AUTOMATEN
&
BEREKENBAARHEID

Jensen Bernard

2016

This page is intentionally left blank.

Voorwoord

Dit document bevat mogelijke examenvragen voor het vak *Automaten en Berekenbaarheid*[1], gedoceerd aan de Katholieke Universiteit Leuven. In geen enkel geval wil dit document een vervanging zijn voor de cursus. De cursustekst, *Automaten en berekenbaarheid*, geschreven door *Bart Demoen*, is zeer goed en het is zeker aangeraden deze grondig door te nemen voor u begint aan de volgende vraagstukken. Ik heb dit document opgesteld tijdens het studeren van het vak, om op deze manier een overzicht te hebben van mogelijke examenvragen die we kunnen verwachten in 2016. Vele vragen uit verschillende jaren komen zeer sterk overeen, daarom kan het zeker geen kwaad om deze extra aandacht te geven.

Het is ook mogelijk dat er verwezen wordt naar delen uit de cursus. De meeste van deze verwijzingen zijn geschreven in December 2015. De meest recente versie was op dit moment de uitgave van 2013. Indien een nieuwere versie beschikbaar is, is het mogelijk dat de pagina's niet meer overeenstemmen. Aarzal niet om deze, zowel als mogelijke inhoudelijke fouten, aan te geven of aan te passen op Github.

Ik heb vele studenten gehoord die vaak het nut niet inzien van dit vak, of dit totaal niet interessant vinden. Het is belangrijk eerst het voorwoord in de cursus eens te lezen, om een goed beeld te hebben van waar we nu eigenlijk mee bezig zijn. Indien u nog steeds van mening bent dat dit een enorm saai vak is, dan raad ik aan om ca. 2u te pauzeren om *The Imitation Game*[2] te kijken. Kom daarna terug en alles zal veel interessanter lijken dan voordien.

Thanks to *Robin Haveneers* voor de hulp.

Jensen Bernard

[1] Course G0P84a and G0P85a.
[2] Een film over Alan Turing, 2014.

Inhoudsopgave

This page is intentionally left blank.

1 Talen en Automaten

1.1 Inleiding

Dit hoofdstuk start met het kennismaken met talen en automaten. Het grootste deel van de leerstof uit dit hoofdstuk kan teruggevonden in de vragen, maar toch zijn er enkele puntjes die niet aan bod komen. Dit wil echter niet zeggen dat deze nooit zullen worden gevraagd op een examen. Ik raad aan de volgende secties achteraf eens te bekijken (sommige zijn misschien handig op voorhand):

1. Kennismaking - *p4-p19*[3]
2. Van reguliere expressie naar *NFA* - *p20*
3. Doorsnede, complement en verschil van *DFA's* - *p15*
4. Reguliere expressies en lexicale analyse - *p47*
5. Varianten van *DFA's* - *p49-p52*
6. Equivalentie van *CFG* en *PDA* - *p68*
7. Algebra van contextvrije talen - *p75*

1.2 Equivalentie-relaties en -klassen

Aangezien velen problemen hadden met Myhill-Nerode relaties, is hier wat achtergrond informatie om dit geheel beter te begrijpen. Stel dat we in het bezit zijn van vijf ballen. Hun volgorde is vast aangezien ze op een rij op tafel liggen. We kunnen dus spreken over bal_i met $i \in \{1, 2, 3, 4, 5\}$. Deze duidt dan op de bal op de i^{de} plaats.

Laten we nu aannemen dat de oneven ballen rood zijn, de even ballen blauw. We kunnen op dit moment de ballen verdelen in groepen, namelijk de rode ($\{1, 3, 5\}$) en de blauwe ($\{2, 4\}$). We gebruiken hier nu al (zonder het misschien te beseffen) equivalentie-relaties en -klassen!

Laten we de equivalentierelatie $\sim_{kleur}$ definiëren, die nagaat of twee ballen dezelfde kleur hebben. Zo is bijvoorbeeld $bal_1 \sim_{kleur} bal_3$ waar, maar $bal_2 \sim_{kleur} bal_5$ niet!

Een equivalentieklasse is dan eigenlijk de klasse (of verzameling) met alle elementen die voldoen aan een bepaalde equivalentierelatie waarvan één element constant is. Zo kunnen we de volgende equivalentieklasses berekenen[4]. We nemen even $i = bal_i$ voor een makkelijkere notatie.

$$1_{\sim_{kleur}} = \{y | 1 \sim_{kleur} y\} = \{1, 3, 5\}$$
$$2_{\sim_{kleur}} = \{y | 2 \sim_{kleur} y\} = \{2, 4\}$$

[3] Ookal komt dit niet expliciet aan bod, wanneer je de vragen beheerst, beheers je ook dit deel. Deze theorie komt overal terug. Je kan dit best lezen op voorhand.

[4] Er bestaan er heel wat meer dan dit.

Houdt dit goed in het achterhoofd wanneer we gaan werken met Myhill-Nerode relaties aangezien deze verder bouwen op deze concepten. In deze cursus zijn x en y strings en worden deze als equivalent beschouwd op basis van (bv. een *DFA*). Zo kunnen we de equivalentierelatie $\sim_{DFA}$ beschouwen die zegt dat

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

Een ander voorbeeld is $x \sim_L y$ indien $x \in L$ en $y \in L$.

Vraag 1. Beschrijf in detail de transformatie van een deterministische eindige toestandsautomaat naar een equivalente deterministische eindige toestandsautomaat met een minimaal aantal toestanden. Beschrijf de notie van equivalentie van automaten in deze context en argumenteer waarom er geen kleinere equivalente deterministische eindige toestandsautomaat bestaat.

Minimale DFA

Voor een gegeven reguliere taal L bestaan er meestal veel *DFA's* die de taal bepalen. Het is belangrijk om kleine machines te maken aangezien deze efficiënter te werk gaan. De kleinst mogelijke *DFA* die de taal L bepaalt, noemen we de minimale *DFA*. Vooraleer we dieper ingaan op het algoritme om deze minimale *DFA* op te stellen, moeten we de term *f-equivalentie* uitleggen.

> **Definitie 1** (f-equivalentie)**.** *Twee toestanden p en q zijn f-gelijk indien $\forall w \in \Sigma^* : \delta^*(p, w) \in F \rightarrow \delta^*(q, w) \in F$. Twee toestanden zijn f-verschillend indien ze niet f-gelijk zijn.*

Het vinden van *f-equivalente* toestanden in een *DFA* is relatief simpel. Om te beginnen weten we dat elke toestand, verschillend van de aanvaardbare eindtoestanden, *f-verschillend* is van de aanvaardbare eindtoestanden. Indien dit niet zo zou zijn, zouden ook de toestanden die geen eindtoestanden zijn, eindtoestanden zijn. Dit is in contradictie met wat we net aannamen.

Daarna kunnen we alle mogelijke koppels van toestanden overlopen en nagaan of er een string s bestaat, die beide toestanden naar twee *f-verschillende* toestanden brengen. In dit geval zijn deze toestanden ook *f-verschillend*. Indien zo een string s niet bestaat, zijn ze *f-gelijk*.

Algoritme

Het principe van het algoritme is simpel. We beginnen met het verwijderen van alle niet bereikbare toestanden. Hierna gaan we alle *f-equivalente* toestanden samennemen. In wat nu volgt wordt het algoritme verder uitgelegd, er van uitgaande dat alle niet-bereikbare toestanden reeds weggelaten zijn. Maak hier zeker oefeningen op! Het is praktisch onmogelijk dit goed uit te leggen als je het niet volledig snapt of zelf niet kunt.

Init Stel een graaf V op, zonder bogen, waarvan de nodes gelijk aan de toestanden van de *DFA*. Verbind nu elke node, verschillend van de accepterende eindtoestanden, met alle mogelijk eindtoestanden en label deze met ϵ. Deze bogen duiden aan dat de verbonden nodes *f-verschillend* zijn[5].

Repeat Ga nu alle koppels van toestanden af die niet verbonden zijn in onze net opgestelde graaf V. Check nu, in de originele *DFA*, voor elk karakter α in het alfabet, naar welke toestanden de toestanden uit het koppel wijzen. Zijn deze twee toestanden verbonden met elkaar in de graaf V? Dan hebben we een

[5]Zoals eerder vermeld zijn alle niet-aanvaardbare eindtoestanden *f-verschillend* met de aanvaardbare eindtoestanden.

conflict en voegen we een boog toe tussen de twee originele toestanden in V met label α. Ga door tot we geen bogen of labels meer moeten toevoegen.

Final Stel de complementsgraaf op van de gevonden graaf V. De complementsgraaf toont nu aan welke nodes (die overeenkomen met de toestanden) samen genomen kunnen worden. Wanneer bv. node 4 en 7 samenhoren, maken we een node $(4, 7)$. Voeg nu gewoon alle bogen van de originele *DFA* toe en je hebt de minimale *DFA*.

Kleinere DFA dan de minimale DFA

Aangezien we weten dat de minimale *DFA*, bekomen uit het algoritme dat we zonet hebben beschreven, geen *f-equivalente* toestanden heeft, kunnen we geen kleinere *DFA* vinden.

> **Definitie 2.** *Als $DFA_1 = (Q_1, \Sigma, \delta_1, q_s, F_1)$ een machine 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.*

Bewijs. Laat DFA_1, zoals hierboven beschreven, de volgende toestanden hebben $\{q_1, ..., q_n\}$, waarbij q_s de starttoestand is. Stel dat $DFA_2 = (Q_2, \Sigma, \delta_2, p_s, F_2)$ minder toestanden heeft dan DFA_1. Beide bepalen wel dezelfde taal L.

Vermits in DFA_1 elke toestand bereikbaar is, bestaan er strings s_i met $i = 1 \dots n$ zodanig dat $\delta_1^*(q_s, s_i) = q_i \in Q_1$. Vermits DFA_2 minder toestanden heeft, moet voor een $i \neq j$ er een string s_j bestaan die in dezelfde toestand eindigt[6]. We zeggen dus dat $\delta_2^*(p_s, s_i) = \delta_2^*(p_s, s_j)$. Vermits q_i en q_j *f-verschillend* zijn, bestaat een string v zodanig dat $\delta_1^*(q_i, v) \in F_1$ en $\delta_1^*(q_j, v) \notin F_1$ of omgekeerd.

Dus ook $\delta_1^*(q_i, s_i v) \in F_1 \delta_1^*(q_j, s_i v) \notin F_1$ of omgekeerd. Dit betekent dat van de strings $s_i v$ en $s_j v$ de DFA_1 er juist één accepteert.

Maar: $\delta_2^*(p_s, s_i v) = \delta_2^*(\delta_2^*(p_s, s_i), v) = \delta_2^*(\delta_2^*(p_s, s_j), v) = \delta_2^*(p_s, s_j v)$ hetgeen betekent dat DFA_2 ofwel beide strings $s_i v$ en $s_j v$ accepteert, of beide verwerpt. Dus kunnen DFA_1 en DFA_2 niet dezelfde taal bepalen. □

Equivalente machines

We kunnen deze notie van *f-equivalentie* meteen in verband brengen met de equivalentie van automaten. Twee automaten zijn namelijk equivalent indien deze dezefde taal bepalen. We kunnen dus zeggen dat hun start toestanden *f-equivalent* moeten zijn. Indien twee machines equivalent zijn, zijn de minimale *DFA's* isomorf[7].

[6] Aangezien deze minder toestanden heeft en er dus meerdere strings moeten eindigen in één toestand.

[7] We bewijzen dit in een andere vraag met behulp van Myhill-Neroderelaties.

Vraag 2. Beschrijf in detail de transformatie van een niet-deterministische eindige toestandsautomaat naar een equivalente deterministische eindige toestandsautomaat. Beschrijf de notie van equivalentie van automaten in deze context en argumenteer waarom de transformatie correct is. Bespreek de uitspraak "deze transformatie is (niet) deterministisch". Kan er zo een *DFA* bestaan met minder toestanden dan de *NFA*?

NFA vs DFA

Een *NFA (niet-deterministische eindige automaat)* laat ϵ-overgangen en meerdere bogen met hetzelfde label vanuit dezelfde knoop toe. Bij een *NFA* kunnen er meerdere mogelijkheden zijn voor de volgende stap. De volgorde van de toestanden die zullen gevolgd worden ligt dus niet vast.
Een *DFA (deterministische eindige automaat)* kan geen ϵ-overgangen hebben en een symbool $\alpha \in \Sigma$ mag hoogstens op één uitgaande boog per toestand staan. De volgende stap van het proces, de overgang naar de volgende toestand, ligt dus vast en er is maar één mogelijkheid.

Algoritme

Gegeven: een $NFA = (Q_n, \Sigma, \delta_n, q_{sn}, F_n)$
Gevraagd: een $DFA = (Q_d, \Sigma, \delta_d, q_{sd}, F_d)$ zodanig dat $L_{NFA} = L_{DFA}$.
Constructie: Vermits de *NFA* ϵ-bogen heeft, zullen we enkele van de toestanden van de *NFA* moeten samennemen. We kunnen dus al stellen dat elke toestand in de *DFA* een verzameling van toestanden van de *NFA* zal zijn:

$$Q_d = \mathcal{P}(Q_n)$$

Verder weten we ook dat de eindtoestand van de *DFA* altijd een eindtoestand van de *NFA* bevat:

$$F_d = \{S | S \in Q_d, S \cap F_n \neq \emptyset\}$$

Nu rest er ons alleen nog de transitiefunctie δ_d om uit te werken:

- We weten al dat $\delta_d : (\mathcal{P}(Q_n) \times \Sigma) \to \mathcal{P}(Q_n)$.
- We beginnen met een nieuwe afbeelding in te voeren: $eb : Q_n \to \mathcal{P}(Q_n)$. Deze functie staat voor *epsilon bereikbaar*. Het resultaat van de afbeelding $eb(q)$ is dus de verzameling van toestanden in de *NFA* die met nul, één of meer ϵ-bogen bereikbaar zijn vanuit q.
- We gaan nu de definitie van eb liften naar $\mathcal{P}(Q_n)$. Voor een $\mathcal{Q} \in \mathcal{P}(Q_n)$ geldt dat: $eb(\mathcal{Q}) = \cup_{q \in \mathcal{Q}} eb(q)$.
- We zullen δ_n liften op dezelfde manier naar $\mathcal{P}(Q_n)$. Dit wil zeggen dat voor $\mathcal{Q} \in \mathcal{P}(Q_n)$ geldt dat: $\delta_n(\mathcal{Q}, a) = \cup_{q \in \mathcal{Q}} \delta_n(q, a)$ met $a \in \Sigma$.

We definiëren δ_d dan als volgt:

- Vanuit een toestand $\mathcal{Q}$ met $\mathcal{Q} \in Q_d = \{Q_{(d,1)}, Q_{(d,2)}, \ldots, Q_{(d,k)}\}$ in de *DFA* ga je naar de volgende toestand in de *DFA* door voor elke '*NFA-toestand*' in die $\mathcal{Q}$ (aangezien $\mathcal{Q} \in Q_d = \mathcal{P}(Q_n)$) eerst de overgangsfunctie

van de *NFA* te volgen en daarna de ϵ-bogen te volgen. Van al deze resulterende toestandsverzamelingen neem je dan de unie:

$$\delta_d(\mathcal{Q}, a) = eb(\delta_n(\mathcal{Q}, a)) \text{ voor elke } \mathcal{Q} \in Q_d$$

- Tenslotte definiëren we nog de starttoestand van de *DFA*:

$$q_{sd} = eb(q_{sn})$$

Equivalentie

Twee automaten zijn equivalent als ze dezelfde taal bepalen. Het algoritme dat we zonet hebben beschreven, is in staat om alle ϵ-bogen weg te werken. Deze veranderen niets aan de taal die bepaalt wordt door de automaat. De gegeven *NFA* en de bekomen *DFA* zijn dus equivalent ($L_{DFA} = L_{NFA}$).

Deterministische transformatie

Aangezien deze werkwijze alle mogelijke toestanden genereert, alsook alle mogelijk bogen, is er maar één mogelijke uitkomst. Het is mogelijk om de toestanden en bogen van de *DFA* op een andere volgorde te construeren, maar dit zal steeds leiden tot dezelfde *DFA*.

DFA met minder toestanden dan een NFA

Het is mogelijk om een equivalente *DFA* te construeren uit een*NFA* die minder toestanden bevat. Dit is bijvoorbeeld het geval wanneer er meerdere toestanden zijn die enkel met ϵ-bogen verbonden zijn. De werkwijze als boven beschreven zal echter altijd meer toestanden hebben dan de *NFA* aangezien $Q_d = \mathcal{P}(Q_n)$. Dit wil niet zeggen dat er naar elk van deze toestanden een boog is. Deze kan bijna altijd nog geminimaliseerd worden.

Vraag 3. Bewijs dat een eindige toestandsautomaat altijd een taal herkent die door een reguliere expressie wordt beschreven. Doe dat door een eindige toestandsautomaat te transformeren naar een gegeneraliseerde niet-deterministische eindige toestandsautomaat met slechts twee toestanden.
Kan voor een *PDA* ongeveer hetzelfde gedaan worden door een gegeneraliseerde *PDA* op te stellen?

Voor we hier aan beginnen definiëren we de *GNFA* als volgt:

Definitie 3 (GNFA)**.** *Een* GNFA *(gegeneraliseerde niet-deterministische eindige automaat) is een eindige toestandsmachine met de volgende wijzigingen en beperkingen:*

1. *Er is slechts 1 eindtoestand $\neq$ starttoestand*
2. *Er is juist 1 boog van de starttoestand naar elke andere toestand en er komen geen pijlen aan (buiten de startpijl)*
3. *Er is juist 1 boog van elke toestand naar de eindtoestand, maar er vertrekken geen pijlen vanuit de eindtoestand*
4. *Tussen elke andere 2 toestanden is er juist 1 boog in beide richtingen*
5. *Er is juist 1 boog van elke andere toestand naar zichzelf*
6. *De bogen hebben als label een reguliere expressie*

In wat volgt bewijzen we dat een eindige toestandsautomaat altijd een taal herkent die door een reguliere expressie wordt beschreven. We volgen volgend stappenplan:

$$\text{NFA} \rightarrow \text{GNFA} \rightarrow \text{GNFA met 2 toestanden} \rightarrow \text{reguliere expressie}$$

In de opgave spreken ze van een "eindige toestandsautomaat", we gaan er hier echter vanuit dat we starten vanaf een *NFA*. Dit mag vermits elke *DFA* omgezet kan worden naar een *NFA*.

Bewijs. **NFA $\rightarrow$ GNFA**

We kunnen onze *NFA* omzetten naar een *GNFA* als volgt:

1. Maak een nieuwe starttoestand en een nieuwe (unieke) eindtoestand
2. Teken ϵ-bogen van de nieuwe begintoestand naar de oude begintoestand (van de *NFA*) en van elke oude eindtoestand (van de *NFA*) naar de nieuwe eindtoestand.
3. Teken ontbrekende bogen met een ϕ-label.
4. Als er tussen twee toestanden twee of meer parallele gerichte bogen zijn, neem die dan samen met als label de unie van de labels van de parallelle bogen.

Deze omzetting van *NFA* naar *GNFA* verandert de taal niet. We doen namelijk enkel correct manipulaties van de *NFA*.
Nu we een *GNFA* geconstrueerd hebben moeten we deze reduceren naar een GNFA met 2 toestanden.

Reduceren van de GNFA

Dit doen we door herhaaldelijk een willekeurige toestand X verschillend van de start- of eindtoestand te kiezen en deze knoop de verwijderen als volgt:

1. Kies toestanden A en B zodat er bogen zijn van[8]:
   - A naar B met label E_4
   - A naar X met label E_1
   - X naar zichzelf met label E_2
   - X naar B met label E_3
2. Vervang het label op de boog van A naar B door $E_4|E_1E_2^*E_3$.
3. Doe dit voor alle koppels A en B
4. Verwijder de knoop X met alle bogen die erin toekomen of vertrekken.

Indien er geen toestanden meer zijn, ga dan naar de volgende stap. Op dit moment hebben we een *GNFA* met twee toestanden.

Bepaal de reguliere expressie

De *GNFA* heeft nu exact twee toestanden (een start- en eindtoestand) met daartussen één boog. Deze boog heeft nu een reguliere expressie als label. Dit is de reguliere expressie die dezelfde taal bepaald als de oorspronkelijke *NFA*. □

Extra

In wat volgt bewijzen we nog dat de reductie met één enkele toestand de verzameling aanvaarde strings niet verandert. Hiervoor moeten we aantonen dat indien s aanvaard werd voor de reductie, deze ook na de reductie aanvaard wordt en indien s niet aanvaard werd voor de reductie, deze na de reductie ook niet aanvaard wordt.

String s wordt nog steeds aanvaard

Bewijs. Als s aanvaard werd door een pad zonder X, wordt s nog steeds aanvaard. Als het pad X bevat zijn er toestanden A en B zodat AX^nB een opeenvolging van toestanden is. De reguliere expressie op bogen AX, XX en XB zijn E_1, E_2 en E_3 en bijgevolg kost van A naar B gaan langs X een stuk string van de vorm $E_1(E_2)^*E_3$. Deze reguliere expressie staat ook in de boog AB van de nieuwe *GNFA*. □

[8] De toestand X moet zicht tussen de toestanden A en B bevinden.

String s wordt nog steeds niet aanvaard

Bewijs. Als s aanvaard wordt door de gereduceerde *GNFA* dan bevat het acceptatiepad uiteraard alleen toestanden verschillend van X. Op een boog AB staat de reguliere expressie $E_4|E_1(E_2)^*E_3$. Die gebruiken betekend een stukje string uitgeven dat voldoet aan E_4 of aan $E_1(E_2)^*E_3$. Dus in de originele *GNFA* komt dit overeen met ofwel de boog AB volgen ofwel AX, XX (een aantal keer) en XB.
Dus als de string aanvaard wordt door de gereduceerde *GNFA* wordt hij ook aanvaard door de originele *GNFA*. □

Kan dit ook met een PDA?

Dit is mogelijk aangezien we bij gewone *FSM's* labels samen nemen in een reguliere expressie. Een *PDA* kan dit ook doen, aangezien we meerdere karakters in één enkele keer op de stapel kunnen schrijven (of verwijderen).

Vraag 4. Geef een precieze formulering van het pompend lemma voor reguliere talen en een bewijs ervan. Geef voorbeelden waarbij je laat zien hoe je dat lemma gebruikt om te bewijzen dat een gegeven taal regulier is en om te bewijzen dat een gegeven taal niet regulier is.

We beginnen met dit alles intuïtief te bekijken:
Stel je hebt een reguliere taal met oneindig veel strings. Er bestaat een *DFA* voor L met $N = \#Q$ toestanden. Neem nu een string in L die langer is dan N en volg het pad van starttoestand naar eindtoestand. Vermits er maar N toestanden zijn en je string meer dan N karakters lang is en je bij elke overgang juist één symbool achterlaat, moet je op je weg naar de eindtoestand minstens één toestand S twee of meer keer tegenkomen. Je hebt dus ergens een kring gemaakt. In deze string werd een substring van de oorspronkelijke string gebruikt.
We definiëren nu het pompend lemma voor reguliere talen.

Definitie 4 (Pompend lemma voor reguliere talen)**.** *Voor een reguliere taal L bestaat 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:*

1. $\forall i \geq 0 : xy^i z \in L$
2. $|y| > 0$
3. $|xy| \leq d$

Bewijs. Neem een *DFA* die L bepaalt. Neem $d = \#Q + 1$. Neem ook een willekeurige string $s = a_1 a_2 \dots a_n$ met $n \geq d$. Beschouw nu de accepterende sequentie van toestanden $q_s = q_1, q_2, \dots, q_f$ voor s; die heeft een lengte strikt groter dan d, dus zijn er bij de eerste d zeker twee toestanden gelijk (omdat er maar $d - 1$ toestanden zijn).

Stel dat q_i en q_j gelijk zijn met $i < j \leq d$, dan nemen we $x = a_1 a_2 \dots a_i$ en $y = a_{i+1} \dots a_j$ en z de rest van de string. Alles volgt nu direct. □

Met het pompend lemma kan je *niet* bewijzen dat een taal regulier is. De stelling zegt immers niet dat als het pompend lemma geldt de taal sowieso regulier is. Er kunnen nog andere condities van een reguliere taal niet voldaan zijn. Je kan echter wel aantonen dat een taal niet-regulier is door een contradictie te bewijzen met de stelling hierboven.

Een soortgelijk bewijs heeft steeds de volgende structuur: kies een willekeurig getal als pomplengte d, bestaat er een string langer dan d waarvoor *elke* opdeling pompen verhinderd dan is deze taal nier regulier. Hier volgt een voorbeeld.

Bewijs. Stel dat er voor $L = \{a^n b^n | n \in \mathbb{N}\}$ een pomplengte d bestaat. Beschouw dan de string $s = a^d b^d$. Neem een willekeurige opdeling van $s = xyz$ met $|y| > 0$ dan zijn er drie mogelijkheden:

1. y bevat alleen a's: dan bevat $xyyz$ meer a's dan b's en zit deze string dus niet in L.

2. y is van de vorm $a^i b^j$ met $i \neq 0, j \neq 0$: dan bevat $xyyz$ niet alle a's voor de b's, en zit de string dus niet in L.

3. y bevat alleen b's: dan bevat xz meer a's dan b's en zit de string dus niet in L.

L kan dus niet regulier zijn. □

Vraag 5. Geef de definitie van een Myhil-Nerode relatie over een taal L, of zoals we noteren een *MN(L)*-relatie.
Bewijs vervolgens dat een *MN(L)*-relatie bestaat als en slechts als L regulier is. Bestaat er voor een taal L soms meer dan één *MN(L)*-relatie? Hoe zit het met *MN(L)*-relaties bij *PDA's*?

Definitie

Definitie 5 (Myhill-Nerode Relatie)**.** *Wanneer een equivalentierelatie $\sim_{DFA}$ voldoet aan de volgende voorwaarden:*

1. $\forall x, y \in \Sigma^*, a \in \Sigma : x \sim_{DFA} y \rightarrow xa \sim_{DFA} ya$ *(m.a.w. rechts congruent)*
2. $\sim_{DFA}$ *verfijnt* $\sim_L$ *(m.a.w.* $x \sim_{DFA} y \rightarrow x \sim_L y$*)*
3. $\sim_{DFA}$ *heeft een eindige index (m.a.w. het aantal equivalentieklassen van* $\sim_{DFA}$ *is eindig)*

Dan spreken we van een Myhill-Nerode relatie voor L (oftewel $MN(L)$) indien de DFA *de taal* L *bepaalt.*

Dit heeft zin aangezien de drie eigenschappen verwijzen naar L. Hierdoor kunnen we, vertrekkend van een *DFA* die L accepteert, een $MN(L)$ relatie contrueren op Σ^*.

L is regulier

We kunnen ook het omgekeerde doen en dus, vertrekkende uit $MN(L)$ een *DFA* contrueren zodat $L_{DFA} = L$.

Definitie 6. *Gegeven een taal* L *over* Σ *en een $MN(L)$-relatie* $\sim$ *op* Σ^**, dan definieert* $(Q, \Sigma, \delta, q_s, F)$ *een* DFA *die* L *bepaalt, waarbij*

1. $Q = \{x_\sim | x \in \Sigma^*\}$[a]
2. $q_s = \epsilon_\sim$
3. $F = \{x_\sim | x \in L\}$
4. $\delta(x_\sim, a) = (xa)_\sim$[b]

[a] Hier is dus $x_\sim$ de toestand waar alle strings x (of equivalent aan x) in terecht komen.
[b] We hebben $x_\sim$ in punt 1 gedefinieerd als $\delta^*(q_s, x)$ dus deze is gelijk aan $\delta^*(q_s, xa) = (xa)_\sim$.

Bewijs. Dat δ goed gedefinieerd is, kan je bewijzen door gebruik te maken van de rechtse congruentie van $\sim$. Verder zijn alle andere ingrediënten van de *DFA* duidelijk, in het bijzonder ook dat Q (en F) slechts een eindig aantal toestanden bevat. We moeten enkel nog bewijzen dat $L_{DFA} = L$. Stel dat x geaccepteerd wordt door de *DFA*.

$$x \in L_{DFA} \iff \delta(q_s, x) \in F$$

Uit de definitie van de *DFA* kunnen we zeggen dat $q_s = \epsilon_\sim$. Dus $\delta(\epsilon_\sim, x) \in F$. De toestand waaring string x komt kunnen we schrijven als $x_\sim$ (zie punt 1). We kunnen dus het volgende besluiten.

$$\delta(\epsilon_\sim, x) \in F \iff x_\sim \in F$$

Aangezien dit laatste geldt, kunnen we zeggen dat $x \in L$. Er bestaat dus een *DFA* voor L, dus L is regulier. □

Meerde MN(L) relaties

Dit is zeker mogelijk aangezien er ook oneindig veel (niet-isomorfe) *DFA's* bestaan die taal L aanvaarden.

Hoe zit het met een PDA?

Het principe blijft volledig hetzelfde. Echter was het bij een *DFA* genoeg om enkel ervoor te zorgen dat de toestand hetzelfde was voor twee strings x en y om deze als equivalent te beschouwen. Indien we specifiek naar een *PDA* kijken, dan moeten we een toestand niet individueel bekijken, maar in combinatie met de stack. Twee strings x en y zijn dan equivalent indien zowel de toestand waarin ze eindigen, als de stack, gelijk zijn.

Het probleem is echter dat de stack oneindig veel mogelijkheden heeft en dus heet de equivalentierelatie geen eindige index. Het is dus geen Myhill-Neroderelate, wel een equivalentierelatie.

Vraag 6. Reconstrueer de details van het verhaal van Myhill-Nerode. Leg uit alle stellingen uit: *DFA* bepaalt een Myhill-Neroderelatie. Elke Myhill-Neroderelatie bepaalt een *DFA*; twee isomorfe *DFA's* bepalen dezelfde Myhill-Nerode-relatie. Het supremum van twee Myhill-Neroderelaties is een Myhill-Neroderelatie. Argumenteer ook het verband tussen de minimale *DFA* voor een reguliere taal L en de grofste *MN(L)*-relatie. Kan je ook een Myhill-Neroderelatie construeren voor een *NFA*? Wat krijg je dan met minimalisatie?

Deze vraag is zeer groot. De eerste twee puntjes *Elke DFA bepaalt een Myhill-Neroderelatie* en *Elke Myhill-Neroderelatie bepaalt een DFA* kun je uitleggen met het antwoord op de vorige vraag.

Twee isomorfe DFA's

Bewijs. Stel dat we twee isomorfe *DFA's* hebben die de taal L bepalen (DFA_1 en DFA_2). Dit wil zeggen dan er een één op één relatie is tussen hun toestanden (volgt uit de definitie van isomorfisme). Stel dat x in DFA_1 eindigt in $q_{1,x}$ met $q_{1,x} = \delta_1(q_{1,s}, x) = q_1$ indien $q_{1,s}$ de starttoestand is van DFA_1. Uit de definitie van isomorfisme volgt dat $\delta_2(q_{2,s}, x) = q_{2,x} = q_2$ met $q_{2,s}$ de starttoestand van DFA_2 en dat q_2 equivalent is met q_1. Beide relaties zijn dus equivalent. □

Supremum van twee MN(L) relaties

Gegeven een reguliere taal L en twee $MN(L)$-relaties $\sim_1$ en $\sim_2$. We kunnen van die twee relaties het supremum beschouwen in de tralie van equivalentierelaties of partities. Het supremum $\sim_{sup}$ is de fijnste relatie die groffer is dan beide relaties en die dus bevat, namelijk:

Definitie 7. *$x \sim_{sup} y$ is de transitieve sluiting van $(x \sim_1 y)$ of $(x \sim_2 y)$.*

Definitie 8. *Het supremum van twee $MN(L)$-relaties is ook een $MN(L)$-relatie: als $\sim_1$ en $\sim_2$ $MN(L)$-relaties zijn, dan is het supremum $\sim_{sup}$ van $\sim_1$ en $\sim_2$ ook een $MN(L)$-relatie.*

Bewijs. Zie boek *p40* voor de formele notaties. Het komt er op neer om het supremum voluit te schrijven en dan de drie hoofdeigenschappen van een Myhill-Neroderelatie aan te tonen (door te termen terug op te splitsen en samen te nemen). □

Minimale DFA

Minimale *DFA* wijst op het minimaal aantal toestanden. Aangezien $x \sim y$ zullen ze beide dezelfde toestand eindigen, zijn er minder equivalentierelaties. Aangezien het universum verdeeld wordt in equivalentieklassen (één klasse per equivalentierelatie) zullen de klassen dus ook zo groot mogelijk zijn. Dit is de definitie van *grofste*.

MN(L) voor een NFA

Aangezien we een *DFA* kunnen opstellen van een *NFA* kunnen we ook een $MN(L)$-relatie opstellen. Dit zijn er echter heel wat, aangezien we verschillende *DFA's* kunnen opstellen die equivalent zijn aan de gegeven *NFA*.

Vraag 7. Geef alle stappen (stellingen & bewijzen) om aan te tonen dat alle minimale *DFA's* die dezelfde taal bepalen isomorf zijn.

De twee voorgaande vragen zijn belangrijk voor deze vraag. Dit antwoord baseert zich op voorgaande theoriën. Ik zou ze niet allemaal geven, maar als hij doorvraagt moet je ze wel kennen.

Bewijs. Neem twee minimale *DFA's* voor eent taal L. We zullen bewijzen dat ze isomorf zijn. Beiden hebben hetzelfde aantal toestanden N - anders was één van de twee niet minimaal. Neem nu het supremum van die twee *DFA's*: je krijgt een *DFA* met opnieuw N toestanden, want meer kan niet, maar ook minder niet.
In termen van de $MN(L)$-relaties door de drie betrokken *DFA's* geïnduceerd betekent dat dat die drie relaties identiek zijn. In termen van de drie *DFA's* betekent het dat ze alle drie isomorf zijn.
Pas nu weten we zeker dat de minimisatieprocedure eindigt met een uniek resultaat (op isomorfisme na). □

De karakterisatie van de $MN(L)$-relatie die hoort bij de minimale *DFA* is redelijk eenvoudig:

$$x \sim_{min} y \iff \forall s \in \Sigma^*(xs \in L \iff ys \in L)$$

In essentie zegt dit: als twee toestanden *f-gelijk* zijn, dan zijn ze identiek.

Vraag 8. Geef een precieze formulering van het pompend lemma voor context-vrije talen en een bewijs ervan. Geef voorbeelden waarbij je laat zien hoe je dat lemma gebruikt om te bewijzen dat een gegeven taal context-vrij is en om te bewijzen dat een gegeven taal niet context-vrij is.

Het pompend lemma voor contextvrije talen gaat als volgt:

Definitie 9 (Pompend lemma voor contextvrije talen)**.** *Voor een contextvrije taal* L *bestaat een getal P (de pomplengte) zodanig dat elke string s van* L *met lengte minstens p kan opgedeeld worden in vijf stukken u,v,x,y en z uit* Σ^* *zodanig dat* $s = uvxyz$ *en*

1. $\forall i \geq 0 : uv^i xy^i z \in L$
2. $|vy| > 0$
3. $|vxy| \leq p$

Dit is te bewijzen als volgt:

Bewijs. Eerst en vooral gaan we onze *CFG* in Chomsky normaalvorm zetten. Dit is een meer restrictieve vorm:

Definitie 10 (Chomsky Normaal Vorm)**.** *Een* CFG *heeft de Chomsky Normaal Vorm als elke regel één van de volgende vormen heeft:*

1. $A \to BC$
2. $A \to \alpha$
3. $S \to \epsilon$

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

Nu dat de *CFG* is omgezet heeft elke regel ofwel twee ofwel geen niet-terminalen aan de rechterkant. Stel nu n gelijk aan het aantal niet-eindsymbolen in de CFG.

Voor een bepaalde string s uit L bestaat er steeds een parse tree. Als je uit deze parse tree de onderste takken wegsnoeit hou je een volledige binaire boom over want de *CFG* staat in Chomsky Normaal Vorm. De hoogte van de boom is minstens gelijk aan $log_2(|s|)$. Het langste enkelvoudige pad van de wortel bevat dus minstens $log_2(|s|) + 1$ knopen en als we s lang genoeg kiezen, dan is $log_2(|s|) + 1$ groter dan n en bijgevolg moet er op dat langste pad minstens één niet-eindsymbool (neem X) herhaald worden.

Neem de laagste X (X_2) en zijn dichtste herhaling (X_1) op dat pad. X is zeker verschillen van het startsymbool. We kunnen nu een afleiding construeren van de vorm:

$$S \Rightarrow^* uX_2z \Rightarrow^* uvX_1yz \Rightarrow^* uvxyz$$

In deze afleiding zijn u, v, x, y, z strings uit Σ^* en bovendien zijn v en y niet tegelijkertijd leeg, want dan zou men uit X zichzelf kunnen afleiden en dat kan niet wegens de vorm van de grammatica.

Vermits bovenstaande afleiding geldig is, zijn

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

en

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

dat ook.

We hebben nu de eerste twee puntjes van de de stelling al bewezen indien er strings gekozen worden die langer zijn dan $2^{(n-1)}$. Dit is onze pomplengte p. vxy wordt afgeleid vanuit X met een parse tree die kleiner is dan n. De parse tree heeft dus hoogstens 2^{n-1} bladeren en die corresponderen juist met vxy. Hiermee is het derde puntje ook bewezen.

□

Vraag 9. Geef het algoritme om een *CFG* naar zijn *PDA* om te zetten en bewijs de correctheid hiervan. Volgens een constructie in de cursus worden meerdere elementen per keer gepushed, waarom mag dit?

Algoritme

We stellen de *PDA* op door middel van drie toestanden. De starttoestand q_s, de eindtoestand q_f en een hulptoestand x. We trekken slechts één boog van q_s naar x: die kijkt niet naar de string of stapel, en zetten een marker \$ op de stapel, samen met het beginsymbool (meestal E). Er is ook slechts één boog van x naar q_f: die consumeert niks van de string en haalt de marker \$ van de stapel. De andere bogen gaan van x naar x, de labels corresponderen met:

1. de symbolen uit het invoeralfabet: voor elke $\alpha \in \Sigma$, is er een boog met label $\alpha, \alpha \to \epsilon$; die bogen beteken dus: als de top van de stapel gelijk is aan het eerste symbool van de string, consumeer dan beide.

2. de regels van de grammatica: voor elke regel $X \to \gamma$ is er een label $\epsilon, X \to \gamma$; die bogen betekenen dus: als de top van de stapel een niet-eindsymbool X is, vervang het door de rechterkant γ van een regel in de grammatica waarvan X de linkerkant is; γ is een rij eind- en niet-eindsymbolen.

Correctheid

Het algoritme is steeds correct, aangezien het hetzelfde werkt dan we intuitief zouden doen. Het vult steeds een nieuwe expressie in en verwijderd eventuele atomen.[9]

Meerdere elementen pushen

Het is mogelijk om meerdere elementen in één stap pushen, aangezien dit overeenkomt met meerdere $\epsilon, \epsilon \to \alpha$ stappen.

[9] Ik heb geen flauw idee, send me the answer.

Vraag 10. Geef de definitie van een *PDA*. Leg de constructie uit van een *PDA* voor een gegeven contextvrije taal. Beargumenteer de correctheid. Is die *PDA* deterministisch?
Wat is deterministisch precies?
Wanneer wordt een string door een *PDA* aanvaardt? En door een *CFG*?
Leg uit ambiguiteit van de *CFG*.
Leidt ambiguiteit altijd tot een niet-deterministische *PDA*? En vice-versa?

Definitie

Definitie 11. *Een push-down automaat is een 6-tal* $(Q, \Sigma, \Gamma, \delta, q_s, F)$ *waarbij*

- Q *is een eindige verzameling toestanden*
- σ *is een eindig inputalfabet*
- Γ *is een eindig stapelalfabet*
- δ *is een overgangsfunctie met signatuur* $Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$
- q_s *is de starttoestand*
- $F \subseteq Q$ *is een verzameling eindtoestanden*

Constructie

Zie antwoord op een van de voorgaande vragen.

Determinisme

Standaard is een *PDA* niet deterministisch aangezien het mogelijk is dat er meerdere mogelijkheden zijn. Zo kan er ook een $\epsilon, \epsilon \to \epsilon$ boog bestaan. Een string wordt aanvaard indien deze eindigt in een aanvaardbare eindtoestand.

Ambiguiteit

Ambiguiteit wil zeggen dan de *CFG* ambigu is, m.a.w. het is mogelijk een keuze te maken om na te gaan of een string wordt geaccepteerd of niet. Aangezien deze keuze ook moet reflecteren in de corresponderende *PDA*, is de *PDA* dus ook niet deterministisch.

2 Talen en Berekenbaarheid

2.1 Inleiding

In dit hoofdstuk, Talen en Berekenbaarheid[10], zal er dieper worden ingegaan op Turingmachines en de werking ervan. Na het instuderen van dit hoofdstuk is het best om onderstaande vragen op te lossen. Het zijn niet zo veel vragen, maar ze bevatten steeds een redelijk groot deel van de leerstof (vaak ook verschillende stukken gecombineerd). De vragen die volgen zijn alle vragen die al eens gesteld zijn. Deze komen echter elk jaar terug. Concreet wil dit zeggen dat als al deze vragen gekend zijn een goed resultaat verwacht kan worden.

Dit wil echter niet zeggen dat het onmogelijk is om een andere vraag te krijgen. In de volgende sectie staat een kleine opsomming van delen uit het hoofdstuk die (tot nu toe) niet aan bod gekomen zijn tijdens de examens[11]. Er bestaat echter een kleine kans dat hij zijn vragen veranderd, wat wil zeggen dat ook deze delen gekend moeten zijn. In het algemeen zijn de vragen uit dit document echter goed genoeg.

In een andere repo kan je ook een implementatie van een Turingmachine terugvinden (geschreven in Haskell). De link vind je in de readme van deze repo. Dit kan een dieper inzicht geven op hoe zo een machine werkt.

2.2 Extra lectuur

De volgende secties dit hoofdstuk zijn tot nu toe nog nooit aan bod gekomen op het examen, maar moeten wel gekend zijn. De pagina's komen overeen met de publicatie van 19 november 2013[12].

1. Basiswerking van een Turingmachine - *p82-p86*
2. Er bestaat een niet herkenbare taal - *p87*
3. Universele Turingmachines - *p95*
4. Verband met reguliere talen - *p100*
5. $Regular_{TM}$ en EQ_{TM} - *p105*
6. Aftelbaar - *p110*
7. The Post Correspondence Problem - *p113-p116*
8. Recursieve functies - *p122-p125*
9. De bezige bever - *p126-p127*

[10] Dit is hoofdstuk 3 in de cursus (versie 2013).
[11] Het gaat hier enkel over het mondeling examen, niet over testen.
[12] De meeste recente versie in 2016.

Vraag 11. Bewijs dat A_{TM} niet beslisbaar is en steun daarbij niet op de stelling van *Rice*. Zou het helpen als het toegelaten was op de stelling van *Rice* te steunen? Is A_{TM} herkenbaar? Co-herkenbaar?

We hebben het hier over het acceptatieprobleem voor Turinmachines. We noemen de taal A_{TM} met de volgende formele definitie. Informeel is elk element een tuple met het eerste element een geëncodeerde Turingmachine die de string s, het tweede element, accepteerd.

$$A_{TM} = \{< M, s > \mid M \textit{ is een Turingmachine en } s \in L_M\}$$

A_{TM} is niet beslisbaar

Bewijs. Stel er bestaat een beslisser B voor A_{TM}. De werking van B kan op de volgende manier formeel gedefinieerd worden.

$$B(< M, s >) \textit{ is accept als } M \textit{ } s \textit{ accepteert en anders reject}$$

B weigert dus indien M de input s weigert of wanneer M in een oneindige lus zit. We construeren nu een contradictie machine C met de eigenschap om telkens het tegenovergestelde te accepteren (of te weigeren) van B. C neemt daarbij enkel Machine M als input en accept indien M de string representatie van zichzelf accepteert. We kunnen dit op de volgende manier formeel schrijven.

$$\forall \textit{ Turingmachine } M : C(< M >) = \neg(B(< M, M >))$$

Daarbij is $\neg accept = reject$ en $\neg reject = accept$. Neem nu voor M hierboven C zelf en vul deze in in C en B zelf. De volgende bewering komt tot stand.

$$C(< C >) = \neg(B(< C, C >))$$

We zien dat C zichzelf test. Indien C zichzelf accepteert, dan is $\neg B(C, C) = C(C) = accept$. Aangezien C C accepteert, moet $B(C, C) = accept$ en dus $\neg B(C, C) = reject$. Contradictie.

Conclusie: C kan niet bestaan. Indien B bestaat kan C wel bestaan, dus B bestaat ook niet. A_{TM} is dus niet beslisbaar. □

De stelling van Rice

Definitie 12 (Stelling van Rice)**.** *Voor elke niet-triviale, taal-invariante eigenschap P van Turingmachines geldt dat Pos_P (en ook Neg_P) niet beslisbaar is.*

Met deze stelling zouden we het hele bewijs kunnen inkorten. Door een niet-triviale, taal-invariante eigenschap P te vinden van alle Turingmachines die s herkennen, hebben we meteen aangetoond dat de taal in kwestie, A_{TM} niet beslisbaar is.[13] Deze taal komt dan overeen met Pos_P. Een eigenschap van $< M, s >$ is bijvoorbeeld dat M s accepteerd. $Pos_p = A_{TM}$ en dus is A_{TM} niet beslisbaar.

[13] Voor meer informatie over het bewijs, zie het laatste hoofdstuk.

A_{TM} is herkenbaar

Bewijs. De herkenner A voor A_{TM} laat, met input $< M, s >$, M lopen op s. Indien deze M s accepteerd, dan accepteerd A zijn input. Indien deze de input *reject*, of gewoon niet stopt, dan zal A deze ook niet accepteren. A_{TM} is dus herkenbaar (ook hier zien we dat A_{TM} niet beslisbaar is omdat deze kan blijven lopen). □

A_{TM} is niet co-herkenbaar

Bewijs. A_{TM} kan echter niet co-herkenbaar zijn. We bewijzen dit met contradictie. Indien A_{TM} co-herkenbaar is, is deze dus herkenbaar en co-herkenbaar (zie vorig bewijs). Wanneer een taal deze beide eigenschappen bezit, is deze beslisbaar. Dit is een contradictie met het eerste bewijs. □

Vraag 12. Bespreek de twee noties ($A \leq_m B$ en $A \leq_T B$) van reduceerbaarheid, hun verband en op welke manier die noties kunnen gebruikt worden om aan te tonen dat een taal (on)beslisbaar/herkenbaar is.

Om over te gaan naar de definitie van de reductie van talen, kunnen we best eerst de definifie van Turing-berekenbaar erbij halen (indien we dit niet doen, kunnen we zeker zijn van deze bijvraag).

Definitie 13 (Turing-berekenbare functie)**.** *Een functie f heet Turing berekenbaar indien er een Turingmachine bestaat die bij input s uiteindelijk stopt met $f(s)$ op de band.*

Veel-één reductie ($\leq_m$)

Definitie 14 (Reductie van talen)**.** *We zeggen dat taal L_1 (over Σ_1) naar taal L_2 (over Σ_2) kan gereduceerd worden indien er een afbeelding f met signatuur $\Sigma_1^* \longrightarrow \Sigma_2^*$ bestaat zodanig dat $f(L_1) \subseteq L_2$ en $f(\overline{L_1}) \subseteq \overline{L_2}$, en zodanig dat f Turing-berekenbaar is. We noteren dat door $L_1 \leq_m L_2$.*

Tot hiertoe is het al duidelijk wat $L_1 \leq_m L_2$ wil zeggen. Het is nu nog belangrijk om het verband met herkenbaarheid en beslisbaarheid aan te tonen. Dit doen we aan de hand van verschillende, specifiekere stellingen apart te bewijzen. We kunnen achteraf dan de vorige bewijzen gebruiken om ze simpeler te maken.

Definitie 15. *Als $L_1 \leq_m L_2$ en L_2 is beslisbaar, dan is L_1 beslisbaar.*

Bewijs. Het is belangrijk te weten dat de functie f, die elementen uit L_1 omzet naar element uit L_2, Turing-berekenbaar is. Concreet wil dit zeggen dat we de mogelijkheid hebben om een turingmachine op te stellen met als in put $s_1 \in L_1$ en als output $f(s_1) \in L_2$.
Neem nu dat L_2 beslisbaar is, met zijn beslisser B. We construeren nu een machine F die elementen uit L_1 omzet (via f) naar elementen uit L_2, waarna we de beslisser B laten beslissen. Hupsa, de combinatie van F en B is de beslisser van L_1 en ook deze taal is dus beslisbaar. □

Definitie 16. *Als $L_1 \leq_m L_2$ en L_2 is herkenbaar, dan is L_1 herkenbaar.*

Bewijs. Dit bewijs werkt hetzelfde als het voorgaande, om na te gaan dat wanneer L_2 beslisbaar is, dat dan ook L_1 beslisbaar is. Hier moeten we enkel de beslisser B vervangen door een herkenner H. □

Definitie 17. *Als $L_1 \leq_m L_2$ en L_1 is niet-herkenbaar, dan is L_2 niet-herkenbaar.*

Bewijs. Stel L_1 is niet-herkenbaar en L_2 wel. We hebben zonet bewezen dat als L_2 herkenbaar is, ook L_1 herkenbaar moet zijn. Contradictie. □

Definitie 18. *Als $L_1 \leq_m L_2$ en L_1 is niet-beslisbaar, dan is L_2 niet-beslisbaar.*

Bewijs. Stel L_1 is niet-beslisbaar en L_2 wel. We hebben zonet bewezen dat als L_2 beslisbaar is, ook L_1 beslisbaar moet zijn. Contradictie. □

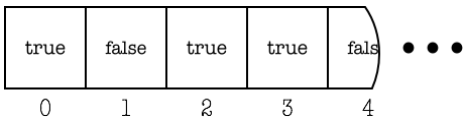
Orakels en hiërarchie van beslisbaarheid ($\leq_T$)

Zoals we weten is een Turingmachine een handig hulpmiddel om te bepalen of strings tot een taal horen of niet. We kunnen de machines gebruiken om talen te herkennen, of specifieker, te beslissen. Niet alle talen kunnen echter beslist worden door zo een Turingmachine. Zo is het bijvoorbeeld onmogelijk om een Turingmachine op te stellen die de taal A_{TM} beslist.

Dit zorgt er voor dat we op zoek moeten gaan naar een betere machine die naast het beslissen van de al besliste talen, ook andere talen kan beslissen. De nieuwe machine moet dus krachtiger zijn. Het resultaat is een orakelmachine.

Een orakelmachine is een uitbreiding op de Turingmachine, die een orakel bevat. Je kan een orakel bekijken als een black box waar de Turingmachine vragen een kan stellen. In theorie is een orakel eigenlijk een soort bitmap, of anders gezegd een rij van booleans. Stel we ordenen alle strings volgens de lexicografische orde met kortere strings eerst. Elke string op index i komt nu overeen met een booleaanse waarde in de bitmap, die ook gealloceerd is op locatie i. Indien de string op een bepaalde locatie tot een gegeven taal L behoord, dan zal de overeenkomstige booleaanse waarde op *true* staan. Indien dit niet het geval is, blijft de waarde op $false$.

De werking van een orakelmachine is nu heel eenvoudig. Het krijgt als input een string s. De machine vraagt nu aan het orakel of de string tot een taal behoort. Het orakel is in staat om de string om te vormen naar de index in de rij volgens de lexicografische volgorde en raadpleegt de overeenkomstige booleaanse waarde in de bitmap. Is die waarde *true*, dan accepteert het orakel de string s. Indien de waarde $false$ is, dan wordt de string s geweigert. Een orakelmachine waarvan de bitmap een configuratie heeft voor een bepaalde taal L te beslissen, noemen we O^L.



Figuur 1: Simpele voorstelling van een orakel (bitmap).

Een orakel kan voor vele problemen gebruikt worden. Het halting probleem is hier echter maar één enkel voorbeeld van. Vaak wordt een orakel gebruikt om op een abstracte manier een antwoord te krijgen op een bepaalde vraag. Deze vraag kan zelfs onoplosbaar zijn.

Definitie 19 (Turingreduceerbaar)**.** *Een taal A is Turingreduceerbaar naar taal B, indien A beslisbaar is relatief t.o.v. B, t.t.z. er bestaat een orakelachine O^B die A beslist. De notatie is $A \leq_T B$.*

Dit is inderdaad zeer gelijkend op het eerste deel van deze vraag. In plaats van een beslisser voor B te hebben, die A ook beslist, gebruiken we nu een orakelmachine. Deze machine is dan een theoretisch hulpmiddel dat we kunnen gebruiken om onze kennis toe te passen op meerdere talen. Deze kunnen we echter in realiteit niet implementeren zoals we net beschreven hebben.

Definitie 20. *Indien $A \leq_T B$ en B is beslisbaar, dan is A beslisbaar.*

Bewijs. De definitie zegt ons dat $A \leq_T B$ enkel geldt indien we een orakelmachine O^B hebben dat B én A beslist. Dit is dus volledig afleidbaar van de definitie. Of anders: stel dat B beslisbaar is en A niet. Dan hebben we een orakel O^B dat (theoretisch) B beslist, maar niet A (want deze is niet beslisbaar). Dit is meteen een contradictie met de definitie. □

Definitie 21. *Indien $A \leq_m B$ dan is ook $A \leq_T B$. M.a.w. $\leq_m$ is fijner dan $\leq_T$.*

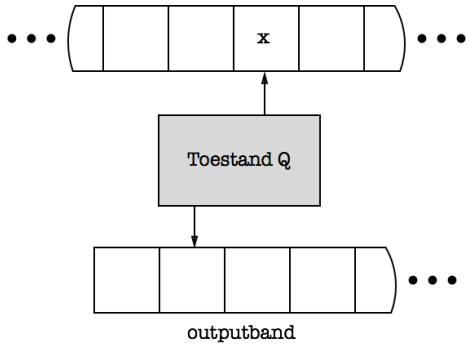
Dit is vanzelfsprekend indien we beseffen dat het orakel een theoretische uitbreiding is op de Turingmachine. We gebruiken de Turingmachines om talen te herkennen of the beslissen. Het is mogelijk zo een machine te implementeren in een taal naar keuze. Er is echter een grens op het aantal talen dat we kunnen beslissen, aangezien een aantal in een oneindige lus kunnen komen tijdens het beslissingsproces. Dit is een probleem dat we in de praktijk kunnen tegenkomen. Een theoretische oplossing daarvoor is de orakelmachine. We kunnen deze machine wel gebruiken om theoretisch verder te redeneren. Dit will zeggen dat het orakel alle talen beslist dit een Turingmachine kan besliseen, plus de talen die een turingmachine niet kan beslissen (oneindig lus). Hierdoor is $A \leq_m B$ fijner dan $A \leq_T B$.

Vraag 13. Definieer de enumeratormachine. Bewijs dat elke herkenbare taal kan geënumereerd worden en dat elke taal die door een enumerator wordt geënumereerd ook herkenbaar is. Kan elke beslisbare taal geënumereerd worden? Bespreek in deze context de uitspraak "de verzameling van Turing machines is een herkenbare taal".

De enumeratormachine is de machine zoals origineel voorgesteld door Alan Turing in 1936. Deze machine bezit deels de Turingmachine zoals we deze al kennen met enkele kleine aanpassingen. Zo heeft de machine ook een outputband, outputmarker en een enumeratortoestand q_e. De δ van de enumerator heeft als signatuur

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

Met Q de huidige toestang en Γ het te lezen symbool. Als output een nieuwe toestand Q en Γ, samen met $\{L, R, S\}$. Het verschil met de Turingmachine die we eerder hebben gezien is Γ_e. Dit symbool zal bij overgang geschreven wordt op de outputband. Na het schrijven van Γ_e verplaatst de outputmarker zich 1 plaats naar rechts. De output kan dan een string zijn, verschillende strings opgesplitst door een scheidingsteken of zelfs een oneindige string... Eender hoe, het heeft zin om te spreken over de verzameling (eindige) outputstrings door de enumerator geproduceerd of geënumereerd. Die verzameling is de taal door de enumerator bepaald of geënumereerd. De enumerator mag daarbij dezelfde string meer dan eens op de outputband zetten.



Figuur 2: Enumeratormachine.

Definitie 22. *De taal door een enumerator bepaald is herkenbaar en elke herkenbare taal wordt door een enumerator geënumereerd. Beide stellingen zullen in aparte bewijzen worden bewezen.*

Bewijs. **Elke taal door een enumerator bepaald is herkenbaar.** Neem de Turingmachine T voor de taal L, bepaald door een gegeven enumerator E. Laat T nu E als een subroutine gebruiken. T neemt een string s aan. Telkens E in q_e komt zal T de laatst geproduceerde outputstring lezen op de outputband van E. Indien deze gelijk is aan de inputstring s, dan accepteert T s. Indien

dit niet zo is, gaat de enumerator doorrrekenen[14]. □

Bewijs. **Elke herkenbare taal wordt geënumereerd door een enumerator.** Neem de Turingmachine T die de willekeurige taal L herkent. Het is voldoende een enumerator E te construeren die L enumereert[15]. Om E op te stellen, maken we gebruik van enkele hulpmachines.

1. Een machine T_{gen} die voor een gegeven getal n de eerste n strings uit Σ^* genereerd[16]. Deze T_{gen} zal de strings op de band van E plaatsen.
2. T_n zal op elk van de gegeven n strings, n stappen van T uitvoeren. Het zal dus het aantal stappen van T limiteren zodat deze niet in een oneindige lus kan gaan voor de strings die niet worden herkend. Waarom doen we dit? We willen eigenlijk alle strings overlopen[17] en daar de aanvaardbare strings ($s \in L$) uit filteren. Deze behoren tot de herkenbare taal L. Hierdoor moeten we ook de strings nakijken die niet tot L behoren. Deze kunnen er echter voor zorgen dat de T in een oneindige lus gaat. Daarom limiteren we het aantal stappen zodat ook E niet oneindig moet blijven wachten. Indien T de string aanvaardt, plaatst E de string op de outputband. Indien T deze niet aanvaard of niet geëindigd is in $\leq n$ stappen, dan zal E de string niet schrijven.
3. T_{driver} zal een opeenvolging van getallen n genereren om de voorgaande machines op te stellen en T_{gen} en T_n oproepen.

Resultaat: Alle $s \in L$ zullen op de outputband worden gezet, maar toch zal de uitvoering van de oneindige lus gestopt worden zodat de enumerator zeker eindigt. Hiermee is het halting probleem voorkomen. Well done. □

Ten slotte moeten we vermelden dat elke taal die beslisbaar is kan geënumereerd worden, aangezien elke beslisbare taal ook herkenbaar is. Uit het laatste bewijs zou ook duidelijk moeten zijn dat A_{TM} een herkenbare taal is. We voeren het bewijs gewoon opnieuw uit, maar met $L = A_{TM}$. Dus in plaats van T n stappen te laten lopen over een inputstring, laten we nu M n stappen lopen over s (uit $< M, s > \in A_{TM}$).

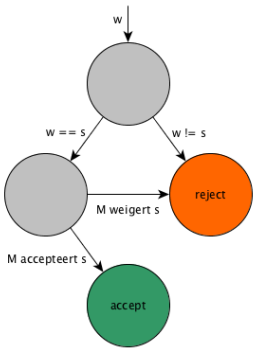
[14] De enumerator kan dus oneindig blijven doorrekenen waardoor de taal door een enumerator bepaald herkenbaar en niet beslisbaar is.
[15] Indien dit lukt is dit mogelijk voor elke L, aangezien L willekeurig is.
[16] Namelijk $s_1, s_2, \ldots, s_n$.
[17] Die mogelijk zijn op het alfabet Σ.

Vraag 14. Geef het bewijs van de stelling: E_{TM} is niet beslisbaar. Doe dit zonder de stelling van Rice te gebruiken. Bespreek de uitspraken E_{TM} is herkenbaar en E_{TM} is co-herkenbaar. Zijn er ook alternatieve bewijzen? Hoe zit het met E_{CFG}?

E_{TM} is niet beslisbaar

Definitie 23 (E_{TM})**.** *$E_{TM} = \{< M > \mid M$ is een TM, en $L_M = \phi\}$ is niet beslisbaar: het is niet beslisbaar of een Turingmachine geen enkele input accepteert.*

Bewijs. In dit bewijs gaan we gebruik maken van een hulpmachine M_s. Het is belangrijk eerst te snappen wat deze machine doet. s is hier een string die gehardcoded is. Voor elke input w die niet gelijk is aan s, zal de machine M_s de input weigeren. Indien $w = s$, dan zal M lopen op w (of s). Indien M de input s accepteert zal ook M_s accepteren. Onderstaande afbeelding is een schematische voorstelling van de hulpmachine M_s.



Figuur 3: Concept van M_s

Nu kunnen we verder met het bewijs. We proberen de stelling aan te tonen met een bewijs uit het ongerijmde. Stel dat E_{TM} beslisbaar is en een beslisser E heeft. We gaan nu een beslisser B voor A_{TM} opstellen met behulp van E[18]. We kunnen twee punten inzien.

1. We stellen eerste onze hulpmachine M_s op. We laten nu E lopen over $< M_s >$. Stel dat E nu $< M_s >$ accepteert. M_s accepteert dus geen strings (zie definitie E_{TM}). Merk op dat in dit geval $B < M, s >$ niet accepteert. M_s bepaalt de lege taal, dus M bepaalt s niet.

2. In het andere geval kan het zijn dat $E < M_s >$ verwerpt. Dan bepaalt M_s wel een taal (die niet leeg is) en zal dus B zijn input wel accepteren.

Uit bovenstaande punten kunnen we concluderen dat E de tegenovergestelde uitkomst heeft van B. B bestaat echter niet, aangezien A_{TM} niet beslisbaar is. E kan dus ook geen beslisser zijn en E_{TM} is dus niet beslisbaar. Contradictie. □

[18] De beslisser B kan niet bestaan, want A_{TM} is onbeslisbaar. Dit wordt de contradictie.

E_{TM} is niet herkenbaar maar wel co-herkenbaar

Bewijs. Het is makkelijk aan te tonen dat door eerst de co-herkenbaarheid van E_{TM} te onderzoeken. Het complement van E_{TM} bestaat uit alle Turingmachines die een niet lege taal bepalen. Neem nu Turingmachine $M \in \overline{E_{TM}}$ en dus $L_M \neq \phi$. We maken een hulpmachine die voor elk van die M alle strings s in de lexicografische volgorde M laat lopen over s. Vanaf M accepteert, accepteert de hulpmachine. Indien dit niet is, gaat hij verder met de volgende string s. Dit proces kan niet oneindig doorgaan, tenzij geen enkele string s geaccepteerd wordt door M. $\overline{E_{TM}}$ is dus herkenbaar en dus is E_{TM} co-herkenbaar.

E_{TM} is dus niet herkenbaar aangezien het co-herkenbaar is. Indien het ook herkenbaar is, zou het ook beslisbaar zijn. Dit is een contradictie met het vorige bewijs. □

Alternatieve bewijzen

Bewijs. Een alternatief bewijs zou kunnen zijn met behulp van de stelling van Rice. Een taal-invariante, niet-triviale eigenschap van E_{TM} is dat $L_M = \phi$ voor elke M in E_{TM} die L_M bepaalt. Volgens de stelling van Rice is E_{TM} dus onbeslisbaar. □

Bespreek E_{CFG}

> **Definitie 24.** *$E_{CFG} = \{< G > \mid G$ is een CFG, en $L_G = \phi\}$ is beslisbaar: emptyness van een CFL is beslisbaar.*

Bewijs. We beschrijven formeel een algoritme dat G transformeert naar een vorm waarin de beslissing gemakkelijk is.

1. Indien er een regel $A \to \alpha$ in zit en α bestaat alleen uit eindsymbolen (mag dus ook ϵ zijn), dan
   (a) Verwijder alle regels waar A aan de linkerkant staat
   (b) Vervang in elke regel waar A rechts voorkomt, de voorkomens van A voor α
2. Blijf dit doen totdat ofwel
   (a) Het startsymbool verwijderd is: reject, want het startsymbool kan een string afleiden.
   (b) Er geen regels zijn van de benodigde vorm: accept, want de taal is leeg.

□

Vraag 15. Leg de stelling van Rice uit en geef het bewijs. Geef minstens één eigenschap van Turingmachines die niet voldoet aan de voorwaarde voor de stelling van Rice en toon aan dat die eigenschap geen aanleiding geeft tot een niet-beslisbare taal. Karakteriseer volledig alle eigenschappen van Turingmachines die aan de stelling van Rice voldoen m.b.v. $IsIn_{TM,S}$.

De stelling van Rice

Vooraleer we verder gaan met de stelling van Rice is het verstandig om sommige termen te verklaren. Niet-triviale en taal-invariante eigenschappen zijn een belangrijk onderdeel van de stelling. In onderstaande verklaringen duidt Pos_P op de verzameling van machines die in het bezit zijn van een eigenschap P en Neg_P de verzameling van machines zonder eigenschap P.

Definitie 25 (Niet-triviale eigenschap)**.** *Een eigenschap P van Turingmachines heet niet-triviaal indien $Pos_p \neq \emptyset$ en ook $Neg_p \neq \emptyset$. Er bestaan dus Turingmachines die deze eigenschap P bezitten, maar ook machines die deze niet bezitten.*

Definitie 26 (Taal-invariante eigenschap)**.** *De eigenschap P heet taal-invariant indien alle machines die dezelfde taal bepalen hebben ofwel allemaal P, ofwel heeft geen enkele ervan P.*

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

Met deze twee definities in het achterhoofd, kunnen we overgaan naar de formele definitie van de stelling van Rice, met het bewijs als gevolg.

Definitie 27 (Stelling van Rice)**.** *Voor elke niet-triviale, taal-invariante eigenschap P van Turingmachines geldt dat Pos_P (en ook Neg_P) niet beslisbaar is.*

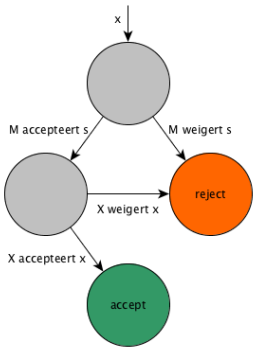
Bewijs. We bewijzen dit met behulp van een contradictie. Neem de Turingmachine $M_\emptyset$ die de lege taal beslist. Laten we er nu van uit gaan dat deze machine een bepaalde eigenschap P heeft. De stelling zegt ons dat de eigenschap P niet-triviaal en taal-invariant is. Uit de definitie kunnen we dan afleiden dat $Pos_P \neq \emptyset$ (en ook $Neg_P \neq \emptyset$). Aangezien deze verzameling niet leeg is, moet er een Turingmachine X bestaan met deze eigenschap P. Laat ons zeggen dat deze Turingmachine de taal L_X beslist.

We gaan nu proberen een contradictie te bekomen door aan te nemen dat de stelling niet waar is. We nemen dus aan dat Pos_P (en dus ook Neg_P) beslisbaar is. We gaan nu een beslisser B proberen op te stellen voor Pos_P die deze beslist[19]. Om B te maken, gaan we eerst een hulpmachine $H_{M,s}$ opstellen.

Deze hulpmachine $H_{M,s}$ heeft een Turingmachine M en een string s in

[19]Later zullen we deze B gebruiken om een beslisser A te maken voor de taal met Turingmachines A_{TM}

zich. Deze staan vast voor de machine en kunnen dus niet veranderen[20]. De input van deze machine is een string $x \in L_X$. Wanneer $H_{M,s}$ gestart wordt, zal deze eerst M laten lopen over s. Indien M s reject, zal $H_{M,x}$ altijd rejecten. Indien M s accept, dan zal $H_{M,s}$ overgaan naar fase 2. Hier zal de hulpmachine X over x laten lopen. Indien X x ook accept, dan zal de hulpmachine accepten. Indien X x reject, dan zal ook de hulpmachine rejecten.



Figuur 4: Concept van $H_{M,s}$

Er zijn nu twee mogelijkheden voor $H_{M,s}$. Indien M s accept, dan gaat $H_{M,s}$ altijd overgaan tot het testen van x in X. In dit geval beslist $H_{M,s}$ de volledige taal L_X. De andere optie is dat M s reject. In dat geval gaat de hulpmachine altijd rejecten en dus enkel de lege taal accepteren.
Laat nu de beslisser B los op $H_{M,s}$. Dit wil zeggen dat de beslisser accept of reject voor de gegeven M en s.
Stel dat we nu een beslisser A maken voor A_{TM}. In dat geval moeten we dus elke M en s in A_{TM} testen. We kunnen dus zeggen dat A accept indien B $H_{M,s}$ accept, anders reject. We bekomen dus de volgende conclusie.

$$
\begin{gathered}
A \text{ accepts } < M, s > \\
\Updownarrow \\
B \text{ accepts } H_{M,s} \\
\Updownarrow \\
H_{M,s} \text{ heeft eigenschap } P \\
\Updownarrow \\
H_{M,s} \text{ accepts } L_X \\
\Updownarrow \\
M \text{ accepts } s
\end{gathered}
$$

Conclusie: A is een beslisser voor A_{TM}, maar dit is onmogelijk aangezien A_{TM} niet beslisbaar is[21]. Hieruit kunnen we concluderen dat alle bovenstaand equivalenties onwaar zijn en dus is ook Pos_P niet beslisbaar. Contradictie. □

[20] Ze zijn als het ware gehardcoded.
[21] Zie vraag 1 van dit hoofdstuk voor het bewijs.

Voorbeeld

We nemen de Turingmachine TM als voorbeeld. We stellen ons de vraag over de volgende eigenschap. Zal de Turingmachine TM ooit zijn leeskop naar links verschuiven? We kunnen hier een beslisser voor opstellen.

Gegeven het tupel $< M, s >$ zal TM M loslaten op s voor maximaal $|s|$ stappen. Als de machine geen enkele keer zijn leeskop naar links heeft bewogen, dan moet de leeskop op de eerste # staan aan de rechterkant achter w op de leesband. Zoek nu de overgang δ in het state diagram van M die # als input heeft. We zijn enkel geïntereseerd in deze, aangezien voor de andere de leeskop nooit naar links is gegaan. Indien geen enkel van deze transacties naar links gaat, accept. Indien een van hun wel naar links gaat, reject.

Algemene notatie

De volgende notatie bepaalt de verzameling van turingmachines M die een taal L_M bepalen die behoren tot een gegeven verzameling van talen S.

$$IsIn_{TM,S} = \{< M, S > | L_M \in S\}$$

We kunnen deze notatie gebruiken om eigenschappen te karakteriseren. Een voorbeeld van een eigenschap van E_{TM} die voldoet aan de voorwaarden van de stelling van Rice, is dat de bepaalde taal leeg is. We kunnen dan de volgende notatie gebruiken om de eigenschap te karakteriseren.

$$E_{TM} = IsIn_{TM,\{\phi\}}$$

Vraag 16. Wat is een orakelmachine? Bespreek de uitspraak "de verzameling orakelmachines (over een gegeven orakel) is strikt krachtiger dan de verzameling van Turing machines". Leg hierbij ook uit wat men bedoelt met "krachtiger". Kan een verzameling orakelmachines (voor bepaald gegeven orakel) alle talen beslissen? Kan een orakelmachine ook A_{TM} of H_{TM} beslissen?

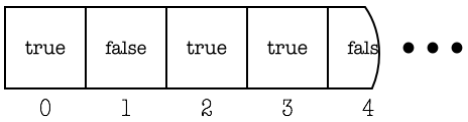
Orakelmachine

Zoals we weten is een Turingmachine een handig hulpmiddel om te bepalen of strings tot een taal horen of niet. We kunnen de machines gebruiken om talen te herkennen, of specifieker, te beslissen. Niet alle talen kunnen echter beslist worden door zo een Turingmachine. Zo is het bijvoorbeeld onmogelijk om een Turingmachine op te stellen die de taal A_{TM} beslist.

Dit zorgt er voor dat we op zoek moeten gaan naar een betere machine die naast het beslissen van de al besliste talen, ook andere talen kan beslissen. De nieuwe machine moet dus krachtiger zijn. Het resultaat is een orakelmachine.

Een orakelmachine is een uitbreiding op de Turingmachine, die een orakel bevat. Je kan een orakel bekijken als een black box waar de Turingmachine vragen een kan stellen. In theorie is een orakel eigenlijk een soort bitmap, of anders gezegd een rij van booleans. Stel we ordenen alle strings volgens de lexicografische orde met kortere strings eerst. Elke string op index i komt nu overeen met een booleaanse waarde in de bitmap, die ook gealloceerd is op locatie i. Indien de string op een bepaalde locatie tot een gegeven taal L behoord, dan zal de overeenkomstige booleaanse waarde op *true* staan. Indien dit niet het geval is, blijft de waarde op $false$.

De werking van een orakelmachine is nu heel eenvoudig. Het krijgt als input een string s. De machine vraagt nu aan het orakel of de string tot een taal behoort. Het orakel is in staat om de string om te vormen naar de index in de rij volgens de lexicografische volgorde en raadpleegt de overeenkomstige booleaanse waarde in de bitmap. Is die waarde *true*, dan accepteert het orakel de string s. Indien de waarde $false$ is, dan wordt de string s geweigert. Een orakelmachine waarvan de bitmap een configuratie heeft voor een bepaalde taal L te beslissen, noemen we O^L.



Figuur 5: Simpele voorstelling van een orakel (bitmap).

Een orakel kan voor vele problemen gebruikt worden. Het halting probleem is hier echter maar één enkel voorbeeld van. Vaak wordt een orakel gebruikt om op een abstracte manier een antwoord te krijgen op een bepaalde vraag. Deze vraag kan zelfs onoplosbaar zijn.

Krachtiger dan een Turingmachine

Zoals zonet vermeld, is een orakelmachine krachtiger dan een Turingmachine. Dit wil zeggen dat een orakelmachine sowieso meer talen kan beslissen dan een Turingmachine. Het kan namelijk alle talen beslissen die een Turingmachine kent vermeerderd met heel wat talen die anders in een oneindige lus zullen komen. A_{TM} is daar een voorbeeld van. Een orakelmachine is dus strikt krachtiger dan een Turingmachine.

Verzameling orakelmachines

Normaal wordt een orakelmachine gegeven en kunnen we deze vraag concreet beantwoorden. Dit is nu dus onmogelijk. De redenering gaat echter van de volgende vorm zijn.

Gegeven een orakel O^A dat een taal A beslist. Volgens de definitie van Turingreduceerbaar (zie onderaan) kunnen we concluderen dat elke B, waarvoor geldt dat $B \leq_T A$, kan beslist worden door de orakelmachine O^B. Het is dus niet mogelijk om alle talen te beslissen, enkel die die Turingreduceerbaar zijn.

> **Definitie 28** (Turingreduceerbaar)**.** *Een taal A is Turingreduceerbaar naar taal B, indien A beslisbaar is relatief t.o.v. B, t.t.z. er bestaat een orakelachine O^B die A beslist. De notatie is $A \leq_T B$.*

Het is misschien wel mogelijk om een orakel te ontwerpen dat dit wel kan. In plaats van een bitmap voor alle strings, maken we een bitmap voor alle talen. Een element bevat hier geen booleaanse waarde, maar een ander orakel voor de overeenkomstige taal. Theoretisch is dit volgens mij mogelijk, maar gaat misschien iets te ver voor op het examen.

Beslisbaarheid A_{TM} en H_{TM}

Dit is een bijvraag en hier gaan we dus niet dieper op in. De bewijzen van de onbeslisbaarheid worden hier dus niet gegeven. Deze staan ook al in andere vragen. Volgens mij kan hier heel beknopt op geantwoord worden. A_{TM} en H_{TM} zijn niet beslisbaar met standaard Turingmachines. Met een orakelmachine kunnen we uiteraard een corresponderende bitmap maken voor de talen en zijn ze dus wel beslisbaar.

Vraag 17. Geef informeel de definitie van een lineair begrensde automaat (LBA). Argumenteer dat het aanvaardingsprobleem voor LBA's (A_{LBA}) beslisbaar is. Geef de stappen in een bewijs dat E_{LBA} (de verzameling van LBA's die de lege taal bepalen) niet beslisbaar is.

Lineair begrensde automaat

Definitie 29 (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.*

De naam komt hier tot stand door de volgende equivalente definitie van een lineair begrense automaat. Deze definitie laat toe dat de LBA een stuk band gebruikt dat met een constante factor f groter mag zijn dan de input.

A_{LBA} is beslisbaar

Het acceptatieprobleem voor LBA is gedefinieerd als de taal

$$A_{LBA} = \{< M, s > | M \text{ is een } LBA \text{ en } s \in L_{LBA}\}$$

Bewijs. We kijken naar alle mogelijke configuraties van een LBA op een string van lengte n. Het aantal toestanden is q met het aantal elementen in het bandalfabet b. Het aantal mogelijke strings is dan 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[22]:

1. Bereken $k = qnb^n$.
2. Simuleer dan M op s met maximaal k stappen.
3. Indien M ondertussen accepteerde, accept.
4. Indien M ondertussen verwierp, reject.
5. Indien M nog niet stopte, betekent dat dat M in een lus zit en dus niet zal accepteren: reject.

□

E_{LBA} is niet beslisbaar

Definitie 30. $E_{LBA} = \{M | M \text{ is een } LBA \text{ die de lege taal bepaalt } \}$ *is niet beslisbaar.*

We laten eerst zien dat voor een gegeven Turingmachine M en string s we een LBA kunnen construeren die gegeven een eindige rij configuraties (van M) kan beslissen of die rij een accepterende computation history is voor s. Een rij configuraties kan gemakkelijk op een band geplaatst worden zoals in de figuur hieronder.

[22] Bij input $< M, s >$.

Figuur 6: $\delta(q_4, c) = (q_7, b, R)$

Wat moet de machine doen om na te gaan of een rij configuraties een accepterende computation history is voor s?

1. Nakijken of twee opeenvolgende configuraties verbonden zijn door de δ.
2. Nakijken of de eerste configuratie q_s bevat op de juiste plaats.
3. Nakijken of de laatste configuratie q_a bevat.

Zonder veel in detail te treden moet het duidelijk zijn dat hiervoor slechts een constante hoeveelheid extra bandruimte nodig is en dat die beslissing dus kan genomen worden door een LBA. We maken die LBA zo dat hij bij een accepterende computation history accepteert en anders reject. Nu kunnen we aan het bewijs zelf beginnen.

Bewijs. Stel dat we een beslisser E hebben voor E_{LBA}. We construeren een beslisser B voor A_{TM} als volgt. Bij input $< M, s >$ doet B:

1. Construeer de LBA $A_{M,s}$ die van input kan beslissen of een inputstring een accepterende computation history is voor M op input s.
2. Laat E los op $< A_{M,s} >$: als E aanvaardt, reject; anders accept.

$$
\begin{array}{c}
B \text{ beslist } A_{TM} \text{ want } B \text{ accepteert } < M, s > \\
\Updownarrow \\
E < A_{M,s} > \text{ reject} \\
\Updownarrow \\
A_{M,s} \text{ aanvaardt minstens één string} \\
\Updownarrow \\
\text{Er bestaat een accepterende computation history voor } M \text{ op } s
\end{array}
$$

Het laatste is equivalent met zeggen dat M s accepteert. Die B kan niet bestaan, dus ook E niet en E_{LBA} is onbeslisbaar. □

3 Herschrijfsystemen

3.1 Inleiding

Herschrijfsystemen definiëren we als een verzameling termen en een verzameling regels die ons toelaten om termen te herschrijven. In het grootste deel van dit hoofdstuk zullen we werken met λ-calculus. Hier zijn de termen λ-expressies en de herschrijfregels conversieregels. Deze calculus wordt ook gebruikt in Haskell, zoals gezien in het vak *Declaratieve Talen*.

Buiten λ-calculus zijn er nog verschillende soorten herschrijfsystemen. We gaan hier vooral in op λ-calculus omdat deze een van de meest basis voorbeelden is dat Turingcompleet is, t.t.z. het is mogelijk om een Turingmachine M te maken die met deze calculus kan werken.

Hieronder vind je nog een overzicht van de delen die niet aan bod komen in de examenvragen.

3.2 Extra lectuur

1. Inleiding - *p128-p135*
2. Programmeren in λ-calculus - *p147-p149*
3. Andere herschrijfsystemen - *p158*
4. Oefeningen - *p160*

Vraag 18. Formuleer en bespreek de stellingen van Church-Rosser. Geef daarbij hun belang i.v.m. het baseren van een programmeertaal op lambda-calculus. Geef de relatie met de programmeertaal Haskell. Hoeveel conversieregels ken je?

Church-Rosser

Definitie 31 (Church-Rosser I)**.** *Indien* $E_1 \overset{*}{\longleftrightarrow} E_2$*, dan bestaat er een expressie* E *zodanig dat* $E_1 \overset{*}{\longrightarrow} E$ *en* $E_2 \overset{*}{\longrightarrow} E$*.*

Indien het mogelijk is om via conversies E_1 om te vormen naar expressie E_2 (en vice versa, ze zijn dus equivalent), dan hebben deze expressies dezelfde normaalvorm op α-conversie na. Hieruit volgt het volgende.

Definitie 32 (Uniciteit van een normaalvorm)**.** *Geen expressie kan geconverteerd worden naar twee verschillende normaalvormen (t.t.z. twee normaalvormen die niet* α*-convertibel zijn in mekaar).*

Bewijs. Stel $E \overset{*}{\longleftrightarrow} E_1$ en $E \overset{*}{\longleftrightarrow} E_2$ waarbij E_1 en E_2 in normaalvorm staan, dan is $E_1 \overset{*}{\longleftrightarrow} E_2$ en dus bestaat er volgens *CRI* een expressie F zodat $E_1 \overset{*}{\longrightarrow} F$ en $E_2 \overset{*}{\longrightarrow} F$. Maar vermits E_1 en E_2 geen redexen bevatten, is $E_1 = F = E_2$ op α-conversie na. □

M.a.w. alle eindige reductierijen vanaf een expressie E eindigen in dezelfde normaalvorm. Vooraleer we verder gaan met *CRII* moeten we het volgende definiëren. De *normaalorde* bepaalt dat de meest linkse buitenste redex eerst moet gereduceerd worden.

Definitie 33 (Church-Rosser II)**.** *Indien* $E \overset{*}{\longrightarrow} N$*, met* N *in normaalvorm, dan bestaat er een reductierij in normaalorde van* E *naar* N*.*

Invloed op programmeertaal en Haskell

Het reduceren volgens een reductierij in normaalorde heeft zowel voor- als nadelen bij het gebruik in een programmeertaal. Wanneer de reductierij niet in normaalorde wordt toegepast, is het mogelijk dat een expressie E op een snellere manier kan gereduceerd worden naar een expressie N in normaalvorm.

Wanneer we wel belang hechten aan de reductierij in normaalorde (zoals bij Haskell), dan zijn we in staat om partiële functies te maken (aka currying).

Conversieregels

We hebben drie verschillende conversieregels gezien, namelijk β-, α- en η-conversie. α-conversie kan omschreven worden als naamverandering van een formele parameter. β-conversie is het toepassen van een functie en vice versa. η-reductie is de eliminatie van redundante abstracties.

Vraag 19. Geef de conversieregels voor lambda-calculus, bespreek de stellingen van Church-Rosser en leg adh daarvan uit hoe je een recursieve functie in lambda-calculus kunt maken. Geef ook een voorbeeld dat we niet besproken hebben in de cursus.

De regels en de stellingen van *Church-Rosser* zijn in de vorige vraag al aan bod gekomen. We bekijken nu enkel recursieve functies in meer detail.

Om te kunnen werken met recursieve functies moeten we goed begrijpen wat vastepuntstheorie is. De uitleg uit het boek vind je op *p154-p155*. Het is niet mogelijk deze in te korten aangezien deze uitleg al zeer compact is. Een ander voorbeeld dat we niet hebben besproken vind je hieronder terug. We gebruiken als voorbeeld 3^2.

$$
\begin{aligned}
\text{Pow } 3\ 2 &= (Y\ H)\ 3\ 2 \\
&= H\ (Y\ H)\ 3\ 2 \\
&= (\lambda f.\ (\lambda xy.\ \text{IF } (=\ Y\ 0)\ 1\ (*\ x\ (f\ x\ (-\ Y\ 1)))))(Y\ H)\ 3\ 2 \\
&= (\lambda xy.\ \text{IF } (=\ Y\ 0)\ 1\ (*\ x\ ((Y\ H)\ x\ (-\ Y\ 1))))\ 3\ 2 \\
&= *\ 3\ ((Y\ H)\ 3\ 1) \\
&= *\ 3\ (H\ (Y\ H)\ 3\ 1) \\
&= *\ 3\ ((\lambda f.\ (\lambda xy.\ \text{IF } (=\ Y\ 0)\ 1\ (*\ x\ (f\ x\ (-\ Y\ 1)))))(Y\ H)\ 3\ 1) \\
&= *\ 3\ ((\lambda xy.\ \text{IF } (=\ Y\ 0)\ 1\ (*\ x\ ((Y\ H)\ x\ (-\ Y\ 1))))\ 3\ 1) \\
&= *\ 3\ (*\ 3\ ((Y\ H)\ 3\ 0)) \\
&= *\ 3\ (*\ 3\ (H\ (Y\ H)\ 3\ 0)) \\
&= *\ 3\ (*\ 3\ ((\lambda f\ (\lambda xy.\ \text{IF } (=\ Y\ 0)\ 1\ (*\ x\ (f\ x(-Y\ 1)))))(Y\ H)\ 3\ 0)) \\
&= *\ 3\ (*\ 3\ ((\lambda xy.\ \text{IF } (=Y\ 0)\ 1(*\ x\ ((Y\ H)\ x\ (-\ Y\ 1))))\ 3\ 0)) \\
&= *\ 3\ (*\ 3\ 1) \\
&= *\ 3\ 3 \\
&= 9
\end{aligned}
$$

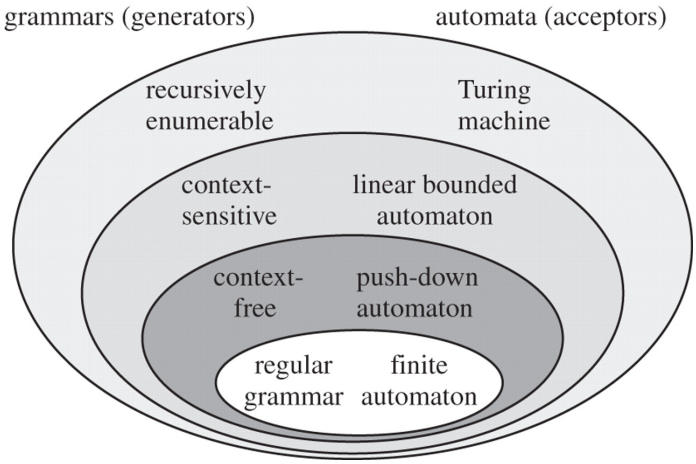
4 Andere rekenparadigma's

Het voorlaatste hoofdstuk gaat over andere rekenparadigma's, maar deze komen niet aan bod in de examenvragen. Het gaat hier onder andere over cellulaire automaten, DNA-computing en Ant-computing. Ookal worden ze niet gevraagd, zeker eens de moeite waard om te lezen.

5 Talen en Complexiteit

5.1 Inleiding

We hebben nu verschillend lagen van talen met complexiteiten gezien. We kunnen nu een overzicht vormen van al deze talen en een uitgebreide vergelijking maken. Het hoofdstuk in de cursus bevat echter maar één enkele pagina, dus ik raad zeker aan deze eens te lezen.



Figuur 7: Overzicht van de Chomsky-hiërarchie

Vraag 20. Vertel alles wat je weet i.v.m. de Chomsky-hiërarchie binnen de 5 minuten. Zijn de talen van een *TM* beslisbaar en wat is de tijdscomplexiteit doorheen de Chomsky-hierarchie?

De Chomsky-hiërarchie is de opdeling van formele talen in klassen op basis van het type grammatica dat de taal kan genereren. Het is een hiërarchie omdat elke klasse erin ook de klasse met een hoger nummer (lager in de hierarchie) omvat. De indeling naar type zegt iets over de uitdrukkingskracht en de complexiteit van generatie en interpretatie. De opdeling ziet er uit als volgt:

Type	**Taalklasse**	**Automatenmodel**	**Grammatica**
Type 3	Regulier	Eindig	Regulier
Type 2	Contextvrij	Push-down	Contextvrij
Type 1	Contextsensitief	Lineair begrensd	Contextsensitief
Type 0	Herkenbaar	Turingmachine	Elke grammatica
Geen type	Alle talen	Geen	Geen

Tabel 1: De Chomsky-hiërarchie

Het is gemakkelijk in te zien dat een reguliere taal beslist wordt in $O(n)$ stappen met n het aantal symbolen aan de input. Dit is logisch aangezien we voor elk karakter van de input string een overgang in de machine moeten maken. Voor *CFL's* bestaat er een $O(n^3)$ algoritme, genaamd *Cocke-Younger-Kasami*.

Voor andere talen is dit niet zo duidelijk. Dit is ook niet aan bod gekomen in de lessen. Voor meer informatie hierover kun je altijd de Wikipedia pagina bekijken.