanden A en B (let op: $A = B$ is geldig) verschillend van X bevat de GNFA een unieke boog $A \to B$ met label E_4, $A \to X$ met label E_1, $X \to X$ met label E_2 en $X \to B$ met label E_3. Vervang het label op de boog $A \to B$ door $E_4 | E_1 E_2^* E_3$. Doe dit voor alle keuzes voor A en B. Verwijder daarna de knoop X en herhaal.



3. Bepaal RE: de boog van de GNFA heeft als label de gezochte RE

Stelling 1.1: Taal herkend door een NFA

De taal herkend door de NFA is de taal van de berekende reguliere expressie, en dus bepalen de twee geziene formalismen NFA en RE precies dezlfde klasse van talen: de reguliere talen.

1.8 Deterministische eindige toestandsmachines

Definitie 1.12: Deterministische eindige toestandsmachines

Een NFA is een DFA indien δ geen ϵ-overgangen bevat en indien voor elke $p \in Q$ en elke $a \in \Sigma$ een unieke $q \in Q$ bestaat zodat $p \xrightarrow{a} q$. Het komt erop neer dat in een DFA, δ een totale functie $Q \times \Sigma \to Q$ is. Voor DFA's zullen a we de unieke toestand q zodat $p \xrightarrow{a} q$ dan ook noteren als $\delta(p, a)$.

Algoritme 1.2: NFA → DFA

Neem voor een willekeurige string $w \in \Sigma^*$ de verzameling $\{q \mid q_s \overset{w}{\rightsquigarrow} q\}$ van toestanden vereikbaar met w. We noteren deze verzameling als q'_w. Deze verzameling dient in een DFA een singleton te zijn, dit sugereert de volgende werkwijze: we bouwen een nieuwe automaat $(Q', \Sigma, \delta', q'_s, F')$ waarvan de toestandenovereenkomen met verzamelingen van toestanden van de NFA. We construeren die zodanig dat voor elke string w geldt dat $q'_s \overset{w}{\rightsquigarrow} q'_w$ (in de DFA), i.e. q'_w is de set van toestanden in de NFA die bereikbaar zijn vanaf de oorspronkelijke begintoestand q_s met w. We definiëren:

- Q' als de verzameling van alle deelverzamelingen q' van Q die gesloten zijn onder ϵ-bogen, dus
$$p \in q' \ \wedge \ p \xrightarrow{\epsilon} q \ \Rightarrow \ q \in q'$$
- $\delta' : Q' \times \Sigma \to Q'$ als
$$\delta'(q', a) = \{q \mid \exists p \in q' : \ p \overset{a}{\rightsquigarrow} q\}$$
- q'_s als de verzameling van q_s en toestanden die bereikbaar zijn vanuit q_s met ϵ-bogen, dus
$$q'_s = \{q_s, q \mid q_s \overset{\epsilon}{\rightsquigarrow} q\}$$
- F' als de verzameling van alle q' zodat $q' \cap F \neq \emptyset$, dus
$$F' = \{q' \in Q' \mid q' \cap F \neq \emptyset\}$$

Uit constructie volgt dat deze automaat $(Q', \Sigma, \delta', q'_s, F')$ een DFA is.

Stelling 1.2: DFA en NFA equivalentie

De geconstrueerde DFA $(Q', \Sigma, \delta', q'_s, F')$ is equivalent met de NFA $(Q, \Sigma, \delta, q_s, F)$.

Bewijs 1.1: DFA en NFA equivalentie

We moeten verifiëren dat
$$\forall w \in \Sigma^* : \ q'_s \overset{w}{\rightsquigarrow} F' \ \Leftrightarrow \ q_s \overset{w}{\rightsquigarrow} F$$
De essentie van dat bewijs is dat
$$\forall w \in \Sigma^* : \ q'_s \overset{w}{\rightsquigarrow} q' \text{ (in de DFA)} \ \Leftrightarrow \ q' = q_w = \{q \mid q_s \overset{w}{\rightsquigarrow} q\} \text{ (in de NFA)}$$
Dit is eenvoudig inductief te bewijzen gebruik makend van het feit dat $q' = q'_w \Rightarrow \delta'(q', a) = q'_{wa}$. Dan geldt dat de DFA een string w aanvaardt als voor de unieke toestand q' zodat $q'_s \overset{w}{\rightsquigarrow} q'$ geldt dat $q' \cap F \neq \emptyset \ \Leftrightarrow \ q'_w \cap F \neq \emptyset \ \Leftrightarrow \ q_s \overset{w}{\rightsquigarrow} F$. □

Stelling 1.3: Taal bepaald door een DFA

De taal bepaald door een DFA is een reguliere taal.

Eigenschap 1.6: Doorsnede, verschil en complement van DFA's

Gegeven: $i \in \{1,2\}: \ \text{DFA}_i = (Q_i, \Sigma, \delta_i, q_{s_i}, \{q_{f_i}\})$

We maken een generische product DFA $(Q, \Sigma, \delta, q_s, F)$ als volgt:

- $Q = Q_1 \times Q_2$
- $\delta(p \times q, x) = \delta_1(p, x) \times \delta_2(q, x)$
- $q_s = (q_{s_1}, q_{s_2})$
- Om tot een volledig definitie te komen, moeten we nog F bepalen:
  - $F = F_1 \times F_2$: de DFA is de doorsnede van de twee talen
  - $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$: de DFA is nu de unie van de twee talen
  - $\forall i \neq j \in [1,2]: \ F = F_i \times (Q_j \times F_j)$: de DFA bepaalt nu de strings die tot L_i behoren, maar niet tot L_j
  - $F = (Q_1 \backslash F_1) \times (Q_2 \backslash F_2)$: de DFA bepaalt nu de strings die tot geen van beide talen behoren.

Hieruit volgt ook dat de unie, doorsnede en complement van twee reguliere talen ook regulier zijn. Daaruit volgt ook dat het complement van een reguliere taal ook regulier is, want $\overline{L} = \Sigma^* \backslash L$.

Definitie 1.13: f-string

We noemen een string s een f-string vanuit q van de DFA indien $\delta^*(q, s) \in F$, t.t.z. indien er een pad is van q naar een toestand van F die s genereert. F-gelijke toestanden zijn dan toestanden met dezelfde f-strings.

Opmerking: $q \in F \Leftrightarrow \epsilon$ is een f-string vanuit q

Definitie 1.14: f-gelijk

Twee toestanden q_1, q_2 zijn f-gelijk indien

$$\{s \in \Sigma^* \mid \delta^*(q_1, s) \in F\} = \{s \in \Sigma^* \mid \delta^*(q_2, s) \in F\}$$

In woorden, als q_1 en q_2 exact dezelfde f-strings hebben.

Eigenschap 1.7: f-gelijk

- De relatie f-gelijk is een equivalentie-relatie.
- Als p, q f-gelijk zijn dan geldt voor elk symbool a dat $\delta(p, a)$ en $\delta(q, a)$ ook f-gelijk zijn.
- Als p, q f-gelijk zijn dan geldt $p \in F \Leftrightarrow q \in F$.

Bewijs 1.2: Eigenschappen van de f-gelijk relatie

- Het is triviaal om te bewijzen dat f-gelijkheid een equivalentie-relatie is. Dit kan je doen door de reflexiviteit, symmetrie en transitiviteit van de relatie na te gaan.
- Veronderstel dat p, q f-gelijk zijn en veronderstel voor een willekeurig symbool a dat $\delta(p, a) = p', \delta(q, a) = q'$. De f-strings van p en q zijn gelijk, en dus ook hun f-strings van de vorm as. De f-strings van p' zijn de strings s zodat as een f-string is van p. Hetzelfde geldt voor q'. Bijgevolg hebben p', q' dezelfde f-strings en zijn ze dus f-gelijk.
- Als p en q f-gelijk zijn, en $p \in F$ dan is ϵ een f-string van p en dus ook van q. Aangezien er in een DFA geen ϵ-bogen zijn, is $q \in F$. Hetzelfde geldt in de andere richting.

□

Definitie 1.15: Minimale DFA

De minimale DFA is de DFA met het minimale aantal toestanden die dezelfde taal bepaalt als de oorspronkelijke DFA. Als we vertrekken van een DFA $(Q, \Sigma, \delta, q_s, F)$, dan bestaat de minimale DFA uit $(\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q_s}, \tilde{F})$ waarbij

- $\tilde{Q} = \{Q_1, \ldots, Q_n\}$ waarbij de Q_i de equivalentie-klasse is van de relatie f-gelijk
- $\tilde{\delta}(Q_i, a) = Q_j$ indien een $q \in Q_i$ bestaat zodat $\delta(q, a) \in Q_j$
- $\tilde{q_s}$ is de equivalentie-klasse Q_i zodat $q_s \in Q_i$
- $\tilde{F}$ is de verzameling van Q_i waarvoor geldt dat $Q_i \subseteq F$.

Stelling 1.4: DFA_{min}

Een DFA_{min} is een DFA, equivalent met DFA, en alle toestanden zijn f-verschillend.

Bewijs 1.3: DFA_{min}

DFA_{min} is een DFA omdat f-gelijkheid van toestanden p en q de f-gelijkheid van $\delta(p, a)$ en $\delta(q, a)$ impliceert. Het gevolg is dat verschillende bogen met hetzelfde symbool vanuit f-gelijke p en q versmelten. Om aan te tonen dat DFA en DFA_{min} equivalent zijn, is de essentiële eigenschap dat elke equivalentie-klasse Q_i en elk element $p \in Q_i$ dezelfde f-strings heeft. Dus heeft $\tilde{q_s}$ dezelfde f-strings als q_s, en deze verzameling is dus de taal die beide DFA's bepalen. Aantonen dat een string w een f-string is van Q_i als en slechts als w een f-string is van $q \in Q_i$ gebeurt door inductie op de lengte van w, gebruikmakend van het feit dat voor elke Q_i, Q_j

$$p \in Q_i,\ q \in Q_j,\ \forall a \in \Sigma :\ Q_i \xrightarrow{a} Q_j \ \Leftrightarrow\ p \xrightarrow{a} q$$

Tenslotte, twee verschillende toestanden Q_i, Q_j bevatten f-verschillende toestanden. Aangezien de f-strings van Q_i en Q_j die van hun elementen zijn, zijn ze f-verschillend. □

Algoritme 1.3: DFA $\rightarrow$ DFA_{min}

We gaan ervoor zorgen dat de alle toestanden van de DFA f-verschillend zijn, ofwel alle f-gelijke toestanden samenvoegen tot één toestand. We beginnen uit Q^2: dit komt neer op de veronderstelling dat alle paren f-gelijk zijn. In eerste instantie schrappen we koppels (p, q) waarvan ofwel p ofwel q behoort tot F maar niet allebei want deze koppels worden uitgesloten door de 'enkel als' regel. Zij zijn zeker niet f-gelijk aangezien 1 ervan ϵ afleidt en de andere niet. In tweede instantie schrappen we koppels (p, q) waarvoor een symbool a bestaat zodat $((\delta(p, a)), (\delta(q, a)))$ niet behoort tot de huidige set want ook die zijn in tegenspraak met de regel. We gaan hiermee door totdat we geen koppels meer kunnen schrappen en dus de verzameling van toestanden f-verschillend is.

Stelling 1.5: DFA_{min}

Als een DFA $N = (Q_1, \Sigma, \delta_1, q_s, F_1)$ een DFA is zonder onbereikbare toestanden en waarin elke twee toestanden f-verschillend zijn, dan bestaat er geen machine met strikt minder toestanden die dezelfde taal bepaalt. De DFA N is dan een minimale DFA of DFA_{min} voor deze taal.

Bewijs 1.4: DFA_{min}

Veronderstel dat $Q_1 = \{q_s, q_1, \ldots, q_n\}$ waarbij q_s de starttoestand is, stel dat $N_2 = (Q_2, \Sigma, \delta_2, q_s, F_2)$ een DFA is met minder toestanden dan N. Vermits in N elke toestand bereikbaar is, bestaan er strings $\forall i \in \mathbb{N}_0^+ : s_i$ zodanig dat $\delta_1^*(q_s, s_i) = q_i$. Vermits N_2 minder toestanden heeft moet voor een $i \neq j : \delta_2^*(p_s, s_i) = \delta_2^*(p_s, s_j)$. Vermits q_i en q_j f-verschillend zijn, is er een string s zodat $\delta_1^*(q_i, s) \in F_1$ en $\delta_1^*(q_j, s) \notin F_1$ of omgekeerd. Dus ook $\delta_1^*(q_s, s_i s) \in F_1$ en $\delta_1^*(q_s, s_j s) \notin F_1$ of omgekeerd. Dit betekent dat DFA_1 van de strings $s_i s$ en $s_j s$ er juist één accepteert. Maar N_2 zal beide strings $s_i s$ en $s_j s$ accepteren of geen van beiden, aangezien het parsen van s_i en s_j naar dezelfde node leidt, waarna hetzelfde pad gevolgd wordt om v te parsen. Dus kunnen de DFA's N en N_2 niet dezelfde taal bepalen. □

Definitie 1.16: DFA isomorfisme

Een DFA $N_1 = (Q_1, \Sigma, \delta_1, q_{s_1}, F_1)$ is **isomorf** met een DFA $N_2 = (Q_2, \Sigma, \delta_2, q_{s_2}, F_2)$ als er een bijectie $b : Q_1 \rightarrow Q_2$ bestaat zodanig dat

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

Twee isomorfe DFA's bepalen dus dezelfde taal.

Opmerking: een minimale DFA is dus altijd uniek op isomorfisme na, sinds elke twee minimale DFA's van een DFA (op een isomorfisme na) dezelfde componenten hebben.

1.9 Myhill-Nerode relaties op Σ^*

Definitie 1.17: Fijnheid van partities

Een partitie P_1 is **fijner** dan een partitie P_2 indien

$$\forall x \in P_1,\ \exists y \in P_2 :\ x \subseteq y$$

Het omgekeerde van fijn is **grof**.

Definitie 1.18: $\sim_{\text{DFA}}$

Voor een DFA $N = (Q, \Sigma, \delta, q_s, F)$ definiëren we de relatie $\sim_{\text{DFA}}$ op Σ^* als volgt:

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

In woorden: er geldt $x \sim_{\text{DFA}} y$ als en slechts als het parsen van x en y vanuit q_s leidt tot dezelfde toestand q, dus $x, y \in \text{reach}(q)$.

Eigenschap 1.8: $\sim_{\text{DFA}}$

- Rechts congruentie van $\sim$: $\forall x, y \in \Sigma^*,\ a \in \Sigma :\ x \sim y \ \Rightarrow\ xa \sim ya$
- het aantal equivalentieklassen van $\sim$ is eindig, m.a.w. $\sim$ heeft een eindige index
- $\sim$ verfijnt $\sim_L$, of: $x \sim y \ \Rightarrow\ x \sim_L y$

Definitie 1.19: Myhill-Nerode relatie

Een equivalentierelatie $\sim$ tussen strings is een **Myhill-Nerode relatie** MN(L) voor een taal L als de equivalentieklasse voldoet aan **bovenstaande** eigenschappen.

Stelling 1.6: $\text{DFA}^{\text{L}}_{\sim}$

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

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

Bewijs 1.5: $\mathbf{DFA}^{\mathbf{L}}_{\sim}$

Dat δ goed gedefinieerd is, kan je bewijzen door gebruik te maken van de rechtse congruentie van $\sim$. Verder zijn alle ingrediënten van de DFA duidelijk, in het bijzonder ook dat Q slechts een eindig aantal toestanden bevat. We moeten nog bewijzen dat $L_{\mathrm{DFA}^{\mathrm{L}}_{\sim}} = L$:

$$x \in L_{\mathrm{DFA}^{\mathrm{L}}_{\sim}} \;\Leftrightarrow\; \delta^*(q_s, x) \in F \;\Leftrightarrow\; x_\sim \in F \;\Leftrightarrow\; x \in L$$

De middelste overgang bekom je door met inductie op de lengte van de string x te bewijzen dat $\delta^*(\epsilon_\sim, x) = (x)_\sim$. □

Stelling 1.7: $\sim_{\mathbf{DFA}}$ en $\sim$ zijn elkaars inverse

Voor elke taal L geldt dat de functie die DFA's van L afbeeldt op de de overeenkomstige MN(L)-relatie $\sim_{\mathrm{DFA}}$ en de functie die MN(L)-relaties $\sim$ afbeeldt op de overeenkomstige $\mathrm{DFA}^{\mathrm{L}}_{\sim}$, elkaars inversen zijn op een DFA-isomorfisme na.

Stelling 1.8: Infimumrelatie van Myhill-Nerode relaties

Als E een niet lege verzameling van MN(L) relaties is, dan is ook het infimum $\sim_{\mathrm{inf}}$ van E een MN(L) relatie. $\sim_{\mathrm{inf}}$ is de transitieve sluiting van de unie van E. Dit betekent dat

$$x \sim_{\mathrm{inf}} y \;\Leftrightarrow\; i \in [0, n-1],\ \sim_i \in E :\ x = x_0 \sim_0 x_1 \sim_1 \ldots \sim_{n-1} x_n = y$$

Bewijs 1.6: Infimumrelatie van Myhill-Nerode relaties

- **Eindigheid:** $\sim_{\mathrm{inf}}$ is een superset van elke willekeurige $\sim \in E$ en heeft dus minder equivalentieklassen dan $\sim$. Elke $\sim$ heeft slechts een eindig aantal equivalentieklasse, zodoende ook $\sim_{\mathrm{inf}}$.
- **Rechts congruentie:** Stel $x \sim_{\mathrm{inf}} y$ dan bestaat de sequentie

  $$x = x_0 \sim_0 x_1 \sim_1 \ldots \sim_{n-1} x_n = y$$

  Aangezien elke $\sim_i$ rechts congruent is, geldt voor elke $a \in \Sigma$ dat

  $$xa = x_0a \sim_0 x_1a \sim_1 \ldots \sim_{n-1} x_na = ya$$

  Bijgevolg geldt dat $xa \sim_{\mathrm{inf}} ya$ en dus dat $\sim_{\mathrm{inf}}$ rechts congruent is.
- **Verfijnen van $\sim_L$:** Stel $x \sim_{\mathrm{inf}} y$ zodat er een sequentie bestaat

  $$x = x_0 \sim_0 x_1 \sim_1 \ldots \sim_{n-1} x_n = y$$

  Aangezien elke $\sim_i$ een verfijning is van $\sim_L$, geldt

  $$x_0 \in L \Leftrightarrow x_1 \in L \Leftrightarrow \ldots \Leftrightarrow x_n \in L$$

  We bekomen $x \in L \Leftrightarrow y \in L$ en dus dat $\sim_{\mathrm{inf}}$ een verfijning is van $\sim_L$.

□

Stelling 1.9: MN(L) toebehorend aan een mininale DFA

De DFA van $\sim_{\text{inf}}$, $\text{DFA}_{\sim_{\text{inf}}}$, is een minimale DFA_{min} van L. Elke minimale DFA is van L is isomorf met mekander, en dus ook met $\text{DFA}_{\sim_{\text{inf}}}$.

Bewijs 1.7: MN(L) toebehorend aan een mininale DFA

Voor elke MN(L) relatie $\sim$ geldt ofwel dat $\sim = \sim_{\text{inf}}$, ofwel bestaan x, y zodat $x \nsim y$ maar $x \sim_{\text{inf}} y$, en dan heeft $\sim$ strikt meer equivalentieklassen dan $\sim_{\text{inf}}$. Uit de overeenkomst tussen DFA's en MN(l) relaties volgt dat elke DFA die L bepaalt, ofwel isomorf is aan DFA_{sup}, de DFA van $\sim_{\text{inf}}$, ofwel heeft hij strikt meer toestanden dan DFA_{sup}. Dus, $\text{DFA}_{\sim_{\text{inf}}}$ is een minimale DFA van L waarmee elke minimale DFA van L isomorf is. □

Eigenschap 1.9: MN(L) toebehorend aan een mininale DFA

Het kan nu ook bewezen worden dat de MN(L) relatie die hoort bij de minimale DFA voldoet aan:

$$x \sim_{\text{inf}} y \;\Leftrightarrow\; \forall s \in \Sigma^* :\; (xs \in L \Leftrightarrow ys \in L)$$

Het is een vorm van f-gelijkheid gedefinieerd op strings in plaats van op toestanden.

Stelling 1.10: Stelling van Myhill-Nerode

Laat $L \subseteq \Sigma^*$ een taal zijn over Σ. De volgende drie uitspraken zijn dan equivalent:

$\Leftrightarrow$ L is regulier

$\Leftrightarrow$ er bestaat een Myhill-Nerode relatie voor L

$\Leftrightarrow$ definieer $\sim$ op Σ^* als volgt:

$$x \sim y \;\Leftrightarrow\; \forall s \in \Sigma^* :\; (xs \in L \Leftrightarrow ys \in L);$$

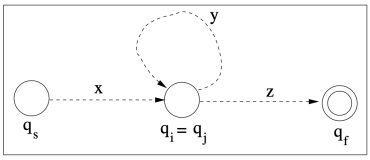
de relatie $\sim$ heeft een eindige index

1.10 Pompend lemma voor reguliere talen

Stelling 1.11: Pompend lemma voor reguliere talen

Voor een reguliere taal L bestaan een pomplengte d, zodanig dat als $s \in L$ en $|s| \geq d$, dan bestaat er een verdeling van s in stukken x, y en z zodanig dat $s = xyz$ en

- $\forall i \in \mathbb{N}_0^+ :\; xy^i z \in L$
- $|y| > 0$
- $|xy| \leq d$



Bewijs 1.8: Pompend lemma voor reguliere talen

Neem een DFA die L bepaalt, neem $d = \#Q$ en neem een willekeurige $s = a_1 \dots a_n \in L$ met $n \geq d$. Beschouw de accepterende sequentie

$$q_s = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \in F$$

Deze rij van toestanden heeft lengte $n+1$, wat strikt groter is dan d. Neem de eerste $d+1$ toestanden van de deze rij, dus $q_0, \dots, q_d$. Er zijn maar d verschillende toestanden, dus er zijn twee gelijke toestanden. Stel dat $q_i = q_j$ met $0 \leq i < j \leq d$, dan nemen we $x = a_1 \dots a_i$, $y = a_{i+1} \dots a_j$ en z de rest van de string. Alles volgt nu direct, zie desnoods te illustratie in de stelling. □

1.11 Contextvrije talen en hun grammatica

Definitie 1.20: Contextvrije grammatica - CFG

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

- V een eindige verzameling is van niet-eindsymbolen of **variabelen**
- Σ een eindige alfabet is van eindsymbolen of **terminals**, disjunct met V
- R is een eindige verzameling van regels of **producties**: een regel is een koppel van één niet-eindsymbool en een strings van elementen uit $V \cup \Sigma_\epsilon$. We schrijven $u \to v$ voor een regel (u, v).
- $S \in V$ is het **startsymbool**

Definitie 1.21: Taal bepaald door een CFG

De taal L_{CFG} bepaald door een CFG (V, Σ, R, S) is de verzameling strings s over Σ zodanig dat $S \Rightarrow^* s$. In formulevorm:

$$L = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

Definitie 1.22: Afleiding m.b.v. een CFG

Gegeven een CFG (V, Σ, R, S). Een string f over $V \cup \Sigma_\epsilon$ wordt afgeleid uit een string b over $V \cup \Sigma_\epsilon$ m.b.v. de CFG als er een eindige rij strings $s_0, \dots, s_n$ bestaat zodanig dat

- $s_0 = b$
- $s_n = f$
- $\forall i \in [0, n-1] : \ s_i \to s_{i+1}$

We noteren: $s_i \Rightarrow s_{i+1}$ en $b \Rightarrow^* f$.

Eigenschap 1.10: Implementaties van afleidingen m.b.v. een CFG

- **diepte-eerst**: bij elke overgang i is het vervangen symbool X de grootste index heeft van alle niet-terminaal symbool in s_i. Dit is een **niet-deterministische** strategie.
- **links-eerst**: bij elke overgang i is het vervangen symbool X het meest linkse niet-terminaal symbool in s_i. Een links-eerst strategie is ook een diepte-eerst strategie, maar niet noodzakelijk omgekeerd.
- **breedte-eerst**: bij elke overgang i is het vervangen symbool X de kleinste index heeft van alle niet-terminaal symbool in s_i. Dit is een **niet-deterministische** strategie.
  - breedte-eerst, linkst eerst: bij elke overgang i is het vervangen symbool X het meest linkse symbool met kleinste index heeft van alle niet-terminaal symbool in s_i.

Eigenschap 1.11: Ambiguïteit van een CFG

Een CFG is ambigu indien er strings bestaan die meerdere meest-linkse parsings bezitten.

Definitie 1.23: Equivalente CFG's

Twee contextvrije grammatica's CFG_1 en CFG_2 zijn **equivalent** als ze dezelfde taal bepalen, en dus $L_{\text{CFG}_1} = L_{\text{CFG}_2}$.

Definitie 1.24: Contextvrije taal - CFL

Een taal L is **contextvrij** indien er een CFG bestaat die L bepaalt, en dus $L = L_{\text{CFG}}$.

Opmerking: niet elke CFL heeft een niet-ambigue CFG

Definitie 1.25: Chomsky normaalvorm

Een CFG (V, Σ, R, S) heeft de **Chomsky normaalvorm** als elke regel één van de volgende vormen heeft:

- $A, B, C \in V : \ A \to BC$
- $A \in V, \ \alpha \in \Sigma : \ A \to \alpha$
- $S \to \epsilon$

Stelling 1.12: Equivalente Chomsky normaalvorm

Voor elke CFG (V, Σ, R, S) bestaat er een equivalentie CFG in Chomsky normaalvorm.

Bewijs 1.9: Equivalente Chomsky normaalvorm

We vertrekken van een willekeurige CFG en transformeren hem terwijl we equivalentie bewaren naar Chomsky Normaalvorm.

1. Vervang alle voorkomens van S door een nieuw symbool X, en daarna voeg de regel $S \to X$ toe. Zo komt S enkel links voor. Zo wordt 1 van de voorwaarden voldaan. Deze stap is evident equivalentiebewarend.

2. Neem de verzameling van alle regels. Itereer de volgende operatie zo lang tot een fixpunt bereikt wordt:

   - Selecteer een regel $A \to \epsilon$ en een regel $B \to \alpha A \beta$.
   - Voeg de regel $B \to \alpha\beta$ toe.

   Deze stap is equivalentiebewarend. Daarvoor moet aangetoond worden dat elke iteratie equivalentiebewarend is. Kortom, als $B \to \alpha A \beta$ toegevoegd wordt, dan kan een string geschreven worden met die regel als en slechts als de strings geschreven kan worden zonder deze regel. Het is evident omdat het herschrijven met $B \to \alpha\beta$ kan gesimuleerd worden met $B \to \alpha a \beta$ en $A \to \epsilon$. Tenslotte: verwijder alle regels $A \to \epsilon$ voor $A \neq S$. Ook deze stap is equivalentiebewarend. Neem een parsing boom $S \to \ldots$; de wortel is S, elke node is gelabeld met een symbool A, elke node die geen blad is eheft een geordende rij kinderen gelabeld $BC \ldots D$ zodat $A \to BC \ldots D$ een regel is, en de bladeren van de bomen vormen een sequentie van terminale symbolen. Dan is het vrij duidelijk dat elke string van de CFL zo'n parsing boom heeft. Ook dat deze boom kan hervormd worden tot een boom die gebruik maakt van de toegevoegde regels en geen gebruik maakt van regels $A \to \epsilon$.

3. Nu willen we afgeraken van de regels van de vorm $A \to B$. Voor een regel van de vorm $\mathcal{E} = A \to B$ en een regel van de vorm $\mathcal{R} = B \to \gamma$, definieer de regel $\mathcal{U}(\mathcal{E}, \mathcal{R}) = A \to \gamma$. Zolang er regels van de vorm $\mathcal{E} = A \to B$ zijn (waaron B ook een niet-terminal is) en regels van de vorm $\mathcal{R} = B \to \gamma$ zijn, voeg de regel $\mathcal{U}(\mathcal{E}, \mathcal{R})$ toe. Nadat dit eindigt, verwijderen we uit de bekomen grammatica alle regels van de vorm $A \to B$. Deze stap is equivalentiebewarend. Neem een string s die kan afgeleid worden met de regels van de vorm $A \to B$. Dan kan s ook afgeleid worden zonder deze regels.

4. We hebben nu nog drie soorten regels te behandelen:

   (a) $A \to \gamma$ waar γ uit juist twee niet-eindsymbolen bestaat.

   (b) $A \to \gamma$ waar γ uit minstens twee symbolen bevat: vervang elke terminal a door een niet-terminaal A_a en voeg de regel $A_a \to a$ toe.

   (c) eventueel $S \to \epsilon$: die mag blijven.

5. De regels van de vorm
   $$n > 2 : A \to X_1 X_2 \ldots X_n$$
   vervang je door
   $$A \to X_1 Y_1,\ Y_1 \to X_2 Y_2, \ldots,\ Y_{n-2} \to X_{n-1} X_n$$

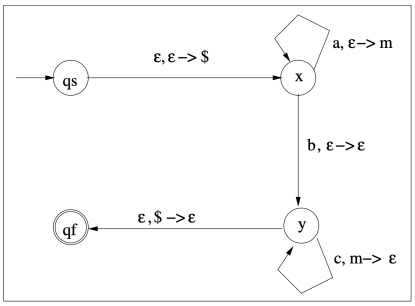
Hierbij is het duidelijk dat bij de transformatie naar de Chomsky Normaalvorm de grammatica equivalent blijft. □

1.12 Push-down automaat

Definitie 1.26: Push-down automaat

Een push-down automaat PDA $(Q, \Sigma, \Gamma, \delta, q_s, F)$ is een automaat waarbij

- Q is een eindige verzameling toestanden
- Σ is een eindig inputalfabet
- Γ is een eindig stapelalfabet
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$
- q_s is de starttoestand
- $F \subseteq Q$ is een verzameling eindtoestanden



Opmerking: de bedoeling van een label zoals $\alpha, \beta \to \gamma$ op een boog is

- indien α het eerste symbool is van de huidige string
- en β staat op de top van de stapel
- volg dan de boog en
  - verwijder α van de string
  - verwijder β van de string
  - zet γ op de stapel

waarbij $\alpha, \beta, \gamma \in \Sigma_\epsilon$.

Definitie 1.27: Aanvaarding van een string s door een PDA

Een string s wordt aanvaard door een PDA indien s kan worden opgesplitst in delen $i \in [1, m] :\ w_i \in \Sigma_\epsilon$, er toestanden $j \in [0, n] :\ q_j$ zijn en stacks $k \in [0, m] :\ \text{stack}_k \in \Gamma^*$, zodanig dat

- $\text{stack}_0 = \epsilon$: de stack is leeg in het begin
- $q_0 = q_s$: we vertrekken in de begintoestand
- $q_m \in F$: we komen aan in een eindtoestand met een lege string
- $x, y \in \Gamma_\epsilon :\ (q_{i+1}, y) \in \delta(q_i, w_{i+1}, x)$ en $x, y \in \Gamma_\epsilon,\ t \in \Gamma^* :\ \text{stack}_i = xt,\ \text{stack}_{i+1} = yt$

Definitie 1.28: Taal bepaald door een PDA

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

1.13 Equivalentie van CFG en PDA

Stelling 1.13: Equivalentie van CFG en PDA

Elke push-down automaat bepaalt een contextvrije taal en elke contextvrije taal wordt bepaald door een push-down automaat. We onderscheiden de twee gevallen:

- CFG $\rightarrow$ PDA: We kunnen een PDA bouwen voor een CFG met drie toestanden q_s, x, q_f waarbij q_s de starttoestand is, q_f de unieke eindtoestand is en x een toestand is die enkel dient om de stack te legen. De transitie bestaat uit 4 soorten overgangen:
  - $\delta(q_s, \epsilon, \epsilon) = (x, S\$)$, met S het startsymbool van de CFG,
  - $\delta(x, \epsilon, X) = (x, \gamma)$, voor elke regel $X \rightarrow \gamma$ van de CFG,
  - $\delta(x, A, A) = (x, \epsilon)$ voor elk terminaal symbool A,
  - $\delta(x, \epsilon, \$) = (q_f, \epsilon)$.
- PDA $\rightarrow$ CFG: We kunnen een CFG (V, Σ, R, S) maken uit een PDA $(Q, \Sigma, \Gamma, \delta, q_s, F)$ waarbij
  - $\forall p, q \in Q: \ V = A_{p,q}$
  - $S = A_{q_s, q_f}$
  - R bestaat uit 3 delen:
    1. $\forall p \in Q : A_{p,p} \rightarrow \epsilon$
    2. $\forall p, q, r \in Q: \ A_{p,q} \rightarrow A_{p,r} A_{r,q}$
    3. $\forall p, q, r, s \in Q, a, b \in \Sigma_\epsilon, t \in \Gamma, (r, t) \in \delta(p, a, \epsilon), (q, \epsilon) \in \delta(s, b, t): \ A_{p,q} \rightarrow a A_{r,s} b$

Eigenschap 1.12: Equivalentie van CFG en PDA

- als een taal door een PDA wordt bepaald, dan is die contextvrij.
- elke reguliere taal is contextvrij, want elke NFA is een PDA waarbij de stapel niet gebruikt wordt.

1.14 Pompend lemma voor contextvrije talen

Stelling 1.14: Pompend lemma voor contextvrije talen

Voor een contextvrije taal L bestaan een getal p (de pomplengte) zodanig dat elke string s van L met lengte strikt groter dan p kan opgedeeld worden in 5 stukken $u, v, x, y, z \in \Sigma^*$ zodanig dat $s = uvxyz$

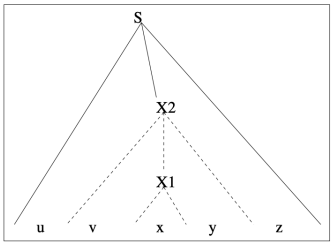
- $\forall i \in \mathbb{N}_0^+: \ uv^i xy^i z \in L$
- $|vy| > 0$
- $|vxy| \leq p$

Bewijs 1.10: Pompend lemma voor contextvrije talen

Als L eindig is dan is de stelling triviaal voldaan. Immers, dan heeft L een langste string s; neem $p = |s|$. Dan zijn er geen strings met lengte strikt groter dan p, en dan is de stelling triviaal voldaan. Veronderstel dus dat L oneindig is en bijgevolg strings van willekeurige lengte bevat en dat de CFG voor L in Chomsky normaalvorm is, dit is makkelijker en immers elke CFG heeft een equivalente CFG in Chomsky normaalvorm. Dus, elke regel heeft nu ofwel twee ofwel nul niet-terminalen aan de rechterkant, en het startsymbool S komt niet rechts voor. Laat het aantal niet-eindsymbolen in de CFG gelijk aan n zijn. We willen nu aantonen dat

- er een pomplengte p bestaat zodat de parse tree van elke string van lengte strikt groter dan p, een tak bevat met een herhaald niet-terminaal symbool X;
- als de parse tree van s een tak bevat met een herhaald niet-terminaal symbool X, dan is s van de vorm $uvxyz$ zodat s kan gepompt worden.

Voor de eerste stap gebruiken we het duivenhokprincipe, maar nu op de parse tree voor voldoende lange strings. De grootste parse tree van de CFG zonder takken met herhaalde niet-terminalen, bevat enkel paden met maximaal n niet-terminalen. Aangezien elke niet-terminale node hoogstens 2 kinderen heeft, bevat een dergelijke boom hoogstens 2^{n-1} bladeren. De string bevat evenveel symbolen als er bladeren zijn, dus hoogstens 2^{n-1}. Neem de pomp lengte $p = 2^{n-1}$. Zij s een string in L van meer dan p symbolen. De parse tree bevat dus een tak met meer dan n niet-eindsymbolen. We selecteren een langste pad in deze parse tree (mogelijks bestaat er meer dan 1 langste pad). Dit pad bevat meer dan n niet-terminalen en dus wordt een niet-terminaal herhaald. Zij X het herhaalde symbool op dat pad met het laagste voorlaatste voorkomen. Het laagste voorkomen van X duiden we aan door X_1 en het voorlaatste voorkomen door X_2. We kunnen nu uit die parse tree (zie figuur) een afleiding construeren waarvan we enkel wat tussenstappen laten zien



$$u, v, x, y, z \in \Sigma^* : \; S \Rightarrow^* uX_2z \Rightarrow^* uvX_1yz \Rightarrow^* uvxyz$$

waarbij v en y niet tegelijkertijd leeg zijn. Vermits dit een geldige afleiding is, is

$$S \Rightarrow^* uX_2z \Rightarrow^* uxz$$

dat ook en ook

$$S \Rightarrow^* uXz \Rightarrow^* uvXyz \Rightarrow^* uvvxyyz$$

is er eentje en ..., hierbij zijn de eerste twee voorwaarden bewezen van de stelling.

We besluiten nu ook de laatste: vxy wordt afgeleid door de deelboom onder X_2. Het deel van het geselecteerde pad onder X_2 is een langste pad van deze deelboom. Dit deelpad is hoogstens n lang (het bestaat enkel uit niet-eindsymbolen, bevat niet S, en bevat exact 1 herhaald symbool X). Het aantal bladeren van deze deelboom, en dus de lengte van vxy is dus hoogstens $2^{n-1} \leq p$, waarbij de laatste voorwaarde ook bewezen is.

□

2 Talen en berekenbaarheid

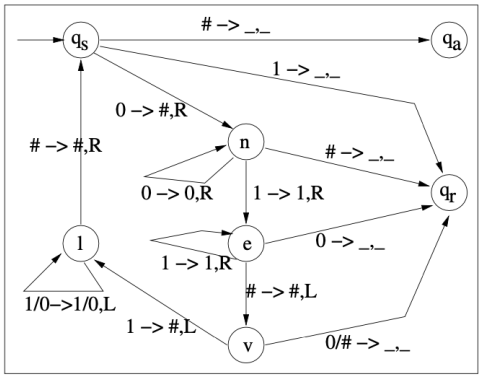
2.1 De Turingmachine als herkenner en beslisser

Definitie 2.1: Turingmachine

Een **Turingmachine** is een 7-tal $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ waarbij Q, Σ, Γ eindige verzamelingen zijn en

- Q is een verzameling toestanden
- Σ is het input alfabet dat # niet bevat
- Γ is het tape alfabet waarbij $\Sigma \subset \Gamma$ en $\# \in \Gamma$
- q_s is de starttoestand
- q_a is de accepterende eindtoestand
- q_r is de verwerpende eindtoestand, verschillend van q_a
- δ is de transitiefunctie: een totale functie met signatuur

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



Definitie 2.2: Turing-herkenbare taal

Een taal L is **Turing-herkenbaar** als er een Turingmachine TM bestaat zodanig dat

$$L = L_{\text{TM}}.$$

Opmerking: ∞_{TM} is niet leeg: voor elke string niet in L gaat de machine in een lus.

Definitie 2.3: Turing-beslisbare taal

Een taal L is **Turing-beslisbaar** als er een Turingmachine TM bestaat zodanig dat

$$L = L_{\text{TM}} \text{ en } \infty_{\text{TM}} = \emptyset.$$

Definitie 2.4: Co-herkenbaar/co-beslisbaar

Een taal L is co-herkenbaar/co-beslisbaar als $\overline{L}$ herkenbaar/beslisbaar is.

Stelling 2.1: Turing-beslisbaarheid en Turing-herkenbaarheid

1. Als een taal L beslisbaar is, dan is L co-beslisbaar.
2. Als een taal L herkenbaar en co-herkenbaar is, dan is L beslisbaar.
3. Er bestaat een taal die niet herkenbaar is.

Bewijs 2.1: Turing-beslisbaarheid en Turing-herkenbaarheid

1. Verwissel de rol van q_a en q_r in de Turingmachine die L beslist.
2. Laat M_1 de machine zijn die L herkent, en M_2 de machine die $\overline{L}$ herkent. De idee is nu dat we M_1 en M_2 samen laten lopen als een nieuwe machine M, in parallel: zodra M_1 accepteert, dan accepteert M, en zodra M_2 accepteert, dan verwerpt M. M_1 en M_2 kunnen niet samen accepteren, en voor elke string zal minstens één van de machines M_1 en M_2 stoppen p in zijn aanvaardende toestand. M beslist L.
3. Het bewijs steunt op het begrip kardinaliteit: we weten van vroeger dat het aantal Turingmachines aftelbaar oneindig is. We weten ook dat elke Turingmachine juist één taal herkent. En tenslotte weten we ook dat het aantal talen niet-aftelbaar oneindig is, want de verzameling talen is $\mathcal{P}(\Sigma^*)$. Bijgevolg bestaat een niet-herkenbare taal. In feite is daarmee zelfs bewezen dat er niet-aftelbaar veel niet-herkenbare talen bestaan. Meer nog: bijna alle talen zijn niet-herkenbaar!

□

2.2 Encodering

Definitie 2.5: Encodering

Een encodering is een mapping van objecten in Σ naar strings in Σ^*. Encoderingen zijn **redelijk**:

- Elk object moet minstens 1 encodering in Σ^* hebben.
- Elke encodering moet volledig het object dat het encodeert bepalen.
- De nodige operaties moeten berekenbaar zijn uit de encodering van de objecten.
- Een redelijke encodering introduceert geen extra informatie.

Definitie 2.6: Universele Turingmachine

De Universele Turingmachine kan elke TM simuleren, als die maar geëncodeerd wordt zoals nodig: we kunnen de UTM zelfs laten beginnen met controleren of wat op de banden staat wel een geldige encodering is.

2.3 Het Halting-probleem

Definitie 2.7: Acceptatieprobleem

Stel een Turingmachine M voor met een input w. De **acceptatietaal** $\mathrm{A_{TM}}$ is

$$\mathrm{A_{TM}} = \{\langle M, s\rangle \mid M \text{ is een Turingmachine en } s \in L_{\mathrm{M}}\}$$

zodanig dat de Turingmachine A de taal **beslist**. Dit is een verwant probleem aan het Halting-probleem.

Stelling 2.2: Acceptatietaal is niet beslisbaar, maar wel herkenbaar

$\mathrm{A_{TM}}$ is niet beslisbaar, maar $\mathrm{A_{TM}}$ is herkenbaar.

Bewijs 2.2: Acceptatietaal is niet beslisbaar, maar wel herkenbaar

- We bewijzen door middel van contradictie. Stel er bestaat een beslisser B voor $\mathrm{A_{TM}}$. Dat betekent dat bij input $\langle \mathrm{M}, s\rangle$ B accepteert als M bij input s stopt in q_a en verwerpt als M bij input s niet accepteert, dus stopt in zijn q_r of loopt. We schrijven

  $$\mathrm{B}(\langle \mathrm{M}, s\rangle) \text{ is accept als M } s \text{ accepteert en anders reject.}$$

  Gebruikmakend van B kunnen we nu de contradictie machine C construeren, Deze neemt als input (de codering van) een Turingmachine M, roept B op met input $\langle \mathrm{M}, \mathrm{M}\rangle$, en als B accepteert dan verwerpt C en omgekeerd. Kortom, C heeft de eigenschap:

  $$\forall M \in \text{Turingmachines}: \ \mathrm{C}(\langle \mathrm{M}\rangle) = \text{opposite}(\mathrm{B}(\langle \mathrm{M}, \mathrm{M}\rangle)).$$

  Neem nu voor M hierboven C zelf, dan krijgen we:

  $$\mathrm{C}(\langle \mathrm{C}\rangle) = \text{opposite}(\mathrm{B}(\langle \mathrm{C}, \mathrm{C}\rangle)).$$

  Er geldt nu het volgende:

  $$\begin{aligned} C \text{ accepteert } C &\Leftrightarrow C(\langle C\rangle) = \text{accept} \\ &\Leftrightarrow \text{opposite}(\mathrm{B}(\langle C, C\rangle)) = \text{reject} \\ &\Leftrightarrow C \text{ verwerpt } C. \end{aligned}$$

  Dit is een contradictie, dus B kan niet bestaan en is A_{TM} niet beslisbaar.

- De herkenner H voor $\mathrm{A_{TM}}$ laat gewooon bij input $\langle \mathrm{M}, s\rangle$ de machine M lopen op input s: als M accepteert, dan accepteert H. Als M verwerpt of loopt, dan loopt H ook.

□

Definitie 2.8: Halting-probleem

Stel een Turingmachine M voor met een input w. De **haltingtaal** $\mathrm{H_{TM}}$ is

$$\mathrm{H_{TM}} = \{\langle M, s\rangle \mid M \text{ is een Turingmachine die stopt bij input } s\}$$

zodanig dat de Turingmachine H de taal **beslist**.

Stelling 2.3: Halting-taal is niet beslisbaar, maar wel herkenbaar

$\mathrm{H_{TM}}$ is niet beslisbaar, maar $\mathrm{H_{TM}}$ is herkenbaar.

Bewijs 2.3: Halting-taal is niet beslisbaar, maar wel herkenbaar

- Stel dat $\mathrm{H_{HM}}$ beslisbaar is door een Turingmachine H. We construeren nu beslisser B voor $\mathrm{A_{HM}}$ als volgt: bij input $\langle \mathrm{M}, s\rangle$ doet B:
  - laat eerst H lopen op $\langle \mathrm{M}, s\rangle$
  - $H(\langle \mathrm{M}, s\rangle) = \text{accept} \Rightarrow$ B accepteert en geeft als resultaat wat M geeft
  - $H(\langle \mathrm{M}, s\rangle) = \text{reject} \Rightarrow$ reject B ook de string $\langle \mathrm{M}, s\rangle$.

  Vermits er geen beslisser voor $\mathrm{A_{HM}}$ bestaat, kan H niet bestaan en is dus ook $\mathrm{H_{HM}}$ niet beslisbaar zijn.
- De herkenner H voor $\mathrm{H_{HM}}$ laat gewoon bij input $\langle \mathrm{M}, s\rangle$ de machine M lopen op input s: als M stopt, dan accepteert H. Als M loopt, dan loopt H ook.

□

Stelling 2.4: De complemententalen zijn niet herkanbaar

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

Bewijs 2.4: De complemententalen zijn niet herkanbaar

- Als $\overline{\mathrm{A_{TM}}}$ herkenbaar is,en vermits herkenbaar is, dan is ook $\mathrm{A_{TM}}$ beslisbaar. Maar dat is niet het geval, dus $\overline{\mathrm{A_{TM}}}$ is niet herkenbaar.
- Als $\overline{\mathrm{H_{TM}}}$ herkenbaar is,en vermits herkenbaar is, dan is ook $\mathrm{H_{TM}}$ beslisbaar. Maar dat is niet het geval, dus $\overline{\mathrm{H_{TM}}}$ is niet herkenbaar.

□

2.4 De enumeratormachine

Stelling 2.5: Enumeratormachine

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

Bewijs 2.5: Enumeratormachine

- We beschrijven informeel een herkenner TM voor de taal L bepaald door een gegeven enumerator Enu: TM gebruikt Enu als subruitine als volgt. Geef een string s aan de TM. De TM start de Enu. Telkens de Enu in zijn q_e komt, kijkt TM na of de laatst geproduceerde string op de outputband van de Enu gelijk is aan s. Indien ja: TM accepteert. Indien niet, laat de Enu verderrekenen.
- Laat de TM L bepalen. We construeren de Enu voor die L als volgt. Maak eerst een TM_{gen} die bij input een getal n, het getal n gevolgd door de eerste n strings uit Σ^* op de band zet: $n, s_1, s_2, \ldots, s_n$. Maak een TM die bij input $n, s_1, s_2, \ldots, s_n$ n stappen van TM uitvoert op elk van de n strings: als daarbij een string s_i geaccepteerd wordt, schrijf die dan op de outputband voor Enu. Maak nu een $\text{TM}_{\text{driver}}$ die de opeenvolgende getallen n genereert en dan TM_{gen} en TM oproept. Waarom is dit algoritme correct? Als TM een string s aanvaardt, bv. in m stappen, zal Enu s op de output zetten tijdens iteratie n bij voldoende grote n, namelijk, zodat s valt binnen $s_1, \ldots, s_n$ en zodat $n \geq m$. Merk op dat Enu s oneindig vaak op de output band zal zetten.

□

2.5 Beslisbare talen

Stelling 2.6: Encoderingen van reguliere talen zijn beslisbaar

De talen

- $\text{A}_{\text{DFA}} = \{\langle \text{D}, s\rangle \mid \text{D is een DFA en D accepteert } s\}$
- $\text{A}_{\text{NFA}} = \{\langle \text{N}, s\rangle \mid \text{N is een NFA en D accepteert } s\}$
- $\text{A}_{\text{RegExp}} = \{\langle \text{R}, s\rangle \mid \text{R is een reguliere expressie en R genereert } s\}$

zijn beslisbaar.

Bewijs 2.6: Beslisbare talen

- De beslisser B krijgt als input $\langle \text{D}, s\rangle$. B simuleert D op s. Als D s accepteert, dan stopt B in zijn q_a en accepteert. Als D s verwerpt, dan stopt B in zijn q_r en verwerpt. Er is geen probleem met niet stoppen.
- De beslisser B krijgt als input $\langle \text{N}, s\rangle$. B zet N om in een DFA D en roept de beslisser voor A_{DFA} op met input $\langle \text{D}, s\rangle$.
- De beslisser B krijgt als input $\langle \text{R}, s\rangle$. B zet R om in een NFA N en roept de beslisser voor A_{NFA} op met input $\langle \text{N}, s\rangle$.

□

Stelling 2.7: Encoderingen van reguliere talen zijn beslisbaar - gevolg

De talen

- $\mathcal{E}_{\text{DFA}} = \{\langle \text{DFA} \rangle \mid \epsilon \in \text{L}_{\text{DFA}}\}$
- $\text{E}_{\text{DFA}} = \{\langle \text{DFA} \rangle \mid \text{L}_{\text{DFA}} = \phi\}$
- $\text{EQ}_{\text{DFA}} = \{\langle \text{DFA}_1, \text{DFA}_2 \rangle \mid \text{L}_{\text{DFA}_1} = \text{L}_{\text{DFA}_2}\}$

zijn beslisbaar.

Bewijs 2.7: Encoderingen van reguliere talen zijn beslisbaar - gevolg

- Dit bewijs is triviaal, want A_{DFA} is beslisbaar.
- Stel B een beslisser voor E_{DFA}. B krijgt als input (een codering van) de DFA. B checkt of dat een eindtoestand bereikbaar is vanuit de starttoestand. Indien ja, dan verwerpt B. Indien nee, dan accepteert B.
- Uit DFA_1 en DFA_2 construeer je de DFA_Δ die het symmetrisch verschil tussen L_{DFA_1} en L_{DFA_2} bepaalt. Beslis dan of DFA_Δ de lege taal bepaalt, zie bewijs hierboven.

□

Stelling 2.8: Encodering van een contextvrije taal is beslisbaar

Aanvaarden van een string door een CFG

$$\text{A}_{\text{CFG}} = \{\langle \text{G}, s \rangle \mid \text{G is een CFG en } s \in L_{\text{G}}\}$$

is beslisbaar.

Bewijs 2.8: Encodering van een contextvrije taal is beslisbaar

Stel B een beslisser voor A_{CFG}. Bij input $\langle \text{G}, s \rangle$, converteer G eerst naar zijn Chomsky Normaal Vorm. Geneer alle mogelijke strings met een derivatielengte $2|s| - 1$: dat zijn er eindig veel. B controleert of s daarbij zit. Indien ja, dan accepteert B. Indien nee, dan verwerpt B. □

Stelling 2.9: Encodering van een contextvrije taal is beslisbaar - gevolg

De talen

- $\text{E}_{\text{CFG}} = \{\langle \text{G} \rangle \mid \text{G is een CFG en } L_{\text{G}} = \phi\}$
- $\text{ES}_{\text{CFG}} = \{\langle \text{G} \rangle \mid \text{G is een CFG en } \epsilon \in L_{\text{G}}\}$

zijn beslisbaar.

Bewijs 2.9: Leegheid van een CFG is beslisbaar

- We beschrijven informeel een algoritme dat G transformeert naar een vorm waarin de beslissing gemakkelijk is.
  - als er een regel $A \to \alpha$ in zit, en α bestaat alleen uit eindsymbolen (mag dus ook ϵ zijn), dan
    - verwijder alle regels waar A aan de linkerkant staat
    - vervang in elke regel waar A rechts voorkomt, de voorkomens van A door α
  - blijf dit doen totdat ofwel
    - het startsymbool verwijderd is: reject, want het startsymbool kan een string afleiden
    - er geen regels zijn van de benodigde vorm: accept, want de taal is leeg

  Het algoritme is eindig, omdat elke iteratie minstens één regel verwijdert en er een eindig aantal regels zijn. Het algoritme is correct, want als de taal leeg is, dan zal het algoritme dat ook vinden. Als de taal niet leeg is, dan zal het algoritme dat ook vinden.
- We transformeren de CFG naar zijn Chomsky Normaal Vorm. Indien die de regel $S \to \epsilon$ bevat, dan accepteren we. Indien niet, dan verwerpen we.

□

Stelling 2.10: Beslisbaarheid van CFL

Elke contextvrije taal is beslisbaar.

Bewijs 2.10: Beslisbaarheid van CFL

We zouden voor de CFL L een PDA kunnen kiezen die L beslist en deze PDA omzetten in een Turingmachine. Het probleem is dat een PDA in het algemeen niet-deterministisch is, en dus direct overeenkomt met een niet-deterministische TM. Maar zo'n NTM kan omgezet worden tot een TM die L beslist. Een alternatief is om een CFG G voor L te nemen en aan te tonen dat een beslisser B_G bestaat die voor elke string s kan beslissen of die door G wordt gegenereerd. Dat is natuurlijk niet hetzelfde als A_{CFG} beslissen, maar het terkt er wel op. We kunnen B_G als volgt construeren: we zetten G eerst om in zijn Chomsky Normaal Vorm. We genereren dan alle strings met een derivatielengte $2|s| - 1$: dat zijn er eindig veel. B_G controleert of s daarbij zit. Indien ja, dan accepteert B_G. Indien nee, dan verwerpt B_G. □

2.6 Niet-beslisbare talen

Stelling 2.11: Niet-beslisbare talen

De talen

- $\mathrm{E_{TM}} = \{\langle \mathrm{M} \rangle \mid \mathrm{M}$ is een TM en $\mathrm{L_M} = \phi\}$
- $\mathrm{REGULAR_{TM}}$
- $\mathrm{EQ_{TM}}$

zijn niet-beslisbaar.

Bewijs 2.11: Niet-beslisbare talen

- Het idee van het bewijs is om aan te tonen dat we met een code $\langle \mathrm{M}, s \rangle$, een Turingmachine $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ kunnen berekenen zodat

  $$\langle \mathrm{M}, s \rangle \in \mathrm{A_{TM}} \Leftrightarrow \mathrm{M}_{\langle \mathrm{M},s \rangle} \in \mathrm{E_{TM}}.$$

  $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ doet het volgende met input w: het test of $w \neq s$; zo ja dan volgt een reject; zo niet wordt M op w gesimuleerd. Als M eindigt wordt het resultaat van M teruggeven. Het is duidelijk dat bij input $\langle \mathrm{M}, s \rangle$, $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ berekend kan worden. Verder geldt dat $L_{\mathrm{M}_{\langle \mathrm{M},s \rangle}} = \emptyset$ indien $\langle \mathrm{M}, s \rangle \notin \mathrm{A_{TM}}$ en $L_{\mathrm{M}_{\langle \mathrm{M},s \rangle}} = \{s\}$ indien $\langle \mathrm{M}, s \rangle \in \mathrm{A_{TM}}$. Bijgevolg,

  $$\langle \mathrm{M}, s \rangle \notin \mathrm{A_{TM}} \Leftrightarrow \langle \mathrm{M}_{\langle \mathrm{M},s \rangle} \rangle \in \mathrm{E_{TM}}.$$

  Stel dat $\mathrm{E_{TM}}$ een beslisser E bezit. Construeer nu een beslisser B voor $\mathrm{A_{TM}}$ als volgt: bij input $\langle \mathrm{M}, s \rangle$, berekent B de code $\langle \mathrm{M}_{\langle \mathrm{M},s \rangle} \rangle$ van $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ en voert E uit op deze code. Indien E reject, dan eindigt B met accept; indien E accepteert, dan eindigt B met reject. Sinds een beslisser B voor $\mathrm{A_{TM}}$ niet bestaat, kan $\mathrm{E_{TM}}$ niet beslisbaar zijn.

- We laten zien dat we voor elk paar $\langle \mathrm{M}, s \rangle$ een TM $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ kunnen berekenen met de eigenschap dat $\mathrm{L}_{\mathrm{M}_{\langle \mathrm{M},s \rangle}}$ gelijk is aan de reguliere taal Σ^* indien M s accepteert, en gelijk is aan de niet-reguliere taal $\{0^n 1^n \mid n \in \mathbb{N}\}$ indien M s niet accepteert. Op deze $\langle \mathrm{M}, s \rangle$ junnen we vervolgens een beslisser voor $\mathrm{REGULAR_{TM}}$ toepassen om te beslissen of M s accepteert. Dit leidt tot contradictie. Voor elk paar $\langle \mathrm{M}, s \rangle$ doet $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ het volgende bij input w: het berekent of w van de vorm $0^n 1^n$ is (gebruikmakend van de beslisser). Zo ja, dan stopt $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ met accept; zo niet, dan wordt M gesimuleerd op s; indien M eindigt, dan eindigt $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ in dezelfde eindtoestand als M. Stel dat $\mathrm{REGULAR_{TM}}$ een beslisser R bezit. Construeer nu een beslisser B voor $\mathrm{A_{TM}}$ als volgt: bij input $\langle \mathrm{M}, s \rangle$, berekent B de code $\langle \mathrm{M}_{\langle \mathrm{M},s \rangle} \rangle$ van $\mathrm{M}_{\langle \mathrm{M},s \rangle}$ en voert R uit op deze code. Indien R accepteert, dan is $\mathrm{L}_{\mathrm{M}_{\langle \mathrm{M},s \rangle}} = \Sigma^*$, en dan eindigt B met accept; indien E reject, dan is $\mathrm{L}_{\mathrm{M}_{\langle \mathrm{M},s \rangle}} = \{0^n 1^n \mid n \in \mathbb{N}\}$, en dan eindigt B met reject. Sinds een beslisser B voor $\mathrm{A_{TM}}$ niet bestaat, kan $\mathrm{REGULAR_{TM}}$ niet beslisbaar zijn.

- We weten al dat $\mathrm{E_{TM}}$ niet beslisbaar is, en eigenlijk kan dat probleem beschouwd worden als een speciaal geval van het equivalentieprobleem: namelijk het probleem om te beslissen of een TM equivalent is met M_ϕ. Dus, als $\mathrm{EQ_{TM}}$ beslisbaar is, dan is $\mathrm{E_{TM}}$ ook beslisbaar, wat een contradictie is.

□

Definitie 2.9: Lineair Begrensde Automaat

Een Lineair Begrensde Automaat is een Turingmachine die niet leest of schrijft buiten het deel van de band dat initieel invoer bevat.

Stelling 2.12: Acceptatieprobleem bij LBA

A_{LBA} is beslisbaar.

Bewijs 2.12: Acceptatieprobleem bij LBA

We kijken naar de configuraties die kunnen voorkomen tijdens de uitvoering van een LBA op een string met lengte n. Het aantal toestanden van de LBA noteren we met q en het aantal elementen in het bandalfabet met b. Het aantal mogelijke strings die tijdens de uitvoering op de band kunnen staan is begrensd door b^n. De leeskop kan onder elk van de symbolen staan terwijl de machine in elk van de toestanden kan zitten. Dat geeft in het totaal maximaal qnb^n configuraties. We kunnen nu een beslisser B voor A_{LBA} construeren als volgt: bij input $\langle M, s\rangle$ doet B het volgende

- berekent $Max = qnb^n$,
- simuleert dan M op s voor maximaal Max stappen,
- indien M ondertussen accepteerde, accept; indien M ondertussen verwierp, reject,
- indien M ondertussen niet stopte, reject.

B is een beslisser, omdat M op s stopt na maximaal Max stappen en sinds een beslisser B voor A_{LBA} bestaat, is A_{LBA} beslisbaar. □

Stelling 2.13: Leegheid van een LBA

De taal

$$E_{LBA} = \{\langle M\rangle \mid M \text{ is een LBA en } L_M = \phi\}$$

is niet beslisbaar.

Bewijs 2.13: Leegheid van een LBA

Stel dat E een beslisser voor E_{LBA} is. We construeren nu een beslisser B voor A_{TM} als volgt: bij input $\langle M, s\rangle$ doet B het volgende

- construeert de LBA $A_{M,s}$ die voor een input x kan beslissen of x een accepterende computation history
  $$C \in \Gamma^* Q \Gamma^* : \ \#C_0\# \dots \#C_n\#$$
  is van M voor input s;
- laat E los op $\langle A_{M,s}\rangle$: als E aanvaardt, reject; anders accept;

B beslist A_{TM}, want

$$\begin{aligned} \text{B accepteert } \langle M, s\rangle &\Leftrightarrow \text{E rejects } \langle A_{M,s}\rangle \\ &\Leftrightarrow A_{M,s} \text{ aanvaardt minstens één string} \\ &\Leftrightarrow \text{er bestaat een computation history van M voor } s \end{aligned}$$

Het laatste is equivalent met zeggen dat M de string s accepteert. Dus is B een beslisser van A_{TM}, wat tot een contradictie leidt. Dus E bestaat niet en is E_{LBA} niet beslisbaar. □

2.7 De stelling van Rice

Definitie 2.10: Niet-triviale eigenschap

Een eigenschap P van Turingmachines heet **niet-triviaal** indien

$$P \neq \emptyset \ \wedge \ P^c \neq \emptyset.$$

Definitie 2.11: Taal-invariante eigenschap

De eigenschap P van Turingmachines is **taal-invariant** indien

$$L_{M_1} = L_{M_2} \Rightarrow P(M_1) = P(M_2).$$

of in woorden alle machines die dezelfde taal bepalen hebben ofwel allemaal P, ofwel heeft geen enkele ervan P.

Stelling 2.14: Stelling I van Rice

Voor elke niet-triviale, taal-invariante eigenschap P van Turingmachines is de taal

$$POS_P = \{\langle M\rangle \mid M \in P\}$$

niet beslisbaar.

Bewijs 2.14: Stelling I van Rice

Eerst tonnen we aan dat voor elke input $\langle M, s\rangle$ voor A_{TM} een TM $M_{\langle M,s\rangle}$ kan berekend worden zodat

$$\langle M, s\rangle \in A_{TM} \ \Leftrightarrow \ \langle M_{\langle M,s\rangle}\rangle \in POS_P.$$

Kies twee TM's, $M_\emptyset$ en een TM X zodat

$$P(X) \neq P(M_\emptyset).$$

Aangezien P niet triviaal is, bestaat er zo'n X. We veronderstellen eerst dat $M_\emptyset$ niet voldoet aan P; dan voldoet X wel aan P. (Als $M_\emptyset$ wel voldoet aan P, verwissel dan P en $\overline{P}$. Uit de rest van het bewijs

volgt dan dat $\text{POS}_{\overline{P}}$ onbeslisbaar is, dus ook POS_P.) Voor input $\langle \text{M}, s \rangle$ voor A_{TM} kan de volgende TM $\text{M}_{\langle \text{M},s \rangle}$ berekend worden: bij input w wordt eerst M gesimuleerd op s; als M rejects, dan zal $\text{M}_{\langle \text{M},s \rangle}$ rejecten; als M accepteert, dan wordt X gesimuleerd op wl als X eindigt, dan stopt $\text{M}_{\langle \text{M},s \rangle}$ in dezelfde eindtoestand als X. We stellen vast dat de taal $\text{L}_{\text{M}_{\langle \text{M},s \rangle}}$ geaccepteerd door $\text{M}_{\langle \text{M},s \rangle}$ gelijk is aan $\emptyset$ indien M s niet accepteert en gelijk is aan L_{X} indien M s wel accepteert. Uit de taal-invariantie van P volgt

$$\text{M}_{\langle \text{M},s \rangle} \text{ heeft de eigenschap } P \Leftrightarrow \text{ M accepteert } s.$$

Stel dat POS_P een beslisser E heeft. Construeer nu een TM B die bij input $\langle \text{M}, s \rangle$, de code $\langle \text{M}_{\langle \text{M},s \rangle} \rangle$ berekent en hierop E uitvoert. Dan is B een beslisser voor A_{TM}, wat een contradictie is. Dus E bestaat niet en is POS_P niet beslisbaar. □

2.8 Het Post Correspondence Problem

Stelling 2.15: Toepassing van PCP's

De taal

$$\{< \text{M}, \text{N} > \mid \text{M,N zijn PDA's en } \text{L}_{\text{M}} \cap \text{L}_{\text{n}} \neq \emptyset\}$$

is niet beslisbaar.

Bewijs 2.15: Toepassing van PCP's

We kunnen dit aantonen door een reductie te maken van het beslissen van een Post Correspondence Probleem, naar bovenstaande taal. Een PCP bestaat uit k stenen: $a_1/b_1, \ldots, a_k/b_k$. We kunnen een dergelijke PCP omzetten in 2 PDA's. De eerste is de PDA die de volgende CFG implementeert

$$S \to a_1 A t_1 \mid \ldots \mid a_k A t_k$$
$$A \to \epsilon \mid S,$$

de tweede is de PDA die de volgende CFG implementeert

$$S \to b_1 B t_1 \mid \ldots \mid b_k B t_k$$
$$B \to \epsilon \mid S$$

waarbij $t_1, \ldots, t_k$ nieuwe symbolen zijn die elk uniek een steen identificeren. De strings van de eerste taal bestaan uit strings samengesteld uit de a_i's bovenop de stenen van het spel, gevolgd door de omgekeerde rij van symbolen t_i die aangeeft welke stenen gebruikt zijn. Analoog bestaan de strings van de tweede taal uit strings die gevormd kunnen worden met strings b_i onderaan op de stenen, gevolgd door de omgekeerde rij van symbolen t_i van de gebruikte stenen. Een string in de doorsnede van deze twee talen geeft een sequentie van stenen zodat de string bovenaan en onderaan identiek is. Dus bestaat er een oplossing voor het PCP probleem als de doorsnede van deze twee talen niet leeg is. □

2.9 Veel-één reductie

Definitie 2.12: Turing-berkenbare functie

Een (totale) functie (of afbeelding) f heet Turing-berekenbaar indien er een Turingmachine bestaat die bij input s uiteindelijk stopt met $f(s)$ op de band.

Definitie 2.13: Reductie van talen

We zeggen dat L_1 (over Σ_1) naar L_2 (over Σ_2) ken gereduceerd worden indien er een totale functie $f : \Sigma_1^* \to \Sigma_2^*$ bestaat zodanig dat

$$f(\mathrm{L}_1) \subseteq \mathrm{L}_2 \ \wedge \ f(\overline{\mathrm{L}_1}) \subseteq \overline{\mathrm{L}_2} \ \wedge \ f \text{ is Turing-berekenbaar.}$$

We noteren: $\mathrm{L}_1 \leq_m \mathrm{L}_2$.

Stelling 2.16: Reductie van talen

- $\mathrm{L}_1 \leq_m \mathrm{L}_2 \ \wedge \ \mathrm{L}_2$ beslisbaar $\Rightarrow \ \mathrm{L}_1$ beslisbaar.
- $\mathrm{L}_1 \leq_m \mathrm{L}_2 \ \wedge \ \mathrm{L}_2$ herkenbaar $\Rightarrow \ \mathrm{L}_1$ herkenbaar.
- $\mathrm{L}_1 \leq_m \mathrm{L}_2 \ \wedge \ \mathrm{L}_1$ niet-herkenbaar $\Rightarrow \ \mathrm{L}_2$ niet-herkenbaar.
- $\mathrm{L}_1 \leq_m \mathrm{L}_2 \ \wedge \ \mathrm{L}_1$ niet-beslisbaar $\Rightarrow \ \mathrm{L}_2$ niet-beslisbaar.
- $\mathrm{L}_1 \leq_m \mathrm{L}_2 \ \Rightarrow \ \overline{\mathrm{L}_1} \leq_m \overline{\mathrm{L}_2}$.

Stelling 2.17: Equivalentietaal bij Turingmachines

De taal

$$\mathrm{EQ}_{\mathrm{TM}} = \{\langle \mathrm{M}_1, \mathrm{M}_2 \rangle \mid \mathrm{M}_1 \text{ en } \mathrm{M}_2 \text{ zijn Turingmachines en } \mathrm{L}_{\mathrm{M}_1} = \mathrm{L}_{\mathrm{M}_2}\}$$

is niet beslisbaar, niet herkenbaar en ook niet co-herkenbaar.

Bewijs 2.16: Equivalentietaal bij Turingmachines

- We weten al dat E_{TM} niet beslisbaar is, en we reduceren nu E_{TM} naar $\mathrm{EQ}_{\mathrm{TM}}$ met de functie f die bij input $\langle \mathrm{M} \rangle$ als resultaat geeft: $\langle \mathrm{M}, \mathrm{M}_\phi \rangle$ waarin M_ϕ een TM is die de lege taal bepaalt. Het is duidelijk dat f Turing-berekenbaar is. Bijgevolg, $\mathrm{E}_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$. $\mathrm{EQ}_{\mathrm{TM}}$ is niet beslisbaar, want E_{TM} is niet beslisbaar.
- We zullen twee mapping reducties construeren, de eerste zal aantonen dat $\mathrm{A}_{\mathrm{TM}} \leq_m \overline{\mathrm{EQ}_{\mathrm{TM}}}$ en de andere toont aan dat $\mathrm{A}_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$. Vermits $\overline{\mathrm{A}_{\mathrm{TM}}}$ niet herkenbaar is, bewijst dat de stelling.
  1. f transformeert $\langle \mathrm{M}, s \rangle$ in $\langle \mathrm{M}_s, \mathrm{M}_\phi \rangle$; daarin is M_s een machine die elke string accepteert indien M s accepteert; het is duidelijk dat f berekenbaar is; we moeten nog de andere voorwaarden aantonen:

- indien M s accepteert, dan zijn M_s en M_ϕ verschillend
- indien M s niet accepteert, dan zijn M_s en M_ϕ gelijk

2. f transformeert $\langle M, s\rangle$ in $\langle M_s, M_{\Sigma^*}\rangle$; M_{Σ^*} is de machine die alles accepteert; M_s is zoals hiervoor; de voorwaarden op f zijn gemakkelijk na te gaan.

□

2.10 Orakelmachine

Stelling 2.18: Orakelmachine

Er bestaat een orakelmachien $O^{A_{TM}}$ die E_{TM} beslist.

Bewijs 2.17: Orakelmachine

We construeren $O^{A_{TM}}$ als volgt: bij input $\langle M\rangle$ doet $O^{A_{TM}}$

- construeer een Turingmachine P die bij input w doet
  - laat M lopen op alle strigs van Σ^*
  - als M een string accepteert, accept
- vraag aan het orakel voor A_{TM} of $\langle P, \epsilon\rangle \in A_{TM}$
- als het orakel ja antwoordt, reject; als het orakel nee antwoordt, accept.

Als $L_M \neq \emptyset$, dan accepteert P elke input inclusief ϵ; dan antwoordt het orakel ja en dan reject $O^{A_{TM}}$. Als $L_M = \emptyset$, dan accepteert P geen enkele input en evenmin ϵ; dan antwoordt het orakel nee en dan accepteert $O^{A_{TM}}$. We besluiten dat $O^{A_{TM}}$ de taal E_{TM} beslist. □

Definitie 2.14: Turingreduceerbaar

Een taal A is Turingreduceerbaar naar een taal B, indien A beslisbaar is relatief t.o.v. B, t.t.z. er bestaat een orakelmachine O^B die A beslist. We schrijven

$$A \leq_T B.$$

Stelling 2.19: Turingreduceerbaarheid

- Indien $A \leq_T B$ en B is beslisbaar, dan is A beslisbaar.
- Indien $A \leq_m B$, dan is $A \leq_m B$.
- transistiviteit: indien $A \leq_T B \leq_T C$, dan $A \leq_T C$.

2.11 Turing-berekenbare functies en recursieve functies

Definitie 2.15: Turing-berekenbare partiële functies

Een partiële funtie f heet Turing-berekenbaar indien

$$\exists \mathrm{M} \in \mathrm{TM},\ \forall s \in \Sigma^* :\ \mathrm{M} \text{ stopt op input } s \ \Leftrightarrow\ s \in \mathrm{dom}(f).$$

of in woorden: er een Turingmachine M bestaat die bij input s stopt als en slechts als s behoort tot domein van f; bovendien, als M stopt, dan staat $f(s)$ op de band. We zeggen dat M de partiële functie f berekent.

Definitie 2.16: Compositie functie constructor

De compositie functie constructor Cn is de functie

$$h : \mathbb{N}^k \to \mathbb{N} :\ h(\overline{x}) = f(g_1(\overline{x}), \ldots, g_k(\overline{x}))$$

waarbij $f : \mathbb{N}^m \to \mathbb{N},\ g_1, \ldots, g_m : \mathbb{N}^k \to \mathbb{N},\ \overline{x} = (x_1, \ldots, x_k)$

Definitie 2.17: Primitief-recursieve functie constructor

De primitief-recursieve functie constructor Pr is de functie

$$h : \mathbb{N}^{k+1} \to \mathbb{N} : \begin{cases} h(\overline{x}, 0) = f(\overline{x}) \\ h(\overline{x}, y+1) = g(\overline{x}, y, h(\overline{x}, y)) \end{cases}$$

waarbij $f : \mathbb{N}^k \to \mathbb{N},\ g : \mathbb{N}^{k+2} \to \mathbb{N}$

Definitie 2.18: Primitief-recursieve functie

Een primitief-recursieve functie wordt inductief gedefinieerd als volgt:

- Elke basis functie is primitief-recursief.
- Indien $g_1, \ldots, g_m : \mathbb{N}^k \to \mathbb{N}$ en $f : \mathbb{N}^m \to \mathbb{N}$ primitief-recursieve functies zijn, dan is

  $$Cn[f, g_1, \ldots, g_m]$$

  ook primitief-recursief.
- Indien $g : \mathbb{N}^{k+2} \to \mathbb{N}$ en $f : \mathbb{N}^k \to \mathbb{N}$ primitief-recursieve functies zijn, dan is

  $$Pr[f, g]$$

  ook primitief-recursief.

Definitie 2.19: Onbegrensde minimizatie

De functie $Mn[f]$ is de functie

$$g(\overline{x}) = y$$

waarbij $g : \mathbb{N}^k \to \mathbb{N}$ op voorwaarde dat

$$f(\overline{x}, y) = 0 \ \wedge \ \forall z < y : \ f(\overline{x}, z) \text{ is gedefinieerd } \wedge f(\overline{x}, y) \neq 0.$$

Als deze voorwaarden niet voldaan zijn, dan is $g(\overline{x})$ niet gedefinieerd en dus $Mn[f]$ ook niet.

Definitie 2.20: Recursieve functies

Een recursieve functie wordt inductief gedefinieerd als volgt:

- Elke basis functie is recursief.
- Indien $g_1, \ldots, g_m : \mathbb{N}^k \to \mathbb{N}$ en $f : \mathbb{N}^m \to \mathbb{N}$ recursieve functies zijn, dan is

  $$CN[f, g_1, \ldots, g_m]$$

  waarbij CN gelijk is aan Cn veralgemeend naar partiële functies, ook recursief.
- Indien $g : \mathbb{N}^{k+2} \to \mathbb{N}$ en $f : \mathbb{N}^k \to \mathbb{N}$ recursieve functies zijn, dan is

  $$PR[f, g]$$

  waarbij PR gelijk is aan Pr veralgemeend naar partiële functies, ook recursief.
- Indien $f : \mathbb{N}^{k+1} \to \mathbb{N}$ recursief is, dan is

  $$Mn[f]$$

  ook recursief.

Recursieve functies worden ook μ-recursief genoemd.

3 Herschrijfsystemen

4 Andere rekenparadigmas

5 Talen en complexiteit