Overzicht van Bewijzen: Automaten en Berekenbaarheid

Tibo Stans

Inhoudsopgave

1 Hoofdstuk 2: Reguliere talen en eindige automaten

1.1 Bewijs: DFA-overgangen komen overeen met wandelingen in de transition graph

Bewijs. Laat

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

een DFA zijn en laat G_M de bijhorende transition graph zijn. We tonen dat voor alle toestanden $q_i, q_j \in Q$ en elk woord $w \in \Sigma^+$ geldt:

$$\delta^*(q_i, w) = q_j$$

als en slechts als er in G_M een wandeling met label w bestaat van q_i naar q_j.

We bewijzen dit met inductie op de lengte van w.

Basisstap. Als $|w| = 1$, dan is $w = a$ voor een symbool $a \in \Sigma$. Dan geldt

$$\delta^*(q_i, a) = q_j$$

precies wanneer $\delta(q_i, a) = q_j$. Volgens de constructie van de transition graph bestaat er dan precies een boog van q_i naar q_j met label a. Dus de uitspraak klopt voor woorden van lengte 1.

Inductiestap. Stel dat de uitspraak klopt voor alle woorden van lengte n. Neem een woord w van lengte $n + 1$ en schrijf

$$w = va$$

met $|v| = n$ en $a \in \Sigma$. Stel eerst dat

$$\delta^*(q_i, w) = q_j.$$

Noem

$$\delta^*(q_i, v) = q_k.$$

Dan geldt ook $\delta(q_k, a) = q_j$. Door de inductiehypothese bestaat er een wandeling met label v van q_i naar q_k. Omdat $\delta(q_k, a) = q_j$, bestaat er een boog met label a van q_k naar q_j. Samen vormen die een wandeling met label $va = w$ van q_i naar q_j.

Omgekeerd, stel dat er een wandeling met label $w = va$ bestaat van q_i naar q_j. Dan bestaat er een toestand q_k zodat het eerste deel van de wandeling label v heeft van q_i naar q_k, en de laatste boog label a heeft van q_k naar q_j. Door de inductiehypothese geldt $\delta^*(q_i, v) = q_k$. Omdat de laatste boog label a heeft, geldt $\delta(q_k, a) = q_j$. Dus

$$\delta^*(q_i, va) = \delta(\delta^*(q_i, v), a) = \delta(q_k, a) = q_j.$$

Daarmee zijn beide richtingen bewezen. □

1.2 Bewijs: NFA naar DFA via subsetconstructie

Bewijs. Het idee is om elke toestand van de nieuwe DFA te laten overeenkomen met een verzameling mogelijke toestanden van de NFA. Neem een NFA

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

We construeren een DFA

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

als volgt:

- $Q' = 2^Q$, dus elke DFA-toestand is een verzameling NFA-toestanden.
- $q_0' = \delta^*(q_0, \lambda)$ als er λ-transities zijn; zonder λ-transities is dit gewoon $\{q_0\}$.
- Voor $S \subseteq Q$ en $a \in \Sigma$ definiëren we

$$\delta'(S, a) = \bigcup_{s \in S} \delta^*(s, a).$$

- Een DFA-toestand is accepterend zodra minstens één van de mogelijke NFA-toestanden accepterend is:

$$F' = \{S \in Q' \mid S \cap F \neq \emptyset\}.$$

De sleutelstelling is dat de DFA na het lezen van een woord v precies de verzameling toestanden bevat waarin de NFA kan zitten na het lezen van v. Dit bewijs je met inductie op $|v|$. Voor $v = \lambda$ volgt dit uit de definitie van q'_0. Voor $v = wa$ volgt het uit de inductiehypothese voor w en uit de definitie van δ'. Daarom geldt:

$$v \in L(M) \iff (\delta')^*(q'_0, v) \in F'.$$

Dus $L(M) = L(M')$. □

1.3 Bewijs: terminatie en correctheid van het DFA-minimalisatiealgoritme

Bewijs. Het markeer-algoritme markeert paren toestanden die onderscheidbaar zijn.

Terminatie. Er zijn maar eindig veel paren toestanden. In elke iteratie wordt een nieuw paar gemarkeerd of stopt het algoritme. Omdat er maximaal $|Q|^2$ paren zijn, moet het algoritme stoppen.

Correctheid. We tonen dat elk onderscheidbaar paar uiteindelijk gemarkeerd wordt. Een paar (p, q) is onderscheidbaar als er een woord v bestaat zodat

$$\delta^*(p, v) \in F \quad \text{en} \quad \delta^*(q, v) \notin F,$$

of omgekeerd. We bewijzen met inductie op $|v|$ dat zo'n paar na hoogstens $|v|$ iteraties gemarkeerd wordt.

Voor $|v| = 0$ is $v = \lambda$. Dan is één van p, q accepterend en de andere niet. Zo'n paar wordt initieel gemarkeerd.

Voor $|v| > 0$ schrijf $v = aw$. Zet $p' = \delta(p, a)$ en $q' = \delta(q, a)$. Dan onderscheidt w de toestanden p' en q'. Volgens de inductiehypothese wordt (p', q') gemarkeerd. Daarom markeert het algoritme ook (p, q), want er bestaat een symbool a dat naar een reeds gemarkeerd paar leidt. Dus alle onderscheidbare paren worden gemarkeerd. Omgekeerd markeert het algoritme alleen paren die aantoonbaar onderscheidbaar zijn. Daarmee is het algoritme correct. □

1.4 Bewijs: de overgangsfunctie van de geminimaliseerde DFA is welgedefinieerd

Bewijs. In de geminimaliseerde DFA worden ononderscheidbare toestanden samengenomen in equivalentieklassen. We moeten tonen dat de overgang vanuit zo'n klasse niet afhangt van de gekozen representant.

Neem twee ononderscheidbare toestanden p en q. Dan geldt voor elk woord v:

$$\delta^*(p, v) \in F \iff \delta^*(q, v) \in F.$$

We moeten tonen dat voor elk symbool $a \in \Sigma$ de toestanden $\delta(p, a)$ en $\delta(q, a)$ ook ononderscheidbaar zijn. Neem een willekeurig woord w. Dan geldt:

$$\delta^*(\delta(p, a), w) = \delta^*(p, aw)$$

en

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

Omdat p en q ononderscheidbaar zijn, hebben aw vanuit p en q hetzelfde acceptatiegedrag. Dus hebben w vanuit $\delta(p, a)$ en $\delta(q, a)$ ook hetzelfde acceptatiegedrag. Daarom zijn $\delta(p, a)$ en $\delta(q, a)$ ononderscheidbaar. De overgangsfunctie op equivalentieklassen is dus welgedefinieerd. □

1.5 Bewijs: de geminimaliseerde DFA is minimaal

Bewijs. Laat $\hat{M}$ de DFA zijn die ontstaat na het samenvoegen van alle ononderscheidbare toestanden. Alle toestanden van $\hat{M}$ zijn bereikbaar, want onbereikbare toestanden worden eerst verwijderd. Voor elke toestand $\hat{q}_i$ bestaat dus een woord w_i zodat

$$\hat{\delta}^*(\hat{q}_0, w_i) = \hat{q}_i.$$

Stel dat er een equivalente DFA M' bestaat met minder toestanden dan $\hat{M}$. Omdat M' minder toestanden heeft, moeten er volgens het duivenhokprincipe twee woorden w_k en w_l zijn die in M' naar dezelfde toestand leiden. In $\hat{M}$ leiden ze naar twee verschillende toestanden $\hat{q}_k$ en $\hat{q}_l$. Omdat verschillende toestanden in $\hat{M}$ onderscheidbaar zijn, bestaat er een woord v dat ze onderscheidt. Dus $w_k v$ en $w_l v$ hebben verschillend acceptatiegedrag in $\hat{M}$. Maar in M' vertrekken beide woorden na w_k en w_l vanuit dezelfde toestand, dus kan M' ze niet verschillend behandelen. Daarom kan M' niet equivalent zijn met $\hat{M}$. Dus $\hat{M}$ is minimaal. □

1.6 Bewijs: pumping lemma voor reguliere talen

Bewijs. Laat L regulier zijn en laat $M = (Q, \Sigma, \delta, q_0, F)$ een DFA zijn met $L(M) = L$. Neem $m = |Q|$. Voor elk woord $w \in L$ met $|w| \geq m$ bekijkt de DFA tijdens de eerste m gelezen symbolen $m + 1$ toestanden. Omdat er maar m toestanden zijn, komt minstens één toestand twee keer voor. Schrijf daarom

$$w = xyz,$$

waarbij $y \neq \lambda$ het deel is dat de lus doorloopt en $|xy| \leq m$. Omdat de lus nul, één of meerdere keren kan worden doorlopen, geldt voor alle $i \geq 0$:

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

Dit is precies het pumping lemma voor reguliere talen. □

2 Hoofdstuk 3: Reguliere expressies en reguliere grammatica's

2.1 Bewijs: reguliere expressies beschrijven precies de reguliere talen

Bewijs. We tonen beide richtingen.

Van reguliere expressie naar NFA. We bewijzen met structurele inductie op de reguliere expressie r dat er een NFA $M(r)$ bestaat met $L(M(r)) = L(r)$.

Voor de basisgevallen maken we eenvoudige automaten voor $\emptyset$, λ en elk symbool $a \in \Sigma$. Voor de inductiestappen gebruiken we de automaten uit de inductiehypothese:

- Voor $r + s$ maken we een nieuwe starttoestand met λ-transities naar de automaten voor r en s.
- Voor $r \cdot s$ verbinden we de eindtoestand van de automaat voor r met de starttoestand van de automaat voor s via een λ-transitie.
- Voor r^* voegen we een nieuwe start- en eindtoestand toe, laten we r overslaan met een λ-transitie, en voegen we een teruglus toe om r opnieuw te kunnen lezen.

Daarmee bestaat voor elke reguliere expressie een equivalente NFA.

Van NFA naar reguliere expressie. Neem een NFA. We maken er een generalized transition graph van, waarin pijlen gelabeld mogen zijn met reguliere expressies. Daarna elimineren we toestanden één voor één. Wanneer een toestand wordt verwijderd, vervangen we alle paden die via die toestand liepen door equivalente reguliere expressies op rechtstreekse pijlen. Uiteindelijk blijven alleen een start- en eindtoestand over. De expressie op de pijl tussen die twee toestanden beschrijft precies de taal van de oorspronkelijke automaat. Dus reguliere expressies en eindige automaten beschrijven precies dezelfde talen. □

2.2 Bewijs: eliminatie van een toestand in een GTG behoudt de taal

Bewijs. Neem een generalized transition graph G met toestanden $q_0, \ldots, q_n$. De boog van q_i naar q_j heeft label r_{ij}, waarbij r_{ij} een reguliere expressie is. We verwijderen een toestand q_k die niet de start- of eindtoestand is. Voor elke twee overblijvende toestanden q_i en q_j vervangen we het label r_{ij} door

$$r'_{ij} = r_{ij} + r_{ik} r_{kk}^* r_{kj}.$$

Noem de nieuwe GTG zonder q_k de graaf G'. We tonen dat G en G' dezelfde taal accepteren.

Neem eerst een woord w dat door G geaccepteerd wordt. Dan bestaat er een accepterende wandeling in G waarvan de labels samen w vormen. Elke keer dat deze wandeling via q_k gaat, ziet dat stuk er uit als:

$$q_i \to q_k \to \cdots \to q_k \to q_j.$$

Het label van zo'n stuk voldoet aan

$$r_{ik} r_{kk}^* r_{kj}.$$

In G' is dit precies opgenomen in het nieuwe rechtstreekse label

$$r'_{ij} = r_{ij} + r_{ik} r_{kk}^* r_{kj}.$$

Dus we kunnen alle bezoeken aan q_k uit de wandeling verwijderen en vervangen door rechtstreekse bogen in G'. Daarmee accepteert G' hetzelfde woord w.

Omgekeerd, neem een woord w dat door G' geaccepteerd wordt. Elke gebruikte boog $q_i \to q_j$ in G' heeft label

$$r'_{ij} = r_{ij} + r_{ik} r_{kk}^* r_{kj}.$$

Als het deel van w voldoet aan r_{ij}, dan nemen we in G gewoon de oude boog $q_i \to q_j$. Als het deel voldoet aan $r_{ik} r_{kk}^* r_{kj}$, dan nemen we in G de weg via q_k, met nul of meer lussen op q_k. Zo kunnen we elke wandeling in G' nabootsen door een wandeling in G. Dus elk woord dat G' accepteert, wordt ook door G geaccepteerd.

Daarom accepteren G en G' dezelfde taal. Dit rechtvaardigt de state-elimination methode waarmee een NFA naar een reguliere expressie wordt omgezet. □

2.3 Bewijs: rechts-lineaire grammatica's beschrijven precies de reguliere talen

Bewijs. We tonen beide richtingen.

Van rechts-lineaire grammatica naar NFA. Neem een rechts-lineaire grammatica G. Maak een NFA waarvan de toestanden overeenkomen met de variabelen van G, plus eventueel een extra eindtoestand. Het startsymbool wordt de starttoestand. Elke productie $A \to aB$ wordt een overgang van A naar B met label a. Elke productie $A \to a$ wordt een overgang van A naar de extra eindtoestand. Elke productie $A \to \lambda$ maakt A accepterend. Deze NFA simuleert exact de afleidingen van de grammatica. Dus $L(G)$ is regulier.

Van DFA naar rechts-lineaire grammatica. Neem een DFA $M = (Q, \Sigma, \delta, q_0, F)$. Maak voor elke toestand $q \in Q$ een variabele. Het startsymbool is q_0. Voor elke overgang $\delta(q_i, a) = q_j$ voegen we de productie

$$q_i \to a q_j$$

toe. Voor elke accepterende toestand $q_f \in F$ voegen we

$$q_f \to \lambda$$

toe. Een pad in de DFA correspondeert nu exact met een afleiding in de grammatica. Dus elke reguliere taal heeft een rechts-lineaire grammatica. □

2.4 Bewijs: links-lineaire grammatica's zijn equivalent aan rechts-lineaire grammatica's

Bewijs. Een rechts-lineaire grammatica kan worden omgezet naar een links-lineaire grammatica door de rechterkanten van de producties om te draaien. Hierdoor wordt de gegenereerde taal gespiegeld. Als G een rechts-lineaire grammatica is, dan genereert de omgekeerde grammatica $L(G)^R$. Reguliere talen zijn gesloten onder spiegeling. Daarom is de taal van een links-lineaire grammatica regulier. Omgekeerd geldt hetzelfde argument in de andere richting. Dus links-lineaire en rechts-lineaire grammatica's beschrijven dezelfde klasse talen: de reguliere talen. □

3 Hoofdstuk 4: Sluitingseigenschappen van reguliere talen

3.1 Bewijs: reguliere talen zijn gesloten onder doorsnede

Bewijs. Laat L_1 en L_2 reguliere talen zijn. Dan bestaan er DFA's

$$M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$$

en

$$M_2 = (P, \Sigma, \delta_2, p_0, F_2)$$

met $L(M_1) = L_1$ en $L(M_2) = L_2$.

We construeren een productautomaat

$$\hat{M} = (Q \times P, \Sigma, \hat{\delta}, (q_0, p_0), \hat{F})$$

waarbij

$$\hat{\delta}((q,p),a) = (\delta_1(q,a), \delta_2(p,a))$$

en

$$\hat{F} = \{(q,p) \in Q \times P \mid q \in F_1 \text{ en } p \in F_2\}.$$

De automaat $\hat{M}$ simuleert dus beide DFA's tegelijk op hetzelfde inputwoord.

We tonen met inductie op $|w|$ dat

$$\hat{\delta}^*((q_0,p_0),w) = \big(\delta_1^*(q_0,w), \delta_2^*(p_0,w)\big).$$

Voor $w = \lambda$ is dit duidelijk. Voor $w = va$ volgt het uit de inductiehypothese voor v en uit de definitie van $\hat{\delta}$.

Daarmee geldt:

$$\begin{aligned} w \in L(\hat{M}) &\iff \hat{\delta}^*((q_0,p_0),w) \in \hat{F} \\ &\iff \delta_1^*(q_0,w) \in F_1 \text{ en } \delta_2^*(p_0,w) \in F_2 \\ &\iff w \in L_1 \text{ en } w \in L_2 \\ &\iff w \in L_1 \cap L_2. \end{aligned}$$

Dus $L(\hat{M}) = L_1 \cap L_2$. Daarom zijn reguliere talen gesloten onder doorsnede. □

4 Hoofdstuk 5: Context-vrije grammatica's

4.1 Bewijs: correctheid van een grammatica voor $\{a^n b^n \mid n \in \mathbb{N}\}$

Bewijs. Neem de grammatica

$$S \to aSb \mid \lambda.$$

We tonen dat $L(G) = \{a^n b^n \mid n \in \mathbb{N}\}$.

Inclusie $\supseteq$. Voor elk n kunnen we n keer de regel $S \to aSb$ toepassen en daarna $S \to \lambda$. Dan krijgen we:

$$S \Rightarrow aSb \Rightarrow a^2Sb^2 \Rightarrow \cdots \Rightarrow a^nSb^n \Rightarrow a^nb^n.$$

Dus elk woord a^nb^n zit in $L(G)$.

Inclusie $\subseteq$. We bewijzen met inductie op het aantal afleidingsstappen dat elke sententiële vorm de vorm a^iSb^i of a^ib^i heeft. In het begin is dit $S = a^0Sb^0$. Een toepassing van $S \to aSb$ verhoogt het aantal a's links en b's rechts allebei met één. Een toepassing van $S \to \lambda$ verwijdert de enige overblijvende variabele. Daarom kan elk terminaal woord alleen de vorm a^ib^i hebben. Dus $L(G) \subseteq \{a^nb^n \mid n \in \mathbb{N}\}$. □

4.2 Bewijs: parse trees komen overeen met derivaties

Bewijs. Elke derivatie bepaalt een parse tree, en elke parse tree kan worden gelineariseerd tot een links- of rechtsafleiding.

Van derivatie naar parse tree: gebruik inductie op het aantal afleidingsstappen. Elke toepassing van een productie $A \to \alpha$ vervangt een blad A door kinderen met labels uit α. Na alle stappen is de yield van de boom gelijk aan het afgeleide woord.

Van parse tree naar derivatie: gebruik inductie op de hoogte van de boom. Kies telkens een variabel blad en vervang het door de kinderen die in de boom onder dat blad staan. Omdat de grammatica context-vrij is, maakt de volgorde waarin onafhankelijke variabelen worden uitgebreid niet uit. Links- en rechtsafleidingen zijn dus verschillende linearisaties van dezelfde parse tree. □

4.3 Bewijs: algemene substitutie-eigenschap voor CFG's

Bewijs. Stel dat G een productie $A \to x_1Bx_2$ bevat en dat de producties voor B precies

$$B \to y_1 \mid \cdots \mid y_n$$

zijn. Vervang $A \to x_1Bx_2$ door

$$A \to x_1y_1x_2 \mid \cdots \mid x_1y_nx_2.$$

Noem de nieuwe grammatica $\hat{G}$.

We tonen dat $L(G) = L(\hat{G})$. Neem eerst een afleiding in G waarin $A \to x_1 B x_2$ wordt gebruikt. Omdat het eindresultaat terminaal is, moet B later worden vervangen door één van de y_i. In $\hat{G}$ kunnen we die twee stappen vervangen door één rechtstreekse stap $A \to x_1 y_i x_2$. Dus elk woord uit $L(G)$ zit ook in $L(\hat{G})$.

Omgekeerd komt elke nieuwe regel $A \to x_1 y_i x_2$ in $\hat{G}$ overeen met twee oude stappen in G:

$$A \Rightarrow x_1 B x_2 \Rightarrow x_1 y_i x_2.$$

Dus elke afleiding in $\hat{G}$ kan worden nagebootst in G. Daarom zijn de talen gelijk. □

5 Hoofdstuk 6: Vereenvoudigingen en normaalvormen van CFG's

5.1 Bewijs: eliminatie van nutteloze variabelen

Bewijs. Een variabele is nuttig als ze voorkomt in een afleiding van het startsymbool naar een terminaal woord. Het algoritme werkt in twee stappen.

Stap 1: variabelen vinden die terminals kunnen genereren. Start met $U = \emptyset$. Voeg herhaaldelijk elke variabele A toe waarvoor er een productie $A \to \vec{x}$ bestaat met

$$\vec{x} \in (T \cup U)^*.$$

Met inductie toon je dat A in iteratie k wordt toegevoegd als en slechts als A in minimaal k stappen een terminaal woord kan genereren.

Stap 2: bereikbare variabelen vinden. Maak een dependency graph waarin er een pijl $X \to Y$ is als Y voorkomt in de rechterkant van een productie van X. Behoud alleen de variabelen die vanuit S bereikbaar zijn.

Na stap 1 kunnen de overblijvende variabelen terminals genereren. Na stap 2 zijn ze ook bereikbaar vanuit S. Dus elke overblijvende variabele is nuttig. Omgekeerd moet elke nuttige variabele zowel terminals kunnen genereren als bereikbaar zijn vanuit S. Daarom verwijdert het algoritme precies de nutteloze variabelen. □

5.2 Bewijs: eliminatie van λ-producties

Bewijs. Bepaal eerst de nullable variabelen N. Start met alle variabelen A waarvoor $A \to \lambda$ een productie is. Voeg daarna iteratief elke variabele A toe waarvoor er een productie

$$A \to B_1 B_2 \cdots B_k$$

bestaat met alle $B_i \in N$.

Daarna vervangen we elke productie door alle varianten waarin nullable variabelen optioneel worden weggelaten. De lege rechterkant wordt alleen behouden als λ expliciet in de taal moet blijven, meestal via een nieuw startsymbool.

Correctheid: als in een oude afleiding een nullable variabele uiteindelijk naar λ afleidt, kan de nieuwe grammatica die variabele meteen weglaten. Omgekeerd komt elke nieuwe productie waarin nullable variabelen zijn weggelaten overeen met een oude afleiding waarin die variabelen eerst aanwezig waren en daarna naar λ afleidden. Dus de taal blijft gelijk, behalve dat de lege string apart moet worden behandeld. □

5.3 Bewijs: eliminatie van unit-producties

Bewijs. Een unit-productie heeft de vorm $A \to B$ met $A, B \in V$. Voor elke variabele A berekenen we alle variabelen B waarvoor

$$A \Rightarrow^* B$$

via alleen unit-producties. Voor elke niet-unit-productie $B \to \alpha$ voegen we vervolgens $A \to \alpha$ toe. Daarna verwijderen we alle unit-producties.

Elke oude afleiding waarin A via een keten van unit-producties naar B gaat en daarna $B \to \alpha$ toepast, wordt in de nieuwe grammatica vervangen door de directe stap $A \to \alpha$. Omgekeerd is elke nieuwe directe stap precies een afkorting voor zo'n oude keten. Daarom verandert de taal niet. □

5.4 Bewijs: elke CFG zonder λ heeft een equivalente Chomsky Normal Form

Bewijs. Neem een CFG G waarvoor $\lambda \notin L(G)$. Eerst verwijderen we nutteloze producties, λ-producties en unit-producties. Daarna vervangen we terminals die in lange rechterkanten voorkomen door nieuwe variabelen T_a met productie $T_a \to a$. Ten slotte splitsen we elke productie met meer dan twee symbolen op in binaire producties met nieuwe hulpvariabelen.

Elke stap behoudt de taal: we vervangen alleen lokale stukken van afleidingen door equivalente lokale stukken. Het resultaat is een grammatica waarin elke productie de vorm

$$A \to BC$$

of

$$A \to a$$

heeft. Dat is Chomsky Normal Form. □

5.5 Bewijs: elke CFG zonder λ heeft een equivalente Greibach Normal Form

Bewijs. Vertrek van een CFG zonder λ. Vereenvoudig de grammatica eerst door nutteloze, λ- en unit-producties te verwijderen. Orden daarna de variabelen. Wanneer een productie begint met een variabele, vervangen we die variabele door haar alternatieven via de substitutie-eigenschap. Zo schuiven we variabelen aan het begin systematisch weg. Eventuele links-recursie wordt verwijderd door extra hulpvariabelen toe te voegen.

Elke transformatie behoudt de taal. Uiteindelijk begint elke rechterkant met een terminal, eventueel gevolgd door variabelen. Elke productie heeft dus de vorm

$$A \to aV_1 \cdots V_k.$$

Dat is Greibach Normal Form. □

6 Hoofdstuk 7: Pushdown automata

6.1 Bewijs: voor elke CFG bestaat een NPDA met dezelfde taal

Bewijs. Neem een CFG G zonder λ en zet die in Greibach Normal Form. Elke productie heeft dan de vorm

$$A \to aV_1 \cdots V_n.$$

We bouwen een NPDA die de nog te verwerken variabelen op de stack bewaart. De stack start met het startsymbool S. Wanneer bovenaan de stack A staat en het volgende inputsymbool a is, mag de NPDA een productie $A \to aV_1 \cdots V_n$ kiezen, a lezen, A poppen en $V_1 \cdots V_n$ pushen.

We bewijzen met inductie op het aantal gelezen symbolen dat de stackinhoud precies overeenkomt met de variabelen die nog in de sententiële vorm staan. In het basisgeval is nog niets gelezen en staat S op de stack. In de inductiestap komt één NPDA-stap overeen met één Greibach-productie, die precies één terminal leest. Als de input volledig gelezen is en de stack leeg is, bestaat er dus een afleiding in G. Omgekeerd kan elke afleiding in Greibach Normal Form door de NPDA worden gevolgd. Dus $L(M) = L(G)$. □

6.2 Bewijs: voor elke NPDA bestaat een CFG met dezelfde taal

Bewijs. Zet de NPDA eerst in een standaardvorm waarin elke transitie ofwel één stacksymbool pusht ofwel één stacksymbool popt. Maak variabelen van de vorm $[pAq]$. De betekenis is: $[pAq]$ genereert alle woorden waarmee de NPDA vanuit toestand p met A bovenaan de stack naar toestand q kan gaan terwijl A netto wordt verwijderd.

Voor elke combinatie van een push-transitie en een bijpassende pop-transitie voegen we producties toe die alle mogelijke tussenliggende toestanden raden. De startvariabele genereert alle woorden waarmee de initiële stack kan worden leeggemaakt en een accepterende toestand kan worden bereikt.

Correctheid volgt uit inductie op het aantal stappen van de NPDA-berekening. Elke accepterende berekening bepaalt een derivatie in de grammatica. Omgekeerd kiest elke derivatie producties die overeenkomen met echte NPDA-transities. Daarom hebben de NPDA en de CFG dezelfde taal. □

6.3 Bewijs: NPDA's zijn strikt krachtiger dan DPDA's

Bewijs. Het is duidelijk dat elke DPDA een NPDA is. Dus $L(\text{DPDA}) \subseteq L(\text{NPDA})$. We tonen dat de inclusie strikt is.

Neem de taal

$$L = \{a^n b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}.$$

Deze taal is context-vrij, want een NPDA kan niet-deterministisch kiezen of hij b's één-op-één of twee-op-één vergelijkt met de a's. Stel uit het ongerijmde dat L deterministisch context-vrij is. Dan kan men uit een DPDA voor L een automaat construeren die ook de keuze kan afdwingen die nodig is om een taal van het type $a^n b^n c^n$ context-vrij te maken. Dat zou impliceren dat $\{a^n b^n c^n \mid n \geq 0\}$ context-vrij is. Maar die taal is niet context-vrij volgens het pumping lemma voor CFL's. Tegenspraak. Dus er bestaan context-vrije talen die door een NPDA maar niet door een DPDA herkend worden. □

7 Hoofdstuk 8: Eigenschappen van context-vrije talen

7.1 Lemma: een CNF-derivatieboom van diepte m heeft hoogstens 2^{m-1} bladeren

Bewijs. Neem een grammatica in Chomsky Normal Form. In zo'n grammatica heeft elke productie de vorm

$$A \to BC$$

of

$$A \to a.$$

Dus een interne knoop met variabelen heeft hoogstens twee kinderen die opnieuw variabelen zijn. Een variabele die rechtstreeks een terminal produceert, geeft één terminaal blad.

Bekijk de boom die je krijgt door alleen de variabele knopen te tellen. Dat is een binaire boom. Als de oorspronkelijke derivatieboom diepte m heeft, dan heeft deze binaire boom diepte hoogstens $m-1$, want de laatste stap naar terminals telt niet meer als variabele vertakking.

Een binaire boom van diepte d heeft hoogstens 2^d bladeren. Dit bewijs je eenvoudig met inductie op d: voor $d = 0$ is er hoogstens één blad, dus $1 = 2^0$. Voor $d+1$ kan de boom in hoogstens twee deelbomen van diepte d worden opgesplitst. Elke deelboom heeft volgens de inductiehypothese hoogstens 2^d bladeren. Samen zijn dat hoogstens

$$2 \cdot 2^d = 2^{d+1}$$

bladeren.

Met $d = m - 1$ krijgen we dus dat een CNF-derivatieboom van diepte m hoogstens

$$2^{m-1}$$

bladeren heeft. Omdat de bladeren precies de terminals van het afgeleide woord vormen, geldt voor elk woord w met zo'n derivatieboom:

$$|w| \leq 2^{m-1}.$$

□

7.2 Bewijs: pumping lemma voor context-vrije talen

Bewijs. Neem een oneindige context-vrije taal L met een CFG G in Chomsky Normal Form. Laat $n = |V|$. In Chomsky Normal Form is elke derivatieboom binair. Een boom van diepte i heeft maximaal 2^{i-1} bladeren. Neem een woord $w \in L$ met $|w| \geq 2^n$. Dan heeft elke derivatieboom voor w een tak met meer dan n variabelen. Volgens het duivenhokprincipe komt op die tak een variabele A minstens twee keer voor. Neem het laagste herhaalde voorkomen. Dan krijgen we een afleiding van de vorm

$$S \Rightarrow^* uAz \Rightarrow^* uvAyz \Rightarrow^* uvxyz.$$

Het stuk tussen de twee voorkomens van A vormt een lus in de derivatieboom. Die lus kan nul, één of meerdere keren worden gebruikt. Daarom geldt voor alle $i \geq 0$:

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

Bovendien geldt $|vxy| \leq 2^n$ en minstens één van v, y is niet leeg. □

7.3 Bewijs: pumping lemma voor lineaire CFG's

Bewijs. In een lineaire CFG bevat elke rechterkant hoogstens één variabele. Daarom heeft een derivatieboom slechts één centrale keten van variabelen. Terminals kunnen links en rechts van die keten worden toegevoegd, maar er ontstaat geen willekeurige vertakking met meerdere variabelen.

Voor een voldoende lang woord moet op die keten een variabele herhaald worden. Het stuk tussen de twee voorkomens van die variabele kan worden gepompt. Omdat de grammatica lineair is, liggen de pompbare delen aan de buitenkant van het deelwoord dat door die variabele wordt gegenereerd. Daarom krijgt men een sterkere vorm van het pumping lemma dan bij algemene CFL's. □

7.4 Bewijs: $\{a^n b^n c^n \mid n \geq 0\}$ is niet context-vrij

Bewijs. Stel dat

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

context-vrij is. Laat m de pumping-lengte zijn. Neem

$$w = a^m b^m c^m.$$

Volgens het pumping lemma kunnen we $w = uvxyz$ schrijven met $|vxy| \leq m$ en $|vy| \geq 1$. Omdat $|vxy| \leq m$, kan vxy niet tegelijk a's, b's en c's bevatten. Het ligt dus in hoogstens twee opeenvolgende blokken. Als we pompen met $i = 2$, verandert het aantal symbolen in hoogstens twee blokken, terwijl minstens één ander blok gelijk blijft. Dan zijn de aantallen a's, b's en c's niet meer allemaal gelijk. Dus $uv^2xy^2z \notin L$. Dat is een tegenspraak. Dus L is niet context-vrij. □

7.5 Bewijs: $\{a^n b^j \mid n = j^2\}$ is niet context-vrij

Bewijs. Stel dat de taal context-vrij is en laat m de pumping-lengte zijn. Neem

$$w = a^{m^2} b^m.$$

Voor elke decompositie $w = uvxyz$ met $|vxy| \leq m$ en $|vy| \geq 1$ liggen v en y ofwel volledig in het a-blok, volledig in het b-blok, of rond de grens. Bij pompen verandert het aantal a's en/of b's slechts lineair in de pompfactor. De voorwaarde $n = j^2$ legt echter een kwadratisch verband op. Door bijvoorbeeld naar $i = 0$ of $i = 2$ te gaan, wordt dat verband verbroken. Het resulterende woord zit dus niet meer in de taal. Dit spreekt het pumping lemma tegen. Daarom is de taal niet context-vrij. □

7.6 Bewijs: CFL's zijn gesloten onder unie, concatenatie en ster

Bewijs. Laat $G_1 = (V_1, T_1, S_1, P_1)$ en $G_2 = (V_2, T_2, S_2, P_2)$ CFG's zijn met disjuncte variabelen.

Voor unie voegen we een nieuw startsymbool S toe met

$$S \to S_1 \mid S_2.$$

Voor concatenatie voegen we

$$S \to S_1 S_2$$

toe. Voor ster voegen we

$$S \to S_1 S \mid \lambda$$

toe. De afleidingen in de nieuwe grammatica's komen precies overeen met respectievelijk kiezen tussen de twee talen, eerst een woord uit de eerste taal en daarna een woord uit de tweede taal nemen, of een eindig aantal woorden uit dezelfde taal na elkaar nemen. Dus CFL's zijn gesloten onder unie, concatenatie en ster. □

7.7 Bewijs: CFL's zijn gesloten onder doorsnede met reguliere talen

Bewijs. Laat L_1 context-vrij zijn en herkend worden door een NPDA M_1. Laat L_2 regulier zijn en herkend worden door een DFA M_2. We bouwen een NPDA met toestandenparen (q, p). De eerste component simuleert de NPDA M_1 en de tweede component simuleert de DFA M_2 op hetzelfde gelezen symbool. De stack wordt alleen gebruikt door de NPDA-component. Een toestand (q, p) is accepterend precies wanneer q accepterend is in M_1 en p accepterend is in M_2. Dan accepteert de productautomaat precies de woorden die door beide machines geaccepteerd worden. Dus $L_1 \cap L_2$ is context-vrij. □

7.8 Bewijs: CFL's zijn niet gesloten onder doorsnede en complement

Bewijs. Neem

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

en

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

Beide talen zijn context-vrij. Hun doorsnede is

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

en die taal is niet context-vrij. Dus CFL's zijn niet gesloten onder doorsnede.

Als CFL's gesloten waren onder complement, dan zouden ze via De Morgan ook gesloten zijn onder doorsnede:

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

Omdat ze wel gesloten zijn onder unie, zou complement-sluiting dus doorsnede-sluiting impliceren. Maar doorsnede-sluiting is vals. Dus CFL's zijn ook niet gesloten onder complement. □

7.9 Bewijs: leegheid van een CFL is beslisbaar

Bewijs. Gegeven een CFG G, voeren we het algoritme voor nuttige variabelen uit. Na dit algoritme blijft het startsymbool S over als en slechts als S een terminaal woord kan genereren. Als S verwijderd wordt, bestaat er geen enkel woord w met $S \Rightarrow^* w$, dus is $L(G) = \emptyset$. Als S overblijft, bestaat er minstens één terminaal woord dat uit S kan worden afgeleid, dus is de taal niet leeg. Omdat het algoritme eindig is, is leegheid beslisbaar. □

7.10 Bewijs: membership voor CFG's is beslisbaar

Bewijs. Neem een CFG G en een woord w. Zet G om naar Chomsky Normal Form, met aparte behandeling van λ. Voor een woord van lengte $n > 0$ heeft elke afleiding in Chomsky Normal Form precies $2n - 1$ relevante knopen in de parse tree en dus een begrensde grootte. Er zijn maar eindig veel zulke parse trees. Een algoritme kan alle mogelijke bomen of alle mogelijke afleidingen tot die grootte doorzoeken. Het accepteert als één ervan yield w heeft, en verwerpt anders. Dus membership voor CFG's is beslisbaar. □

7.11 Bewijs: eindigheid van een CFL is beslisbaar

Bewijs. Gegeven een CFG G, verwijder eerst nutteloze producties, λ-producties en unit-producties. Maak daarna een dependency graph waarin er een pijl $X \to Y$ is als Y voorkomt in de rechterkant van een productie van X.

Als er een cyclus bereikbaar is vanuit S, dan kan die cyclus herhaald worden. Omdat de grammatica vereenvoudigd is en alle variabelen nuttig zijn, levert herhaling van de cyclus steeds langere terminale woorden op. Dan is $L(G)$ oneindig.

Als er geen cyclus is, dan is de dependency graph acyclisch. Elke afleiding heeft dan een begrensde lengte, en er kunnen maar eindig veel terminale woorden worden gegenereerd. Dus $L(G)$ is eindig. Omdat cykels in een eindige graaf beslisbaar zijn, is eindigheid van CFL's beslisbaar. □

8 Hoofdstuk 9: Turingmachines

8.1 Bewijs: een TM met stay-optie is equivalent aan een standaard TM

Bewijs. Een TM met stay-optie mag naast links en rechts bewegen ook blijven staan. Een stay-stap

$$\delta(q_i, a) = (q_j, b, S)$$

kan door een standaard TM worden gesimuleerd met twee stappen. Eerst schrijft de machine b en beweegt ze naar rechts:

$$\delta(q_i, a) = (\hat{q}_j, b, R).$$

Daarna beweegt ze onmiddellijk terug naar links zonder het symbool te veranderen:

$$\delta(\hat{q}_j, x) = (q_j, x, L)$$

voor elk tapesymbool x. Zo eindigt de kop op dezelfde cel als bij de stay-stap. Elke stay-stap kan dus eindig worden gesimuleerd. Daarom hebben beide modellen dezelfde rekenkracht. □

8.2 Bewijs: semi-infinite tape TMs zijn equivalent aan gewone TMs

Bewijs. Een gewone TM kan een semi-infinite tape rechtstreeks simuleren door nooit links van de linkermarker te bewegen. Voor de andere richting simuleert een semi-infinite tape TM een tweezijdig oneindige tape met twee sporen. Het eerste spoor bevat de cellen rechts van de startpositie. Het tweede spoor bevat de cellen links van de startpositie, maar in omgekeerde volgorde. De positie van de gesimuleerde kop wordt bijgehouden met markeringen. Wanneer de gesimuleerde kop over de oorsprong gaat, wisselt de machine van spoor. Elke stap van de tweezijdige tape kan zo met eindig veel stappen worden gesimuleerd. Dus de modellen zijn equivalent. □

8.3 Bewijs: multi-tape TMs zijn equivalent aan gewone TMs

Bewijs. Elke gewone TM is triviaal een multi-tape TM met één tape. Omgekeerd kan een gewone TM de inhoud van alle tapes op één tape coderen, gescheiden door speciale markers. Voor elke gesimuleerde stap scant de gewone TM de codering, vindt de gemarkeerde koppositities, leest de symbolen, bepaalt de multi-tape transitie, past de symbolen aan en verplaatst de kopmarkeringen. Als een gesimuleerde tape moet uitbreiden, past de gewone TM de codering aan door symbolen op te schuiven. Elke stap van de multi-tape TM kost eindig veel stappen van de gewone TM. Dus multi-tape TMs herkennen geen extra talen. □

8.4 Bewijs: multi-tape TMs zijn equivalent aan multi-track TMs

Bewijs. Een multi-tape TM met n tapes kan worden gesimuleerd door één tape met meerdere sporen. Voor elke gesimuleerde tape gebruiken we één spoor voor de inhoud en één spoor voor de positie van de kop. Een marker geeft aan waar de kop van elke gesimuleerde tape staat. Om één stap te simuleren, scant de machine de tape om alle markers te vinden, leest ze de bijbehorende symbolen, voert ze de transitie uit en verplaatst ze de markers. Daarom kan een multi-track TM elke multi-tape TM simuleren. De omgekeerde richting is triviaal, want elke track kan als aparte tape worden gezien. Dus beide modellen hebben dezelfde rekenkracht. □

8.5 Bewijs: niet-deterministische TMs zijn equivalent aan deterministische TMs

Bewijs. Een niet-deterministische TM accepteert een woord als er minstens één accepterend berekeningspad bestaat. Een deterministische TM kan alle mogelijke berekeningspaden systematisch doorzoeken. Dit gebeurt bijvoorbeeld breadth-first: eerst alle paden van lengte 1, dan alle paden van lengte 2, enzovoort. Omdat elk eindig accepterend pad uiteindelijk aan de beurt komt, zal de deterministische TM accepteren als de niet-deterministische TM accepteert. Als er geen accepterend pad is, accepteert de deterministische simulatie ook niet. Daarom herkennen deterministische en niet-deterministische TMs dezelfde klasse talen. □

9 Hoofdstuk 10: Recursief opsombare en recursieve talen

9.1 Bewijs: niet-recursief opsombare talen bestaan

Bewijs. Het bewijs gebruikt een telargument. Voor een eindig alfabet $\Sigma \neq \emptyset$ is Σ^* aftelbaar. Elke Turingmachine kan eindig worden beschreven, dus er zijn slechts aftelbaar veel Turingmachines. Daarom zijn er ook slechts aftelbaar veel talen die door Turingmachines herkend kunnen worden.

Maar de verzameling van alle talen over Σ is 2^{Σ^*}. Omdat Σ^* aftelbaar oneindig is, is 2^{Σ^*} overaftelbaar. Er zijn dus meer talen dan Turingmachines. Bijgevolg bestaan er talen die door geen enkele Turingmachine herkend worden. Die talen zijn niet recursief opsombaar. □

9.2 Bewijs: als L recursief is, dan zijn L en $\overline{L}$ recursief opsombaar

Bewijs. Als L recursief is, bestaat er een beslissende TM M die op elke input halt en correct ja of nee antwoordt. Een herkenner voor L simuleert M en accepteert precies wanneer M ja zegt. Een herkenner voor $\overline{L}$ simuleert dezelfde M en accepteert precies wanneer M nee zegt. Omdat M altijd halt, accepteren deze herkenners alle woorden in hun respectieve talen. Dus zowel L als $\overline{L}$ zijn recursief opsombaar. □

9.3 Bewijs: L is recursief als en slechts als L en $\overline{L}$ recursief opsombaar zijn

Bewijs. De ene richting volgt uit het vorige bewijs.

Voor de andere richting: stel dat L recursief opsombaar is en dat $\overline{L}$ ook recursief opsombaar is. Dan bestaat er een TM M die L herkent en een TM $\overline{M}$ die $\overline{L}$ herkent. Op input w simuleren we beide machines dovetailend: telkens één stap van M en één stap van $\overline{M}$. Omdat elk woord ofwel in L ofwel in $\overline{L}$ zit, zal één van beide machines uiteindelijk accepteren. Als M accepteert, accepteren we w. Als $\overline{M}$ accepteert, verwerpen we w. Deze procedure halt altijd en beslist L. Dus L is recursief. □

9.4 Bewijs: er bestaat een recursief opsombare taal waarvan het complement niet recursief opsombaar is

Bewijs. Som alle Turingmachines op als $M_1, M_2, \ldots$. Beschouw de diagonaaltaal

$$L = \{a^i \mid a^i \in L(M_i)\}.$$

Deze taal is recursief opsombaar: op input a^i simuleer je M_i op a^i en accepteer je als die simulatie accepteert.

Stel dat $\overline{L}$ ook recursief opsombaar is. Dan bestaat er een machine M_n met $L(M_n) = \overline{L}$. Bekijk nu a^n. Dan geldt:

$$a^n \in \overline{L} \iff a^n \notin L(M_n).$$

Maar omdat $L(M_n) = \overline{L}$, krijgen we:

$$a^n \in \overline{L} \iff a^n \notin \overline{L}.$$

Dat is onmogelijk. Dus $\overline{L}$ is niet recursief opsombaar. □

9.5 Bewijs: $\{a^n b^n c^n \mid n \geq 0\}$ is context-sensitief

Bewijs. We geven een constructieve schets. Een context-sensitieve grammatica mag producties gebruiken die de lengte niet verkleinen. Voor de taal $a^n b^n c^n$ kan men werken met een boodschappervariabele. Die beweegt eerst naar rechts door de b's, maakt bij de c's een extra bc aan, en stuurt daarna een boodschapper terug naar links om een extra a aan te maken. Door deze lus telkens te herhalen, worden a, b en c steeds samen toegevoegd. Alle producties behouden of vergroten de lengte. Daarom is de grammatica context-sensitief en is de taal context-sensitief. □

9.6 Bewijs: context-sensitieve talen vormen een strikte subset van de recursieve talen

Bewijs. Elke context-sensitieve taal is beslisbaar, bijvoorbeeld via een linear bounded automaton of door de begrensde zoekruimte van niet-verkortende grammatica's te doorzoeken. Dus elke context-sensitieve taal is recursief.

De inclusie is strikt via diagonalisatie. Codeer elke context-sensitieve grammatica G als een binair woord $e(G)$. Definieer

$$L = \{b \mid b = e(G) \text{ en } b \notin L(G)\}.$$

Deze taal is recursief, omdat membership voor context-sensitieve grammatica's beslisbaar is. Stel dat L zelf context-sensitief is. Dan bestaat er een context-sensitieve grammatica G_L met $L(G_L) = L$. Bekijk haar eigen encoding $e(G_L)$. Dan geldt:

$$e(G_L) \in L \iff e(G_L) \notin L(G_L).$$

Omdat $L(G_L) = L$, krijgen we een tegenspraak. Dus er bestaat een recursieve taal die niet context-sensitief is. □

10 Hoofdstuk 11: Onbeslisbaarheid

10.1 Bewijs: het haltingprobleem is niet beslisbaar

Bewijs. Stel uit het ongerijmde dat er een TM H bestaat die het haltingprobleem beslist. Op input $e(M)e(w)$ halt H altijd en antwoordt H ja als M halt op w, en nee anders.

Construeer nu een machine H'. Op input x kopieert H' de input zodat hij xx krijgt, en voert daarna H uit op xx. Als H ja antwoordt, gaat H' in een oneindige lus. Als H nee antwoordt, halt H'.

Voer nu H' uit op haar eigen encoding $e(H')$. Dan geldt:

$$H' \text{ halt op } e(H')$$

als en slechts als

$$H \text{ op } e(H')e(H') \text{ nee antwoordt}$$

als en slechts als

$$H' \text{ niet halt op } e(H').$$

Dit is een tegenspraak. Dus zo'n beslisser H bestaat niet. Het haltingprobleem is onbeslisbaar. □

10.2 Bewijs: het state-entry-probleem is onbeslisbaar

Bewijs. We reduceren het haltingprobleem naar het state-entry-probleem. Gegeven een TM M en input w, construeren we een machine M' met een nieuwe toestand $\hat{q}$. M' simuleert M op w. Wanneer M zou stoppen, gaat M' naar de nieuwe toestand $\hat{q}$. Dan geldt:

$$M \text{ halt op } w \iff M' \text{ bereikt } \hat{q}.$$

Als state-entry beslisbaar was, konden we daarmee het haltingprobleem beslissen. Dat kan niet. Dus state-entry is onbeslisbaar. □

10.3 Bewijs: blank-tape halting is onbeslisbaar

Bewijs. We reduceren opnieuw vanuit het haltingprobleem. Gegeven M en w, construeren we een machine M' die op een lege tape eerst w schrijft en daarna M op w simuleert. Dan geldt:

$$M' \text{ halt op lege tape} \iff M \text{ halt op } w.$$

Een beslisser voor blank-tape halting zou dus een beslisser voor het haltingprobleem opleveren. Dat is onmogelijk. Dus blank-tape halting is onbeslisbaar. □

10.4 Bewijs: er bestaat een welgedefinieerde niet-berekenbare functie $f : \mathbb{N} \to \mathbb{N}$

Bewijs. Fixeer een tape-alfabet, bijvoorbeeld $\Gamma = \{0, 1, \square\}$. Voor elk aantal toestanden n bestaan er maar eindig veel Turingmachines met n toestanden. Definieer

$$f(n) = \max\{m \in \mathbb{N} \mid \text{er bestaat een TM met } n \text{ toestanden die op } \lambda \text{ halt na } m \text{ stappen}\}.$$

Deze functie is welgedefinieerd omdat het maximum over een eindige verzameling loopt.

Stel dat f berekenbaar is. Dan kunnen we blank-tape halting beslissen: voor een TM met n toestanden simuleren we hoogstens $f(n)$ stappen. Als de machine dan nog niet is gestopt, zal ze nooit stoppen. Maar blank-tape halting is onbeslisbaar. Dus f is niet berekenbaar. □

10.5 Bewijs: de vraag of $L(M)$ eindig is, is onbeslisbaar

Bewijs. We reduceren vanuit het haltingprobleem. Gegeven M en w, construeren we een machine M' die op elke input eerst die input negeert, daarna M op w simuleert, en accepteert als M halt.

Als M niet halt op w, accepteert M' geen enkel woord en is

$$L(M') = \emptyset,$$

dus eindig. Als M wel halt op w, accepteert M' elke input en is

$$L(M') = \Sigma^*,$$

dus oneindig. Een beslisser voor eindigheid van $L(M')$ zou dus het haltingprobleem beslissen. Dat is onmogelijk. Daarom is de vraag of $L(M)$ eindig is onbeslisbaar. □

10.6 Bewijs: Rice's theorem

Bewijs. Laat P een niet-triviale semantische eigenschap van Turingmachines zijn. Semantisch betekent dat P alleen afhangt van $L(M)$, niet van de syntactische vorm van M. Niet-triviaal betekent dat sommige Turingmachines de eigenschap hebben en andere niet.

Kies een machine M_P die de eigenschap heeft en een machine $M_{\neg P}$ die de eigenschap niet heeft. We reduceren het haltingprobleem naar het beslissen van P. Gegeven M en w, construeren we een machine N. Op input x simuleert N eerst M op w. Als die simulatie halt, gedraagt N zich als M_P op x. Als die simulatie niet halt, zal N nooit aan dat gedrag beginnen en krijgt N de semantiek van de gekozen niet-P-machine of van de lege taal, afhankelijk van de constructie.

Een beslisser voor P zou kunnen bepalen of N eigenschap P heeft, en dus of M halt op w. Dat zou het haltingprobleem beslissen. Tegenspraak. Dus elke niet-triviale semantische eigenschap van Turingmachines is onbeslisbaar. □

10.7 Bewijs: Post Correspondence Problem is onbeslisbaar

Bewijs. Het bewijs gebeurt via een reductie vanuit een reeds bekend onbeslisbaar probleem voor grammatica's. Bij een PCP-instantie krijgen we twee rijen strings. We zoeken een niet-lege rij indices zodat de concatenatie van de bovenste strings gelijk is aan de concatenatie van de onderste strings.

Men construeert uit een grammatica en een doelwoord een PCP-instantie waarvan de tegels de afleiding simuleren. Sommige tegels starten de afleiding, sommige kopiëren terminals, sommige passen producties toe, en een laatste tegel sluit de simulatie af. De tweede rij loopt telkens een beetje achter op de eerste rij. Alleen wanneer de gesimuleerde afleiding correct is, kunnen beide rijen exact gelijk eindigen.

Er bestaat dus een PCP-oplossing als en slechts als de oorspronkelijke grammatica het doelwoord genereert. Omdat dat bronprobleem onbeslisbaar is, is PCP ook onbeslisbaar. □

10.8 Bewijs: ambiguïteit van CFG's is onbeslisbaar

Bewijs. We reduceren vanuit PCP. Neem een PCP-instantie met bovenstrings v_i en onderstrings w_i. Bouw een grammatica met startsymbool

$$S \to S_v \mid S_w.$$

De variabele S_v genereert indexreeksen samen met de bijbehorende concatenatie van v-strings. De variabele S_w doet hetzelfde voor de w-strings.

Als de PCP-instantie een oplossing heeft, bestaat er een indexreeks waarvoor de v-concatenatie gelijk is aan de w-concatenatie. Dan kan hetzelfde woord op twee manieren worden afgeleid: één keer via S_v en één keer via S_w. De grammatica is dan ambigu.

Omgekeerd kan ambiguïteit alleen ontstaan wanneer zo'n zelfde indexreeks dezelfde boven- en onderconcatenatie oplevert. Dat is precies een PCP-oplossing. Dus een beslisser voor ambiguïteit zou PCP beslissen. Omdat PCP onbeslisbaar is, is ambiguïteit van CFG's onbeslisbaar. □

10.9 Bewijs-sjabloon: andere onbeslisbare grammatica-problemen

Bewijs. Veel onbeslisbare vragen over grammatica's worden bewezen met dezelfde reductiestrategie. Men vertrekt van een bekend onbeslisbaar probleem, zoals PCP. Daarna construeert men effectief één of meer grammatica's zodat het nieuwe probleem een ja-antwoord heeft als en slechts als de oorspronkelijke PCP-instantie een oplossing heeft. Voorbeelden zijn vragen over doorsnede-leegheid, inclusie, equivalentie of regulariteit van context-vrije grammatica's. Omdat een beslisser voor het nieuwe probleem ook PCP zou beslissen, zijn deze problemen onbeslisbaar. □