Limits of algorithmic computation

Leereenheid 10

Limits of algorithmic computation

INTRODUCTIE

In deze cursus hebt u kennis gemaakt met verschillende families van formele talen en hebt u drie soorten automaten leren hanteren die in relatie staan met formele talen. Onder die automaten zijn de turingmachines de krachtigste. De stelling van Church-Turing (zie Linz chapter 9 en chapter 13) zegt dat een turingmachine kan worden ontworpen voor elk beslissingsprobleem dat opgelost kan worden met een algoritme. Een oplossing vinden voor een probleem hangt sterk af van de aard van het probleem zelf, en er bestaan problemen waarvoor geen algoritme gevonden kan worden. Zulke problemen heten onbeslisbaar.
Voor informatici in het algemeen, en programmeurs in het bijzonder, is het goed om te weten dat er (algoritmische) problemen bestaan waarvoor geen oplossing is. In deze leereenheid maakt u kennis met een aantal problemen die in de categorie onbeslisbaar vallen. Het meest bekende daarvan is het stopprobleem voor turingmachines. Alan Turing heeft in 1936 bewezen dat het stopprobleem onbeslisbaar is.
Verder leert u in deze leereenheid de techniek van reductie, die kan helpen om aan te tonen dat een probleem onbeslisbaar is.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
– kunt uitleggen wat de concepten beslisbaarheid en berekenbaarheid inhouden
– kunt uitleggen hoe reductie gebruikt kan worden bij beslisbaarheidsproblemen
– het stopprobleem voor turingmachines, het state-entry-probleem en het blank-tape-stopprobleem kunt omschrijven
– kunt aangeven hoe aangetoond kan worden dat het state-entry-probleem en het blank-tape-stopprobleem onbeslisbaar zijn
– voor een gegeven eenvoudig probleem met het gebruik van reductie kunt aantonen dat het probleem onbeslisbaar is.

Studeeraanwijzingen
Bij deze leereenheid hoort chapter 12 van het tekstboek. Alleen section 12.1 behoort tot de leerstof. Aan het begin van de leereenheid maken we tevens gebruik van theorie uit de chapters 10 en 11 van Linz.
Deze leereenheid gaat over een moeilijk onderwerp; het kan helpen om de stof twee of drie keer te bestuderen, met tussenpozen van enkele dagen om het te laten bezinken.

De studielast van deze leereenheid bedraagt circa 6 uur.

LEERKERN

In de chapters 10 en 11 van Linz (die we niet bestudeerd hebben) staan een paar belangrijke onderdelen die we hier ter introductie op deze leereenheid willen presenteren.

Universele turingmachine

Bestudeer de beschrijving van de *universele turingmachine* in section 10.4 op de pagina's 276 en 277 tot en met de zin "Finally, tapes 2 and 3 will be modified to reflect the result of the move". Belangrijk is dat u inziet dat een turingmachine een andere vorm kan aannemen dan u in leereenheid 9 hebt geleerd. Belangrijk is ook dat u begrijpt hoe de werking van een turingmachine met behulp van nullen en enen beschreven kan worden.

Recursief opsombaar

Een taal L over een alfabet Σ is *recursief opsombaar* als er een turingmachine bestaat die de taal accepteert; zie hiervoor leereenheid 9 en definition 11.1 in Linz. Elke string uit L brengt de turingmachine in een eindtoestand. Voor strings die niet tot L behoren is niets gespecificeerd.

Recursieve taal

Een taal L over een alfabet Σ is *recursief* als er een turingmachine bestaat die L accepteert en die stopt voor elke string uit Σ^+; zie hiervoor definition 11.2 in Linz. Zo'n turingmachine kan dus voor iedere string beslissen of die string wel of niet tot de taal behoort.

Studeeraanwijzing

Lees nu de introductie op chapter 12 in het tekstboek van Linz.

1 Some problems that cannot be solved by Turing machines

1.1 COMPUTABILITY AND DECIDABILITY

Studeeraanwijzing

Bestudeer in section 12.1 van het tekstboek van Linz de introductie en subsection Computability and decidability.

OPGAVE 10.1

a Leg uit wat precies bedoeld wordt met "By a problem we will understand a set of related statements".
b Wanneer is een probleem beslisbaar?
c Gegeven is de bewering "Het is een onbeslisbaar probleem om te bepalen of een context-vrije grammatica wel of niet dubbelzinnig is". Toch hebben we in opgave 5.19 bewezen dat de context-vrije grammatica van example 5.4 dubbelzinnig is. Is de uitwerking van opgave 5.19 dan niet tegenstrijdig met de gegeven bewering?

1.2 THE TURING MACHINE HALTING PROBLEM

Studeeraanwijzing

In deze subsection wordt een speciaal probleem behandeld. Het gaat namelijk over het oplossen *door een turingmachine* van een probleem *betreffende turingmachines*. In het algemeen hoeven er helemaal geen turingmachines aan te pas te komen bij beslisbaarheids- en berekenbaarheidsvraagstukken. Met andere woorden, in het algemeen hoeft het domein van het probleem geen turingmachines te bevatten, en het algoritme dat het probleem oplost hoeft niet als turingmachine beschreven te zijn.
Bestudeer nu in section 12.1 van het tekstboek van Linz de subsection The Turing machine halting problem, tot en met definition 12.1.

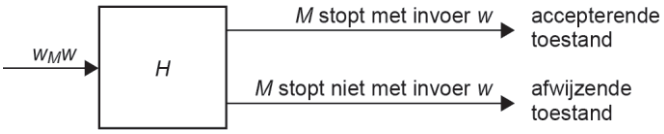
OPGAVE 10.2
a Geef in eigen woorden een beschrijving van het stopprobleem voor turingmachines.
b Beschrijf in eigen woorden de essentie van definition 12.1.

Studeeraanwijzing theorem 12.1

We zijn nu toe aan het bestuderen van theorem 12.1, waarin gesteld wordt dat er geen turingmachine bestaat die het stopprobleem voor turingmachines oplost.
Het bewijs van theorem 12.1 hoeft u maar door te lezen en proberen te begrijpen, bijvoorbeeld aan de hand van de extra uitleg in het werkboek. Er wordt niet van u verwacht dat u het bewijs kunt reproduceren.

Het idee achter het bewijs van theorem 12.1 is het volgende: we beginnen met aan te nemen dat er wel een turingmachine H bestaat die het stopprobleem oplost. Vervolgens transformeren we H tot een turingmachine H', en daarna transformeren we H' nog eens tot $\hat{H}$. Tot slot passen we $\hat{H}$ op zichzelf toe, en dan beginnen de problemen: $\hat{H}$ spreekt zichzelf namelijk tegen.
Het is belangrijk dat u inziet dat die problemen niet veroorzaakt worden door de uitgevoerde transformaties: die transformaties zijn niets anders dan volkomen legale aanpassingen aan een gegeven turingmachine, die leiden tot een nieuwe turingmachine (zie ook opgave 10.4).

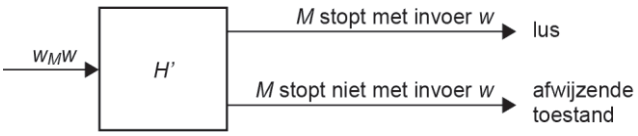
We illustreren het bewijs van theorem 12.1 met een viertal schema's, in figuur 10.1 tot en met 10.4.
De turingmachine H, waarvan we aannemen dat het een implementatie is van het algoritme dat het stopprobleem voor turingmachines oplost, is getekend in figuur 10.1:



FIGUUR 10.1 De turingmachine H

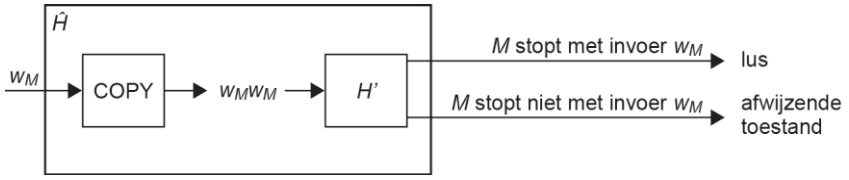
H is dus een turingmachine die bepaalt of een gegeven turingmachine al dan niet stopt met een gegeven invoer, en M is een turingmachine waarvan we willen weten of die stopt op invoer w. De invoer van H bestaat dus uit M en w, samen gecodeerd in de string $w_M w$. Merk op dat H altijd stopt, ook als M niet stopt.

We gaan nu H transformeren naar H', en wel zo dat H' in een oneindige lus komt dan en slechts dan als H stopt in q_y (zie figuur 10.2). De transformatie bestaat uit het toevoegen van een paar nieuwe toestanden en overgangen aan 'het eind van' H (zie figure 12.1 en 12.2 in Linz).



FIGUUR 10.2 De turingmachine H'

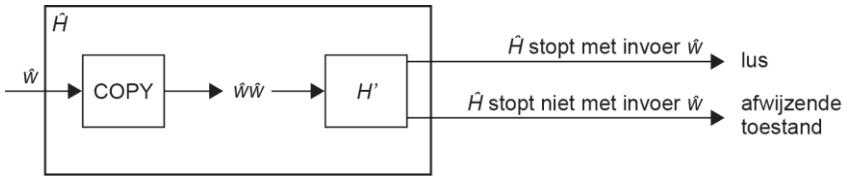
H' doet dus iets anders dan H, maar dat geeft niet: het gaat erom dat duidelijk is dat "als H bestaat, dan bestaat H' ook".
Vervolgens transformeren we H' naar $\hat{H}$, deze keer door aan 'het begin van' H' nieuwe toestanden en overgangen toe te voegen die ervoor zorgen dat $\hat{H}$ eerst zijn invoer kopieert. Zodra dat gebeurd is laten we $\hat{H}$ verdergaan zoals H' dat zou doen. We laten $\hat{M}$ werken met een ander soort invoer dan H': we geven alleen de beschrijving w_M van M aan $\hat{H}$, en laten w dus weg. In figuur 10.3 geven we een schematische weergave van $\hat{H}$ met w_M als invoer:



FIGUUR 10.3 De turingmachine $\hat{H}$ met invoer w_M

Weer geldt: $\hat{H}$ doet iets heel anders dan H', maar dat maakt niet uit. Het doel was alleen maar te laten zien dat $\hat{H}$ bestaat (onder de aanname dat H en dus H' bestaat), zodat we in de volgende stap een tegenspraak kunnen creëren.

We zijn nu klaar met transformeren. Aangezien $\hat{H}$ een turingmachine is, bestaat er een codering $\hat{w}$ van $\hat{H}$, die we als invoer aan $\hat{H}$ kunnen aanbieden. Die situatie is getekend in figuur 10.4:



FIGUUR 10.4 De turingmachine $\hat{H}$ met invoer $\hat{w}$

We hebben nu een situatie gecreëerd waarin $\hat{H}$ zichzelf tegenspreekt: als $\hat{H}$ stopt met invoer $\hat{w}$ dan komt $\hat{H}$ in een oneindige lus, en als $\hat{H}$ niet stopt met invoer $\hat{w}$ dan stopt $\hat{H}$ in een afwijzende toestand. Dit is een situatie die nooit voor kan komen, en omdat al onze redeneerstappen (de transformaties van H naar H' naar $\hat{H}$ en het toepassen van $\hat{H}$ op een beschrijving $\hat{w}$ van zichzelf) gezond waren, moet de oorzaak van deze onmogelijke situatie wel in onze aanname liggen. Die aanname – dat er een turingmachine H bestaat die het stopprobleem voor turingmachines oplost – is dus niet waar.

OPGAVE 10.3
Beschrijf waarom aan het einde van het bewijs geconcludeerd wordt: "This is clearly nonsense".

OPGAVE 10.4
Maak exercise 1 uit section 12.1 van Linz.

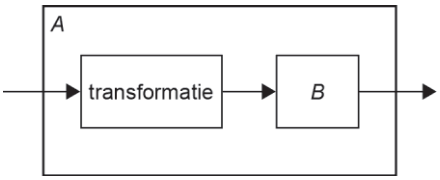
Studeeraanwijzing Bestudeer nu de rest van subsection The Turing machine halting problem.

1.3 REDUCING ONE UNDECIDABLE PROBLEM TO ANOTHER

In het geval van het stopprobleem hebben we in theorem 12.1 bewezen dat dit onbeslisbaar is door alleen (een hypothetisch algoritme voor) het stopprobleem zelf te gebruiken, en daarop een – weliswaar gekunstelde en vergezochte, maar in ieder geval geldige – logische redenering toe te passen: 'als we dat algoritme een beetje aanpassen, krijgen we iets dat niet kan'.
Als we eenmaal een probleem hebben waarvan we weten dat het onbeslisbaar is, kunnen we dat gebruiken om van andere problemen aan te tonen dat ze onbeslisbaar zijn. We hoeven dan niet meer per probleem een slimme truc zoals in het bewijs van theorem 12.1 te bedenken, maar kunnen een algemeen toepasbare methode gebruiken: *reductie* (overigens moet er dan nog steeds wel per probleem een 'trucje' verzonnen worden). Op Wikipedia staat op de pagina over berekenbaarheid een heel duidelijke omschrijving van het begrip reductie:

Reductie

Een belangrijk hulpmiddel bij het bepalen of een gegeven probleem berekenbaar (of beslisbaar) is of niet, is de techniek van *reductie*.
Een reductie van een probleem *A* naar een ander probleem *B* is een formele manier om *A* uit te drukken in termen van *B* (transformeren naar *B*), zodanig dat een oplossing voor *B* ook gebruikt kan worden om *A* op te lossen. Enerzijds kun je zo'n reductie gebruiken om *A* op te lossen indien voor *B* al een oplossing bekend is. Anderzijds, indien al bekend is dat *A* een onberekenbaar (of onbeslisbaar) probleem is, kun je met een reductie aantonen dat *B* ook onberekenbaar (of onbeslisbaar) is; zie figuur 10.5.



FIGUUR 10.5 Reductie van *A* naar *B*

Anders geformuleerd in termen van beslisbaarheid:
als "probleem *A* wordt gereduceerd tot probleem *B*"
dan geldt:

"als *B* beslisbaar is, dan is *A* ook beslisbaar"

ofwel

"als *A* onbeslisbaar is, dan is *B* ook onbeslisbaar".

In de examples 12.1 en 12.2 – in de subsection die u zo dadelijk gaat bestuderen – wordt steeds als probleem *A* het stopprobleem voor turingmachines gebruikt. Om te bewijzen dat een ander probleem onbeslisbaar is, wordt het stopprobleem gereduceerd tot dat andere probleem.

NB

– Een veel gemaakte fout bij reduceren is 'de verkeerde kant op reduceren'. Om te bewijzen dat *B* onbeslisbaar is, moet een probleem *A*, waarvan bekend is dat het onbeslisbaar is, gereduceerd worden tot *B*. Dus niet andersom!

– Om de techniek van reductie te kunnen gebruiken moet een transformatie van *A* naar *B* bestaan. Zonder bruikbare transformatie, geen reductie!

Wat we eerder over beslisbaarheids- en berekenbaarheidsvraagstukken zeiden, geldt ook voor reducties: in subsection 12.1 komen daar steeds turingmachines in voor, omdat we nu eenmaal bezig zijn de grenzen van turingmachines te verkennen. In het algemeen kunnen reducties ook gebruikt worden voor problemen die niets met turingmachines te maken hebben.

Terminologie

In het vervolg zullen we de volgende terminologie zo consequent mogelijk hanteren: we gebruiken de term algoritme voor de turingmachine die een bepaald probleem oplost. De turingmachine die als invoer van het algoritme dient blijven we een turingmachine noemen (dit geldt natuurlijk alleen als het probleem ook daadwerkelijk over turingmachines gaat). Natuurlijk is het zo dat een algoritme en een turingmachine equivalent zijn, maar we willen de twee rollen uit elkaar houden door verschillende namen te gebruiken.

Studeeraanwijzing

Bestudeer in section 12.1 van het tekstboek van Linz de subsection Reducing one undecidable problem to another, tot en met example 12.1.

Example 12.1

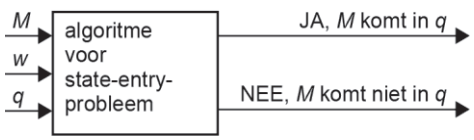
We gaan hier verder in op example 12.1 door het bewijs te illustreren met een aantal schema's.

State-entry-probleem

Gegeven een willekeurige turingmachine *M*, een willekeurige toestand *q* van *M* en een willekeurige invoerstring *w* van *M*. Het *state-entry-probleem* is de vraag of *M* met *w* als invoer ooit in toestand *q* komt.

Er wordt gesteld dat het probleem onbeslisbaar is. Het bewijs wordt gegeven door het stopprobleem te reduceren tot het state-entry-probleem, ofwel door te laten zien dat een algoritme voor het state-entry-probleem eenvoudig zou kunnen worden omgevormd tot een algoritme voor het stopprobleem.

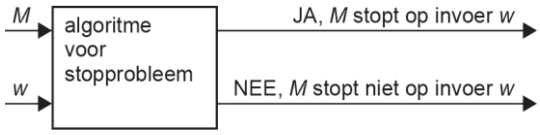
Stel we hebben een algoritme (geïmplementeerd met een turingmachine) voor het state-entry-probleem; zie figuur 10.6:



FIGUUR 10.6 Een algoritme voor het state-entry-probleem

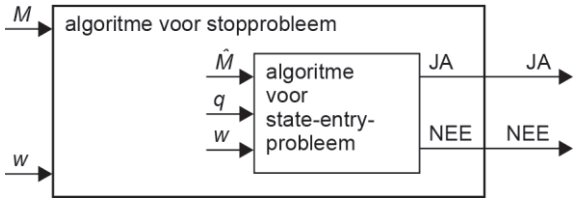
Dit algoritme heeft dus als invoer een turingmachine, een invoerstring en een toestand van die turingmachine, en geeft als uitvoer het antwoord 'JA' of 'NEE'.

We willen een algoritme maken voor het stopprobleem; zie figuur 10.7:



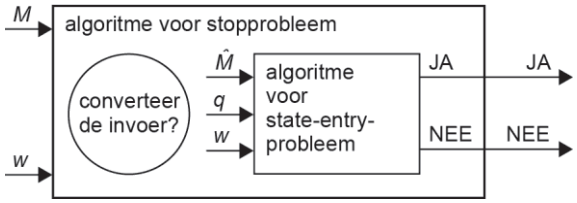
FIGUUR 10.7 Een algoritme voor het stopprobleem

Voor dit laatste algoritme willen we gebruikmaken van het algoritme voor het state-entry-probleem. We willen dus het stopprobleem reduceren tot het state-entry-probleem; zie figuur 10.8:



FIGUUR 10.8 Reductie tot het state-entry-probleem (onvolledig)

Welke transformatie moet er plaatsvinden? Ofwel: hoe moeten we de invoer van het stopprobleem transformeren tot invoer voor het state-entry-probleem, zo dat een JA-antwoord van het state-entry-probleem een JA-antwoord van het stopprobleem betekent, en een NEE-antwoord van het state-entry-probleem een NEE-antwoord van het stopprobleem? Figuur 10.9 toont een schematische weergave van deze vraag.

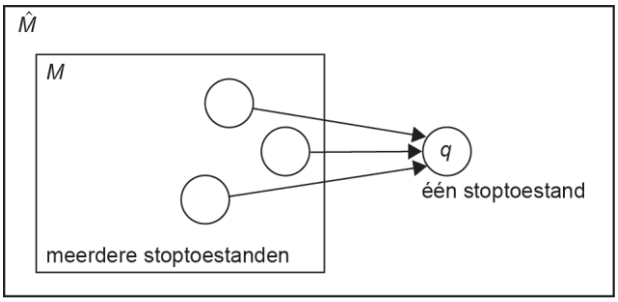


FIGUUR 10.9 Welke transformatie moet er plaatsvinden?

De invoer (M, w) voor het stopprobleem moet worden getransformeerd tot invoer ($\hat{M}$, q, w) voor het state-entry-probleem, en wel zo dat

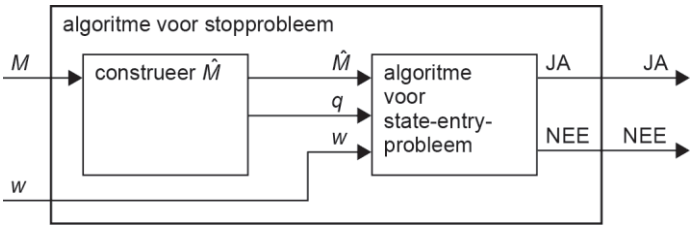
M stopt op invoer w
dan en slechts dan als
$\hat{M}$ komt in toestand q op invoer w.

Dit kunnen we bereiken door een nieuwe toestand q toe te voegen aan M en door, voor elke stoptoestand van M, een overgang naar q toe te voegen. Het maakt niet zoveel uit wat er dan in toestand q gebeurt: het gaat er alleen maar om dat $\hat{M}$ in toestand q komt dan en slechts dan als M stopt. In example 12.1 is ervoor gekozen om q een eindtoestand te maken.
Figuur 10.10 toont de turingmachine $\hat{M}$.



FIGUUR 10.10 De turingmachine $\hat{M}$ met eindtoestand q

Gebruiken we deze transformatie in figuur 10.9, dan krijgen we een algoritme voor het stopprobleem; zie figuur 10.11 (zie ook figuur 10.7).



FIGUUR 10.11 Een algoritme voor het stopprobleem dat gebruikmaakt van een algoritme voor het state-entry-probleem

Omdat het stopprobleem onbeslisbaar is, is het niet mogelijk om er een algoritme voor te construeren. Toch hebben we dat gedaan in figuur 10.11, dankzij onze aanname dat er een algoritme bestaat voor het state-entry-probleem. Die aanname is dus niet juist: het state-entry-probleem is onbeslisbaar.

OPGAVE 10.5
In example 12.1 wordt een manier beschreven waarop je aan de overgangsfunctie van een turingmachine M kunt zien of M stopt: "If M halts, it does so because some $\delta(q_i, a)$ is undefined". Waarom kunnen we deze observatie niet gebruiken om het stopprobleem op te lossen?

Tot zover example 12.1; we bekijken nu example 12.2.

Studeeraanwijzing Example 12.2 Bestudeer example 12.2 in de subsection Reducing one undecidable problem to another van section 12.1 van het tekstboek van Linz.

Blank-tape-stopprobleem

Het *blank-tape-stopprobleem* is de vraag of, gegeven een willekeurige turingmachine, deze turingmachine stopt met een lege tape, dat wil zeggen met invoer λ. Dit probleem is onbeslisbaar, en dat kunnen we bewijzen met behulp van de techniek van reductie. De volgende twee vragen moeten dan worden beantwoord:
– Welke reductie kunnen we toepassen om te bewijzen dat het blank-tape-stopprobleem onbeslisbaar is?
– Welke transformatie hebben we daarvoor nodig?

Een antwoord op de eerste vraag is: we kunnen proberen of we het stopprobleem (probleem A in figuur 10.5) kunnen reduceren tot het blank-tape-stopprobleem (probleem B in dezelfde figuur). Als dat kan (als we een geschikte transformatie kunnen vinden), dan zou uit een algoritme voor het blank-tape-stopprobleem een algoritme voor het stopprobleem geconstrueerd kunnen worden (zie figure 12.3 in Linz). We weten echter dat dit laatste algoritme niet kan bestaan, en kunnen dan dus concluderen dat er ook geen algoritme voor het blank-tape-stopprobleem kan bestaan.

Voor de transformatie (het antwoord op de tweede vraag) moeten we de invoer voor het algoritme A voor het stopprobleem transformeren naar invoer voor het algoritme B voor het blank-tape-stopprobleem, en wel zo dat een JA-antwoord van B een JA-antwoord van A betekent, en een NEE-antwoord van B een NEE-antwoord van A. Met andere woorden, we moeten een turingmachine M, waarvan we willen weten of die stopt op invoer w, transformeren naar een turingmachine M_w, en wel zo dat

M stopt op w dan en slechts dan als M_w stopt op λ

Hoewel M_w dus start met een lege tape, moet hij toch na kunnen gaan of M zal stoppen met invoer w. Dat kunnen we voor elkaar krijgen door M_w als volgt uit M en w te construeren: M_w bevat alle toestanden en overgangen van M, plus een aantal nieuwe. Die nieuwe toestanden en overgangen gebruikt M_w om eerst de invoerstring w op zijn lege tape te schrijven (de string w is immers bekend, en het is eenvoudig om een stukje 'code' voor een turingmachine te schrijven waarmee een gegeven string op een lege band wordt gezet). Vervolgens zet M_w de leeskop op het meest linkse symbool van w en gaat in de begintoestand q_0 van M. Hierna doet M_w precies wat M zou doen vanuit de configuratie q_0w. Nu geldt dus dat de turingmachine M stopt met invoer w dan en slechts dan als de turingmachine M_w stopt met λ als invoer.

Samengevat: we kunnen vanuit iedere turingmachine M en iedere invoerstring w een turingmachine M_w construeren die werkt zoals hierboven beschreven. Als we een algoritme hebben dat het blank-tape-stopprobleem oplost kunnen we dat algoritme loslaten op M_w, en dan krijgen we daarmee een antwoord op de vraag of M stopt op w. We weten echter dat die laatste vraag in het algemeen niet te beantwoorden is, dus kunnen we concluderen dat dat algoritme voor het blank-tape-stopprobleem niet kan bestaan.

OPGAVE 10.6
Beschrijf de transformatie van (M, w) naar M_w uit example 12.2.

Studeeraanwijzing Bestudeer nu de rest van de subsection Reducing one undecidable problem to another van section 12.1 van Linz.

OPGAVE 10.7
Deze opgave gaat over example 12.3 in Linz.
a In figure 12.4 wordt gebruik gemaakt van een turingmachine met de naam M_u. Geef een omschrijving van M_u.
b Verklaar de titel van figure 12.4.

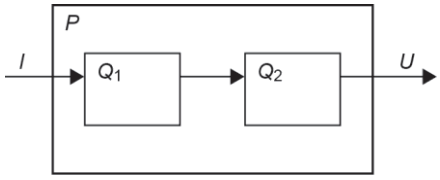
OPGAVE 10.8
Deze opgave gaat ook over example 12.3 in Linz.
Maak exercise 11 uit section 12.1 van Linz.

OPGAVE 10.9
Maak exercise 3 uit section 12.1 van Linz.

OPGAVE 10.10
Maak exercise 9 uit section 12.1 van Linz.

ZELFTOETS

1 Gegeven een probleem P dat gereduceerd kan worden tot twee deelproblemen Q_1 en Q_2, volgens de volgende figuur. Q_1 is algoritmisch.



Welke van de volgende vier beweringen over P en Q_2 is in ieder geval *niet* waar:

– P en Q_2 zijn beide algoritmisch.
– P is algoritmisch, maar Q_2 is dat niet.
– P is niet algoritmisch, maar Q_2 is dat wel.
– P en Q_2 zijn geen van beide algoritmisch.

2 a We definiëren probleem P als volgt: gegeven een string l over een zeker alfabet, bepaal of l even lengte heeft. Geef aan hoe u invoer voor probleem P kunt transformeren tot invoer voor het blank-tape-stop-probleem.
b Waarom is het incorrect om de reductie die in onderdeel a gesuggereerd wordt te gebruiken om te concluderen dat P onbeslisbaar is?

TERUGKOPPELING

1 Uitwerking van de opgaven

10.1 a Een probleem kunnen we beschouwen als een verzameling uitspraken die alleen verschillen in het element van het domein dat erin is ingevuld. In het voorbeeld dat Linz geeft, de bewering "For a context-free grammar G, the language $L(G)$ is ambiguous", kun je deze ene bewering ook opvatten als een hele verzameling beweringen of uitspraken, namelijk één uitspraak voor iedere concrete context-vrije grammatica die we kunnen verzinnen. Het 'probleem' is dus de algemene versie van die uitspraken.

b Een probleem is beslisbaar als er een turingmachine bestaat die voor ieder element in het domein van het probleem het correcte antwoord geeft (ja of nee) op de specifieke uitspraak die bij dat element hoort.

c De uitwerking van opgave 5.19 is niet tegenstrijdig met de bewering "Het is een onbeslisbaar probleem om te bepalen of een context-vrije grammatica wel of niet dubbelzinnig is".
De uitwerking geeft een antwoord op het al dan niet dubbelzinnig zijn van één gegeven context-vrije grammatica. Om het antwoord te geven maakt de uitwerking gebruik van de eigenschappen van de gegeven grammatica.
Het domein van de bewering is de verzameling van alle context-vrije grammatica's. De bewering gaat over een algoritme dat voor elke willekeurige context-vrije grammatica G een antwoord zou moeten geven op de vraag of G wel of niet dubbelzinnig is. Zo'n algoritme kan niet uitgaan van de eigenschappen van G, omdat over G niets anders bekend is dan dat het een context-vrije grammatica is. Daarom is zo'n algoritme niet te vinden.

10.2 a Het stopprobleem voor turingmachines is de vraag of, gegeven een willekeurige turingmachine M en een willekeurige string w, de turingmachine wel of niet zal stoppen met de string w als invoer.

b Definition 12.1 zegt dat een oplossing voor het stopprobleem voor turingmachines een turingmachine H is die, als hij de beschrijving w_M van een turingmachine M en een invoerstring w voor M krijgt aangeboden, stopt in één van de twee eindtoestanden q_y (accepterende eindtoestand) en q_n (afwijzende eindtoestand). H stopt in eindtoestand q_y dan en slechts dan als M stopt op de invoer w en H stopt in eindtoestand q_n dan en slechts dan als M niet stopt op de invoer w.

10.3 In plaats van de beschrijving van M wordt de beschrijving $\hat{w}$ van $\hat{H}$ aan $\hat{H}$ aangeboden. Er zijn dan hoogstens twee mogelijkheden: óf $\hat{H}$ stopt op invoer $\hat{w}$, óf $\hat{H}$ stopt niet op invoer $\hat{w}$. Beide mogelijkheden leiden tot een tegenspraak, want de een zegt "als $\hat{H}$ stopt op invoer $\hat{w}$, dan stopt $\hat{H}$ niet op invoer $\hat{w}$", en de ander zegt "als $\hat{H}$ niet stopt op invoer $\hat{w}$, dan stopt H op invoer $\hat{w}$". Er blijven dus geen mogelijkheden over, en dat is de reden waarom Linz zegt "This is clearly nonsense".

10.4 We kunnen de overgangsfunctie als volgt uitbreiden voor alle $a \in \Gamma$:

$\delta(q_y, a) = (q_a, a, R)$
$\delta(q_a, a) = (q_b, a, R)$
$\delta(q_b, a) = (q_a, a, L)$

Q is nu uitgebreid met q_a en q_b. F is ook gewijzigd: q_y is geen eindtoestand meer.

10.5 Je kunt inderdaad aan iedere turingmachine zien of hij ongedefinieerde toestandsovergangen heeft (combinaties van een toestand q en een tapesymbool a waarvoor niet is vastgelegd wat er moet gebeuren). Als de turingmachine in zo'n situatie belandt, zal hij ook inderdaad stoppen. Het probleem is dat we niet kunnen bepalen *of* hij in zo'n situatie terecht zal komen, en in het geval van het stopprobleem is dat nu juist precies wat we moeten weten.
In het geval van example 12.1 hoeven we niet te weten *of M* stopt, we willen er alleen voor zorgen dat *als M* stopt hij automatisch overgaat in een speciale toestand. En dat kan wel, op de manier die in example 12.1 beschreven is.

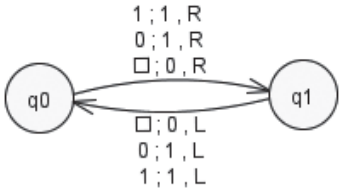
10.6 Laat $w = w_1w_2\ldots w_n$ voor w_i in het invoeralfabet van M en $n \geq 1$ (als $n = 0$ is M_w gewoon gelijk aan M).
M_w heeft alle toestanden en overgangen van M, alleen is de begintoestand q_0 van M in M_w een gewone toestand. M_w heeft verder een nieuwe begintoestand q_{in} en nieuwe gewone toestanden $q_1, \ldots, q_n$, en overgangen $\delta(q_{in}, \square) = (q_1, w_1, R)$, $\delta(q_1, \square) = (q_2, w_2, R), \ldots, \delta(q_{n-1}, \square) = (q_n, w_n, L)$, waarmee w op de band gezet wordt. Daarna heeft M_w voor iedere w_i een overgang $\delta(q_n, w_i) = (q_n, w_i, L)$, waarmee de kop weer naar het begin van w loopt. Vervolgens doet M_w de stap $\delta(q_n, \square) = (q_0, \square, R)$: hij gaat in de begintoestand van M met de kop op het eerste symbool van w. Daarna simuleert hij M.

10.7 a M_u is een universele turingmachine die de beschrijving van $\hat{M}$ als invoer krijgt. M_u beslist of M stopt in m (of minder) stappen.

b De turingmachine die in figure 12.4 wordt geconstrueerd kan gebruikt worden om te beslissen of M wel of niet stopt met een lege tape als invoer. Dit is een turingmachine voor het blank-tape-stopprobleem. In deze turingmachine is aangenomen dat F, een turingmachine voor het berekenen van $f(n)$, bestaat. Deze aanname is onjuist.

10.8 Hoewel de functie niet berekenbaar is, is het mogelijk om in sommige gevallen de waarde van f te berekenen.
Heeft een turingmachine één toestand, dan geldt $f(1) = 0$. Zou $\delta(q_0, \square)$ gedefinieerd zijn, dan zou de turingmachine nooit stoppen. Ongeacht welk tapesymbool op de tape wordt geschreven, gaat in dat geval de turingmachine steeds verder naar rechts (of naar links) en komt steeds een blanco tegen.
Heeft een turingmachine twee toestanden, dan geldt $f(2) = 5$. De overgang $\delta(q_0, \square)$ moet nu gedefinieerd zijn om in q_1 te kunnen komen. Is q_1 een eindtoestand dan stopt de turingmachine na één stap. Maar q_1 hoeft

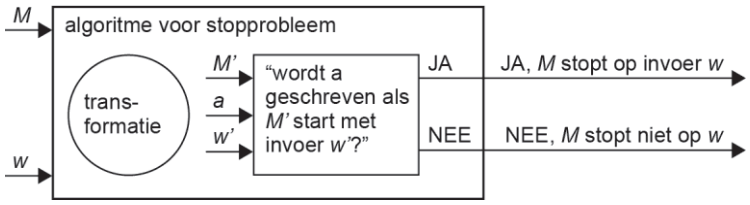
geen eindtoestand te zijn omdat de turingmachine kan stoppen door een ongedefinieerde overgang. We kunnen alle mogelijke overgangen opsommen die geen lus inhouden. Kijk bijvoorbeeld naar de volgende turingmachine:



Eén van de zes overgangen moet ongedefinieerd zijn, wil de turingmachine stoppen. Dit is bijvoorbeeld het geval als $\delta(q_0, \square)$, $\delta(q_1, \square)$, $\delta(q_0, 1)$, $\delta(q_1, 0)$ en $\delta(q_0, 0)$ gedefinieerd zijn en $\delta(q_1, 1)$ ongedefinieerd is. In dat geval kunnen we maximaal 5 bewegingen maken. Andere waarden van de overgangen en een andere keus voor een ongedefinieerde overgang leiden tot vergelijkbare resultaten en we kunnen concluderen dat $f(2) = 5$.

10.9 Stelling: Er bestaat geen algoritme A dat kan bepalen of een tapesymbool $a \in \Gamma$ ooit op de tape geschreven wordt gedurende het toepassen van M op $w \in \Sigma^+$

Om te bewijzen dat een probleem B onbeslisbaar is, kunnen we proberen of we een probleem A waarvan we al weten dat het onbeslisbaar is, kunnen reduceren tot B (figuur 10.5). In de praktijk kan het even zoeken zijn naar een geschikte kandidaat voor A, maar aangezien we bezig zijn met een introductie op dit gebied en aangezien we tot nu toe alleen twee voorbeelden hebben gezien waarin de rol van A wordt vervuld door het stopprobleem, lijkt het een veilige keuze om nu ook voor het stopprobleem te kiezen. We krijgen dan de volgende situatie:



Deze reductie werkt alleen als we een geschikte transformatie kunnen vinden. Als we bedenken dat bovenstaand plaatje wel erg veel lijkt op figuur 10.11 (reductie van het stopprobleem naar het state-entry-probleem) is het vinden van die transformatie niet zo moeilijk meer: we gebruiken gewoon dezelfde 'truc' als in example 12.1, maar in plaats van M zo aan te passen dat hij in een speciale *toestand* gaat dan en slechts dan als hij stopt, zorgen we ervoor dat hij (ook) een speciaal *symbool* schrijft dan en slechts dan als hij stopt. Iets concreter:
We construeren M' uit M door het toevoegen van een speciaal tapesymbool #, een nieuwe accepterende toestand $q_\#$ en nieuwe overgangen $\delta(q, y) = (q_\#, \#, R)$ voor elke $q \in Q$ en $y \in \Gamma$ waarvoor δ nog niet gedefinieerd is.

We hebben er dan voor gezorgd dat
– als M stopt op w, dan schrijft M' een #, en
– als M' een # schrijft, dan stopt M op w.
Kortom, de transformatie is rond, en als er een algoritme bestaat dat bepaalt of een turingmachine tijdens zijn berekening een # schrijft, dan kunnen we dat algoritme dus gebruiken om het stopprobleem op te lossen. We weten echter dat het stopprobleem niet opgelost kan worden, dus 'ons' algoritme bestaat ook niet.
(Een kortere versie van deze uitwerking staat op pagina 426 in Linz.)

10.10 Ja, het stopprobleem is beslisbaar voor deterministische stapelautomaten (dpda's).
Deterministische stapelautomaten kennen regels van de vorm:

$\delta(q_1, a, b) = \varnothing$ $\quad a \in \Sigma$ en $b, c \in \Gamma^*$
$\delta(q_1, a, b) = \{(q_2, c)\}$
$\delta(q_1, \lambda, b) = \varnothing$
$\delta(q_1, \lambda, b) = \{(q_2, c)\}$ $\quad$ als $\delta(q_1, d, b) = \varnothing$ voor alle $d \in \Gamma$

Aangezien de strings eindig zijn, kan de automaat niet oneindig doorgaan met het lezen van letters. Wij zijn dus alleen geïnteresseerd in de regels van de vorm $\delta(q_1, \lambda, b) = \{(q_2, c)\}$. Beter gezegd, wij zijn alleen geïnteresseerd in automaten waar het mogelijk is een oneindig aantal λ's te 'lezen'. Ofwel waar

$\delta'(q_i, \lambda, s) = \{(q_i, sx)\}$ met $s \in \Gamma$ en $x \in \Gamma^*$

mogelijk is zodanig dat de stapel ofwel groeit, ofwel gelijk blijft. Aangezien de stapelautomaat deterministisch is, kan altijd worden voorzien of een dpda voor een string w twee maal in zo'n q_i met een gelijke of grotere stapel terecht komt. Als dat het geval is zal de deterministische stapelautomaat niet stoppen.

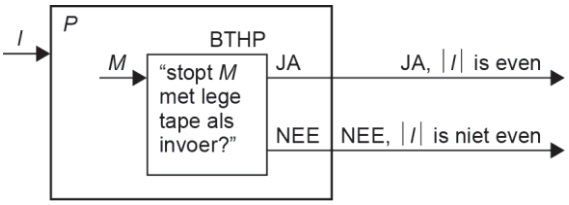
Formeler: wij kunnen een turingmachine maken die gegeven een dpda M en een string w voor elke λ-stap registreert uit welke toestand (q_i, sx), met $q_i \in Q$, $s \in \Gamma$ en $x \in \Gamma^*$, machine M komt. Bij het uitvoeren van een niet-λ-stap wordt dit register geschoond. Na elke λ-stap kan gecontroleerd worden of de automaat in een toestand (q_i, sy) is beland, $y \in \Gamma^*$, met $|y| \geq |x|$. Zo ja, dan beslist de turingmachine dat automaat M toegepast op w niet stopt.

2 Uitwerking van de zelftoets

1 Het derde alternatief is in elk geval niet waar.

Als Q_2 algoritmisch is, dan volgt uit de gegeven reductie dat P ook algoritmisch is. Het eerste alternatief kan dus waar zijn, maar het derde niet.
Is Q_2 niet-algoritmisch, dan weten we nog niets over P. De reductie die hier getoond is, is in dat geval zeker geen algoritme voor P. Maar er kan een andere reductie bestaan, die wel geheel uit algoritmische stappen bestaat. Zowel het tweede als het vierde alternatief zijn dus mogelijk.

2 a Als we de situatie die door de genoemde transformatie wordt gesuggereerd tekenen zoals in figuur 10.5, dan krijgen we het volgende plaatje:



We moeten dus, uitgaande van alleen een string l, een turingmachine M construeren die start met een lege tape én die stopt dan en slechts dan als $|l|$ even is. Die turingmachine M kan er dan als volgt uitzien:

Start met een lege tape.
Zet l op de tape (vervang steeds □ door een symbool van l).
Ga m.b.v. twee toestanden q_{even} en q_{oneven} na of $|l|$ even is.
    Zo ja: ga vanuit q_{even} naar een eindtoestand.
    Zo nee: ga vanuit q_{oneven} in een oneindige lus.

Het moge duidelijk zijn dat nu geldt
$|l|$ is even dan en slechts dan als M stopt met invoer λ.

b Het enige dat we weten is dat het blank-tape-stopprobleem onbeslisbaar is. Dat betekent dat we geen algoritme voor P kunnen construeren uit een algoritme voor het blank-tape-stopprobleem (want dat laatste bestaat niet), maar het betekent niet per se dat er geen ander algoritme kan zijn dat we kunnen ombouwen tot een algoritme voor P. Anders gezegd: een reductie van P tot het blank-tape-stopprobleem kan alleen gebruikt worden voor de volgende twee (equivalente) uitspraken:
– "Als P onbeslisbaar is, dan is het blank-tape-stopprobleem ook onbeslisbaar".
– "Als het blank-tape-stopprobleem beslisbaar is, dan is P beslisbaar".
De uitspraak "Als het blank-tape-stopprobleem onbeslisbaar is, dan is P onbeslisbaar" is niet equivalent met bovenstaande twee uitspraken, en daarom kan de gesuggereerde reductie niet gebruikt worden om aan te tonen dat P onbeslisbaar is.

NB En dat is maar goed ook, want P is wel beslisbaar: het is gewoon een instantie van het lidmaatschapsprobleem voor reguliere talen ("zit l in de reguliere taal $L = \{ w : |w| \text{ is even}\}$?").