Is elf plus één twaalf?; over rekenen en puzzelen

Peter van Emde Boas

ILLC, Faculteit Wiskunde en Informatica, Universiteit van Amsterdam,
Plantage Muidergracht 24, 1018 TV Amsterdam
8 september 1999

De puzzel

Het thema van deze demonstratie is een op het eerste gezicht weinig indrukwekkende legpuzzel. Op tafel ligt een 15-tal stapeltjes vierkante plankjes. De voorkant van deze plankjes is verdeeld in vier gekleurde driehoeken. In de achterkant zijn twee gaten geboord die ervoor zorgen dat de plankjes slechts op één manier opgehangen kunnen worden op het bord waarop de puzzel gemaakt moet worden. Op het bord is tevens een rand aangebracht van gekleurde stroken. Doel is het gehele bord vol te hangen met plankjes uit de op tafel liggende stapels, en wel op een zodanige wijze dat overal waar twee plankjes tegen elkaar aan liggen de kleuren overeenstemmen.

De ervaring leert dat het voor een middelbare scholier niet al te lastig is de puzzel te maken. Zeker als de instructeur daarbij enige voor de hand liggende aanwijzingen geeft, zoals:

- Gebruik bij voorkeur geen zeldzame stukken als je ook een stuk uit een grotere stapel kunt gebruiken (de verschillende stapels zijn immers bij aanvang van ongelijke hoogte)

- Probeer de puzzel systematisch van boven naar beneden, en per rij, van links naar rechts op te lossen (hierbij blijkt het nuttig te zijn om iets onder het midden de richting van links naar rechts om te draaien, maar dat wordt pas duidelijk als je de puzzel aan het oplossen bent)

Als de puzzel eenmaal is opgelost ontstaat een plaatje als aangegeven in figuur 1. De oplossing laat kolommen oninteressante blokken zien die worden herhaald. Er loopt echter een strook schuin doorheen die links boven begint, halverwege tegen de rechterrand wordt teruggekaatst en tenslotte beneden in het midden eindigt. Bovendien blijken de kolommen ter hoogte van de terugkerende interessante strook van kleur te verschieten (althans voorzover de strook doorloopt).

Hoe we met de puzzel kunnen rekenen

De relatie met de in de titel gestelde vraag laat zich als volgt leggen. Kijkende naar de bovenste strook kunnen we een verband leggen tussen kleuren en (binaire) cijfers: oranjerood staat voor 0 en donkerblauw staat voor 1 ; de kleur wijnrood geven wij de speciale functie van "einde getal". In deze interpretatie kunnen we de bovenrand lezen als een binair getal: de tekenrij 01011 laat zich volgens de bekende interpretatie van binaire getallen ontcijferen tot: $0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 8 + 2 + 1 = 11$. Op dezelfde wijze laat de rij op de bodem die wij kunnen ophangen nadat de puzzel gemaakt is zich ontcijferen tot 01100 met waarde $8 + 4 = 12$. Volgens deze interpretatie is de waarde van de onderrij één hoger dan die van de bovenrij waarmee we zijn begonnen. Toeval??

Men kan proberen de puzzel opnieuw op te lossen maar nu met een andere rij symbolen bovenaan als uitgangspositie. Soms is er een domme reden waarom het niet lukt; als er alleen maar oranje symbolen staan blijkt de beschikbare hoeveelheid "oninteressante" oranje plankjes ontoereikend te zijn om het gehele bord vol te hangen; jammer. Begint men met een rij bestaande alleen maar uit donkerblauwe stukken dan zijn er evenzeer te weinig stukken en bovendien loopt de actieve "diagonaal" na te zijn teruggekaats door tot op de onderrand: het bord is klaarblijkelijk niet hoog genoeg. Als de kleur wijnrood boven ontbreekt dan loopt de actieve diagonaal rechts het bord uit zonder dat er iets gebeurt. Om iets interssants te zien moet er klaarblijkelijk aan zekere beperkingen worden voldaan. Als ontwerper van de puzzel kan ik U die ook melden: niet meer dan drie optredens van een kleur in de bovenrand, precies een keer de kleur wijnrood gebruiken (bij voorkeur aan de rechterkant) en niet eindigen met drie optredens van de kleur donkerblauw.

Rekening houdende met deze beperkingen blijken de onderstaande beginrijen realiseerbaar te zijn: 00101, 00110, 01001, 01010, 01100, 10001, 10010, 10100, 11000, 01011, 01101, 01110, 11001, 11010, 11100, 10011, 10101 10110, en 11100. Daarnaast zijn er een aantal "ingekorte" patronen waarbij de kleur wijnrood in de vijfde of een nog vroegere kolom optreedt (waarbij opgemerkt moet worden dat rechts van de kleur wijnrood helemaal niets interessants gebeurt: de boven gekozen kleur wordt herhaald tot de onderrij bereikt wordt). Alle patronen van lengte 4 blijken toelaatbaar te zijn met uitzondering van de rijen 1111 en 0000. Maken we de rij nog korter dan blijken problemen op te treden als de kleur oranje in het geheel niet wordt gebruikt: de teruggekaatste diagonaal loopt nu links van het bord af. Rekening houdende met deze beperkingen blijkt toch een redelijk aantal begincondities te kunnen worden geprobeerd; mogelijk zijn de begincondities behorende bij de getallen 5,7,9,10,12,17,18,20,24,11,13,14,25,26,28,21,22 en 28 (waarbij is afgezien van de "ingekorte" voorbeelden). Ik laat het aan de lezer c.q. toeschouwer over om al deze mogelijkheden te proberen en vast te stellen dat inderdaat na het maken van de puzzel aan de onderkant een rij onstaat waarvan de waarde (na ontcijfering als binair getal) één hoger is dan de waarde waarmee bovenaan is begonnen. Toeval???

In het vervolg van deze brochure zal ik toelichten wat de wiskundige structuur is aan de puzzel ten grondslag ligt. Hiertoe zullen wij nader ingaan op het binair rekenen en tellen en aangeven hoe "mechanisch" dat kan verlopen. Vervolgens zullen wij een geschikt machinemodel beschrijven. Tenslotte laten wij zien hoe de puzzel gezien kan worden als het resultaat van een vertaling van dat machinemodel. Deze constructie laat zien dat er van toeval geen sprake is.

Binair rekenen en tellen

Wat betekent het om bij een binair getal één op te tellen? Optellen van binaire getallen doen wij in principe op de zelfde wijze als het optellen van decimale getallen zoals wij dat op de kleuter- of lagere school leren: zet de twee getallen onder elkaar en tel cijfergewijs op, rekening houdende met het van links naar rechts propageren van eventuele "carries".

In het binaire geval baseren wij ons op een opteltabel met de vier regels: $0+0=0, 0+1=1+0=1, 1+1=10$; alleen het laatste geval genereert een carry. Een binair getal met één vermeerderen is het zelfde als het bij een getal optellen van een getal dat bestaat uit een rij nullen eindigende op een één. Het effect daarvan in de schoolmethode laat zich als volgt formuleren: neem het blok enen dat rechts in het getal staat, en vervang dat door een blok nullen van dezelfde lengte, en vervang de nul ervoor door een één. Als deze nul niet bestaat vervang dan de spatie voor het getal door een één. Voorbeelden ter illustratie:

- $11+1=12$; $01011+00001=01100$
- $15+1=16$; $01111+00001=10000$
- $31+1=32$; $11111+00001=100000$

Merk op dat in het laatste voorbeeld de binaire schrijfwijze van het getal echt één symbool langer is geworden.

In feite hebben we het probleem van het rekenen op deze wijze gereduceerd tot een vrijwel mechanische vorm van symbolen manipulatie. Een domme machine zou dat ook moeten kunnen.

Het Turingmachine model

Het model bij uitstek van een domme machine die geschikt is voor dit soort bewerkingen is de in 1936 geintroduceerde *Turingmachine*. De Britse wiskundige en logicus Alan Turing introduceerde deze machine als model van de menselijke wiskundige die aan het rekenen is en daarbij tussenresultaten berekent, opschrijft, onthoudt en later weer kan terugvinden en aldus kan gebruiken bij verdere berekeningen. Het model laat zich op een nette wiskundige wijze beschrijven zodat het zich met wiskundige hulpmiddelen laat onderzoeken.

Het model laat zich het beste uitleggen aan de hand van figuur 2. De machine bestaat uit een eindige automaat die via een zogenaamde *leeskop* wordt gekoppeld aan een opslagmedium. Dit opslagmedium heeft de structuur van een tweezijdig oneindige band bestaande uit velden waarop een teken uit een eindige verzameling Σ van *bandsymbolen* kan worden geschreven. Er is een speciaal bandsymbool *blanco* (geschreven $\Box$) dat op al die plaatsen van de band wordt aangetroffen waarop aan het begin van een berekening van de machine niets anders geschreven staat en waar de machine tijdens de berekening tot nu toe nooit geweest is. De controle van de berekening berust bij de eindige automaat. Op ieder tijdstip bevindt dit gedeelte van de machine zich in een bepaalde toestand. Deze toestanden komen uit een eindige verzameling van toestanden K.

Het programma voor de berekening P wordt beschreven door een verzameling vijftallen die element zijn van de productverzameling $K \times \Sigma \times \Sigma \times (K \cup \{0\}) \times \{L, 0, R\}$. De betekenis van het optreden van het vijftal (q, s, s', q', m) in dit programma is dat de volgende actie van de machine legaal is: *indien de machine in toestand q verkeert en via de leeskop een symbool s leest, mag zij dat symbool overschrijven met het symbool s', overgaan in toestand q' en de leeskop verplaatsen over de afstand m.* Hierbij wil $m = L$ ($m = R$) zeggen: *ga één veld naar links (rechts) over de band*, terwijl $m = 0$ betekent: *blijf staan op je huidige positie*. De waarde $q' = 0$ in een instructie betekent zoveel als: *er bestaat geen volgende toestand; na deze instructie is de berekening afgelopen.*

Merk op dat in deze beschrijving het goed denkbaar is dat voor een gegeven combinatie van een bandsymbool s en een toestand q er geen enkel vijftal in het programma is dat begint met $(q, s, ...)$. In dit geval is er geen toepasbare instructie en eindigt de berekening eveneens. Men beschouwt een dergelijke berekening als mislukt. Het is ook denkbaar dat er meerdere vijftallen in het programma zijn opgenomen die toepasbaar zijn. Dit lijkt op de situatie van een ouderwets imperatief programma met een 'goto' opdracht waarbij het label waarnaartoe gesprongen wordt op meerdere plaatsen in het programma optreedt. Men beschouwt dit in het algemeen als een programmeerfout, maar binnen ons model is het toegestaan. Het effect is dat een van de mogelijke instructies wordt uitgevoerd en de anderen worden genegeerd. Later in de berekening, als wij andermaal voor dezelfde keuze worden gesteld wordt misschien een andere keuze gemaakt. Men noemt dit gedrag van de machine *Nondeterministisch*. De berekening is niet volledig bepaald door de begintoestand maar er kunnen onderweg redelijk arbitraire keuzen worden gemaakt. Als geen van deze ongewone omstandigheden zich voordoet noemt men het programma en de bijbehorende machine *Deterministisch*.

Om te komen tot een beschrijving van de rekenwijze van een Turingmachine hebben wij nog enkele begrippen nodig. Een *configuratie* van de machine is een beschrijving van de toestand waarin de machine en het opslagmedium zich bevinden. Dat wil zeggen: de toestand van de machine, de bandinhoud (voorzover niet bestaande uit blanco symbolen), en de positie van de leeskop. Een elegante codering van een configuratie heeft de vorm: $\$s_0, s_1, ..., (q, s_j), ..., s_n\$$. In deze configuratie is het non-blanco gedeelte van de band de beschreven met de tekenrij $s_0, s_1, ..., s_j, ..., s_n$, de toestand van de eindige automaat is q en de leeskop is staat boven symbool s_j. Het is vrij eenvoudig uit te schrijven wat het effect is van het uitvoeren van een instructie op een configuratie van deze vorm. Bijvoorbeeld als (q, s, s', q', R) de uitgevoerde instructie is (hetgeen veronderstelt dat $s = s_j$) dan is de resulterende configuratie gelijk aan $\$s_0, s_1, ..., s', (q', s_{j+1}), ..., s_n\$$. In het grensgeval dat $j = n$ wordt de configuratie daartoe uitgebreid met een optreden van het blancosymbool (dwz. $s_{n+1} = \Box$). Hetzelfde gebeurt als de kop links de configuratie uitloopt. Als $q' = 0$ verdwijnt de toestand uit de configuratie en houden wij een rij kale bandsymbolen over.

Een *startconfiguratie* is een configuratie van de vorm $\$(q_0, s_0), s_1, ..., s_n\$$. Hierbij is q_0 een speciaal aangewezen *starttoestand*. Een *berekening* is een reeks configuraties beginnende in een startconfiguratie en eindigende in een configuratie waarin geen instructie kan worden uitgevoerd, hetzijn doordat de toestand is verdwenen (hetgeen de nette manier van eindigen is), hetzij doordat er geen toepasbare instructie in het programma is opgenomen (hetgeen de mislukte vorm van terminatie is). De *invoer* van de berekening wordt gevormd door het aaneengesloten blok van non-blanco symbolen in de beginconfiguratie. Het *resultaat* van de berekening wordt gevormd door de rij non-blanco symbolen die wij na afloop op de band aantreffen. Het is verder een zaak van de gebruiker van de machine om uit te leggen hoe zij dit resultaat wenst te interpreteren.

Ik wil dit model illustreren aan de hand van de bewerkingen nodig om een binair getal met één te vermeerderen. Als bandsymbolen gebruiken wij de beide binaire cijfers 0 en 1 en uiteraard het blanco symbool $\Box$. Er zijn twee toestanden: de startoestand q waarin het einde van het getal wordt gezocht (dwz. het meest linker blanco symbool rechts van de startpositie), en de toestand r waarin het rechter aaneengesloten blok enen wordt overschreven door nullen, totdat een ander symbool ervoor (0 of $\Box$) kan worden overschreven door een één, waarna de berekening eindigt. Een deterministisch programma

dat deze berekening uitvoert bestaat uit de volgende zes instructies:
$(q,0,0,q,R),(q,1,1,q,R),(q,\square,\square,r,L),(r,1,0,r,L),(r,0,1,0,0),(r,\square,1,0,0)$.

Een voorbeeld van een volledige berekening van deze machine is gegeven in figuur 3. In dit voorbeeld hebben wij de opeenvolgende configuraties symboolsgewijs onder elkaar geschreven. Van boven naar beneden ziet U de tijd verlopen, terwijl van links naar rechts de verschillende velden van de band worden beschreven. Dit diagram heet wel het *Tijd-Geheugen diagram* van de berekening. De berekening die in dit voorbeeld uitgevoerd wordt is $11+1=12$. De structurele overeenstemming met het patroon van de oplossing van de puzzel in figuur 1 is evident.

De Universaliteit van de Turingmachine

Alvorens in te gaan op het verband tussen de bovenbeschreven successor Turing machine en onze puzzel is het zinvol iets meer te vertellen over de kracht van het Turingmachine model. Op het eerste gezicht lijkt het maar een moeizaam soort machine; een beetje zakjapanner kan meer en doet dat veel en veel sneller. Toch kan men aantonen dat het Turingmachine model *Universeel* is. Alles wat in principe op enige denkbare vorm van een computer kan worden uitgerekend laat zich in principe ook uitrekenen op een Turingmachine in de vorm als beschreven. De grondoorzaak hiervan is gelegen in het feit dat, net als bij echte programma's, verschillende elementaire bewerkingen in dit model zich laten samenstellen tot complexe programma's waarbij uitgebreide berekeningen blijken te bestaan uit herhalingen van volstrekt elementaire acties. Volgens de beroemde Amsterdamse wiskundige en logicus L.E.J. Brouwer vormt het tellen de grondslag van de wiskunde; wij hebben vastgesteld dat wij met een Turingmachine kunnen tellen, dus welk verder bewijs zouden wij nog nodig hebben??

Deze stellingname is in de logica bekend onder de naam *These van Church - Turing*. Het is geen “bewijsbare” stelling in de traditionele wiskundige zin, maar het is een ervaringsfeit waarvan de geldigheid keer op keer bevestigd is toen alternatieve formele beschrijvingen van het begrip berekenbaarheid gelijkwaardig bleken te zijn met het Turingmachine model. Sterker nog, in die voorbeelden dat de gelijkwaardigheid niet van toepassing lijkt te zijn ontleent men daaraan een argument om te zoeken naar een “fout” in het alternatieve model.

Er geldt echter nog meer. Tegenwoordig pleegt men behalve naar de zuivere notie van berekenbaarheid ook rekening te houden met het feit dat rekenen iets kost. Berekeningen vragen tijd en gebruiken een vorm van geheugen. In ons model wordt de tijd gemeten door het aantal configuraties in een volledige berekening terwijl het geheugen wordt gemeten door de maximale lengte van een configuratie (dwz. het maximale aantal non-blanco en/of bezochte bandcellen) in een berekening. En nu kan men zich de vraag stellen of het Turingmachine model Universeel blijft indien men ook rekening houdt met de hoeveelheid tijd en geheugen die berekeningen kosten. Met andere woorden: het zou wel eens zo kunnen zijn dat het model wezenlijk inefficient is en dat een alternatief model wezenlijk sneller en/of zuiniger de verschillende berekeningen kan uitvoeren.

Ook in dit opzicht blijken wij echter met een redelijk model van doen te hebben. Als wij bereid zijn de snelheden van de verschillende modellen met een zekere tolerantie met elkaar te vergelijken dan is het model echt een voorbeeld van de klasse van standaard (sequentiele) machinemodellen die in de wereld zijn ingeburgerd. De hierbij gehanteerde tolerantie bestaat uit een onderlinge simulatie met een niet meer dan polynomiale vertraging in de rekentijd en een constante factor in het geheugengebruik. Concreet geformuleerd: als U problemen waarvan de invoerbeschrijving n bytes in het geheugen van uw PC innemen op deze PC kunt oplossen in tijd $T(n)$ en geheugen $S(n)$ zal een passende vertaling van deze problemen die opgelost kan worden op een geschikte Turingmachine aan rekentijd $C \times T(n)^3$ en geheugen $C' \times S(n)$ kosten voor zekere constanten C en C'.

In de praktijk geeft deze vertraging voldoende motief om bij de plaatselijke dozenschuiver bij voorkeur geen Turingmachine te bestellen, maar vanuit het gezichtspunt van de grote fundamentele vragen over de relatie tussen tijd en geheugen, en de efficiente oplosbaarheid van de combinatorische zoekproblemen waarmee de samenlevening dagelijks wordt geconfronteerd is een dergelijke kubische overhead acceptabel. In deze wereld gaat het namelijk primair over het onderscheid tussen problemen die in polynomiaal begrensde tijd oplosbaar zijn en problemen die de indruk wekken slechts te kunnen worden opgelost met brute zoekmethoden die exponentiele rekentijd vragen. Exponentiele functies groeien wezenlijk sneller dan polynomen en aangezien een polynoom tot de derde macht verheven nog steeds een polynoom is verandert een kubische vertraging niets aan dit verschil.

De vertaling van Turingmachines naar puzzels

We zijn thans op het punt gekomen dat ik de relatie tussen de puzzel en de behandelde successor machine kan uitleggen. Het zal blijken dat de puzzel uit de machine is verkregen door toepassing van een standaardconstructie die in principe toepasbaar is op iedere Turingmachine.

Voor de goede orde herhaal ik wat de structuur van onze puzzel is. Er is een vaste collectie tegeltypes. Ieder tegeltype bestaat uit een vierkant met vier gekleurde driehoeken. Van ieder tegeltype is (in tegenstelling tot ons demonstratiemodel) een onbeperkte voorraad tegels aanwezig. Tegels mogen niet worden gedraaid en gespiegeld. Het bord bestaat uit een te betegelen gebied, waar op een aantal posities reeds een kleur gevraagd wordt door middel van een randconditie.

Doel van de vertaling is om gegeven een Turingmachine programma een puzzel te ontwerpen zo dat de oplossingen van de puzzel corresponderen met tijd-geheugen diagrammen van deze machine. Uiteraard staat een tijd-geheugen diagram vol met symbolen terwijl wij bij de puzzel gebruik maken van kleuren. Kennelijk zullen wij kleuren gebuiken als code voor een symbool (of eventueel een paar symbolen). In het bijzonder zal de reeks kleuren langs een horizontale lijn tussen twee rijen puzzelstukken de codering vormen van een configuratie in de berekening. Naast elkaar geplaatste tegels delen in het algemeen een niet zichtbaar gemaakte kleur ("wit" in deze tekst, geel in het demonstratiemodel) die afwezigheid van actie aanduidt.

Voor deze vertaling hebben wij in feite drie soorten stukken nodig. De eerste soort bestaat uit "inactieve" stukken die er voor dienen aan te geven dat op een plaats in de configuratie waar gedurende een rekenstap niets gebeurt het gegeven bandsymbool blijft staan. In figuur 4 zijn de voor dit doel nodige stukken aangeduid als *doorgeeftegels*. De zijkanten van deze tegels hebben de afwezige kleur wit.

De tweede soort tegel dien ervoor te zorgen dat een "inactief" bandsymbool in de volgende configuratie actief kan worden doordat de leeskop van links of rechts komend naar de bewuste cel beweegt. In deze *opvangtegels* is een zijkant voorzien van een speciale kleur corresponderende met de toestand waarin de machine naar deze cel toebeweegt. De kleur op de bodem correspondeert met het paar (toestand, symbool).

De constructie van deze tegeltypes geeft onmiddelijk aanleiding tot een complicatie. Het lijkt immers legaal om twee opvangtegels naast elkaar te plaatsen als de rechter tegel een toestand opvangt die van links komt en de linker tegel een van rechts komende toestand opvangt. Dit verstoort onze interpretatie aanzienlijk. We beginnen met een configuratie waarin twee inactieve symbolen staan en een stap later heeft zich op de plaats een "phantoom" paar koppen gevormd. Zoiets is op subatomaire schaal mogelijk in de quantumphysica, maar dat kunnen wij bij onze vertaling allerminst gebruiken.

De remedie is echter eenvoudig. In ons voorbeeld programma hebben wij reeds gezien dat de bewegingsrichting in een toestand eenduidig wordt bepaald door de toestand. Het is gemakkelijk (door zo nodig toestanden te verdubbelen) er voor te zorgen dat de machine bij het overgaan naar een toestand q hetzij naar rechts, hetzij naar links beweegt maar zeker niet in beide richtingen. Dit heeft tot gevolg dat van beide soorten opvangtegels er slechts één in onze collectie tegeltypes hoeft te worden opgenomen, waarmee de constructie van phantoomkoppen wordt geblokkeerd.

Het derde type tegels correspondeert met de instructies in het programma. De bovenkant van dit type heeft een kleur corresponderende met een paar (toestand, symbool). Voor iedere instructie in het programma die voor dit paar toepasbaar is is een tegeltype beschikbaar. Als een echte beweging wordt uitgevoerd verschijnt de kleur behorende bij het nieuwe symbool op de bodem en de nieuwe toestand verschijnt aan de juiste zijkant; als er geen beweging is verschijnt op de bodem een nieuwe kleur die een paar codeert. Als de toestand verdwijnt ontstaat de kleur van een inactief symbool beneden en zijn beide zijkanten wit.

Hiermee is in feite verklaard hoe onze puzzel is geconstrueerd. Kijken wij terug naar figuur 1 dan is aldaar duidelijk te zien wat het resultaat is van onze vertaling voor het geval van de successormachine. Tevens is de oplossing afgedrukt die correspondeert met de eerder gegeven berekening. Om vandaar te komen tot het demonstratiemodel is nog slechts een kwestie van het koppelen van kleuren met arceringstypes. Hierbij zal het de oplettende lezer opvallen dat tussen de codering van berekeningen in deze toelichting en in de puzzel een onbeduidend doch wellicht verwarrend verschil is geslopen: de puzzel gebruikt een beginconfiguratie gecodeerd als: $\$q_0, s_0, s_1, ..., s_n\$$. De eerste rekenstap wordt daardoor besteed aan de vorming van het eerste toestand-symbool paar. Beide conventies zijn acceptabel: het is gewoon een kwestie van smaak, en daarbij is Uw auteur klaarblijkelijk niet volledig consistent. In de tekst volg ik de standaard literatuur, en het is nu eenmaal technisch niet goed mogelijk de vorm van het demonstratiemodel daaraan aan te passen....

Het voordeel van de gekozen vorm van de vertaling is dat in onze puzzel er een duidelijke afbakening ontstaat tussen informatie afkomstig van het programma en informatie gegeven door de invoer van de berekening. Het programma bepaalt eenduidig de collectie van beschikbare tegeltypes. De invoer wordt volledig bepaald door de randconditie: invoertekst langs de bovenrand, in combinatie met een witte linker en rechter rand; slechts het verticale strookje links boven is voorzien van de kleur behorende bij de starttoestand. De benedenrand is niet ingevuld want wij willen niet van te voren verklappen wat de eindconfiguratie zal zijn.

De kern van de poedel: de NP-volledigheidstheorie

Voor de goede orde wil ik ter aflsuiting van deze inleiding de fundamentele stelling vermelden die in feite via deze constructie wordt bewezen. Er bestaat een klasse van problemen die allemaal de volgende vorm hebben: er is een verzameling W gegeven van woorden en men moet van een gegeven woord x vaststellen of dat woord x al dan niet tot W behoort. Om vast te stellen dat x tot W behoort kan men volstaan met het vinden van een *getuige* y, waarbij het erg eenvoudig is om gegeven een candidaat getuige y en het woord x te bepalen of y een goede getuige is. Als x niet tot W behoort bestaat er geen getuige. Verder is het zo dat de getuige niet veel groter is dan de invoer x zelf: er moet sprake zijn van een polynomiale begrenzing in de lengte. Uiteraard zijn deze problemen oplosbaar: probeer alle mogelijke getuigen en behoudt de goede. Helaas kost dat wel exponentieel veel werk.

De problemen van deze vorm vormen een collectie die in de literatuur bekend staat onder de naam NP. Een berucht voorbeeld van een NP-probleem is het *handelsreizingers probleem*: bepaal of een handelsreiziger een gegeven collectie steden kan bezoeken zonder ooit enige stad anders dan de uitgansgstad meer dan eens te bezoeken en daarbij niet langer dan een gegeven afstand onderweg te zijn. Een getuige is hier een rondrit in combinatie met de berekening die aantoont dat de maximale afstand niet wordt overschreden. Helaas zijn er voor n steden $n!$ mogelijke rondritten, en het vinden van de goede getuige door ze allemaal te proberen zal daardoor exponentiele rekentijd vergen.

In het concrete voorbeeld van betegelingen met een vaste collectie tegeltypes is W de verzameling randcondities waarvoor de betegeling bestaat. Gegeven een dergelijke randconditie x vormt een volledige betegeling een getuige.

De onbesproken stelling spreekt uit dat onze betegelingsproblemen in zekere zin de meest algemene problemen van deze vorm zijn. Gegeven een dergelijk probleem kunnen wij ons voorstellen dat een nondeterministische Turingmachine een getuige bij elkaar gokt en vervolgens controleert dat de gegokte getuige voldoet. Op grond van de begrenzingen en de universaliteit van het Turingmachine model kan een Turingmachine dit in polynomiaal begrensde rekentijd uitvoeren. Dank zij onze constructie kunnen wij het programma van deze machine weer vertalen in een collectie tegeltypes en de bijbehorende invoer in een passende randconditie, waarna de vraag of de originele invoer x tot W behoort is vertaald in de oplosbaarheid van een geconstrueerde betegelingspuzzel.

Het feit dat op deze wijze geinterpreteerd de betegelingsproblemen de lastigste probleem van deze klasse vormen staat in de literatuur bekend als de NP-volledigheid van het betegelingsprobleem. Gevolg van deze eigenschap is dat een eventuele eficiente deterministische oplossingsmethode voor het betegelingsprobleem zich terug laat vertalen tot een efficiente oplossingsmethode voor andere NP-problemen, en dat is iets waar de wereld veel aan zou hebben. Nodeloos te vermelden dat wij een dergelijke methode niet kennen; in feite gelooft het merendeel van de informatici dat een dergelijke methode niet bestaat maar bewijzen kan men dat niet, althans niet met gebruik van thans bekende bewijsmethoden.

Achtergronden en literatuur

De betegelingspuzzel die in deze toelichting wordt besproken is voor het eerst in haar onbegrensde vorm behandeld in artikelen van Hao Wang [14, 15]. De hier gegeven vertaling van

Turingmachine programma's in betegelingsproblemen is afkomstig van R.M. Robinson [9]. Bij deze beide auteurs is het doel van de invoering het bewijzen van onbeslisbaarheidsresultaten. De voornaamste complicatie in deze artikelen is gelegen in het aantonen dat ook de versie van het probleem waar geen enkele randconditie gegeven is onbeslisbaar blijft. Hiertoe dient men een collectie tegeltypen te construeren die afdwingt dat binnen een betegeling ergens een berekening wordt gesimuleerd, zonder dat dat expliciet via een randconditie mag worden afgedwongen. De eerste die er in is geslaagd een dergelijke constructie door te voeren was R. Berger [1]; naderhand is de constructie vereenvoudigd door Robinson.

Het model van de Turingmachine is ingevoerd in 1936 door A.M. Turing [11]. Voor een nadere uitleg dat dit model universeel is in de zin van de complexiteitstheorie verwijs ik naar een inleiding over machine modellen van mijn hand [13].

Het vroegste gebruik van betegelingsproblemen in de complexiteitstheorie is te vinden bij L. Levin [6], de Soviet onderzoeker die onafhankelijk van Cook en Karp [2, 5] de notie van NP-volledigheid heeft ingevoerd en onderzocht. In de westelijke literatuur treffen wij haar voor het eerst aan bij H. Lewis [7]. De ontwikkeling van de NP-volledigheidstheorie op basis van een "meesterreductie" naar begrensde betegelingsproblemen is terug te vinden in het leerboek van Lewis en Papdimitriou [8]. Dat langs deze route de NP-volledigheidstheorie aanmerkelijk verder kan worden ontwikkeld dan door deze auteurs gedaan kan men nalezen in de (helaas vrijwel onvindbare) artikelen van van Emde Boas en van M. Savelsberg & P. van Emde Boas [12, 10].

Populaire inleidingen tot de NP-volledigheistheorie zijn te vinden in het (inmiddels verouderde maar toch nog steeds waardevolle) boek van Garey & Johnson [3] en in het beneden vermelde boek van D. Harel [4].

Begrensde betegelingen worden sinds een tiental jaren gebruikt bij het inleidende onderwijs in de complexiteitstheorie aan de Faculteit Wiskunde en Informatica van de Universiteit van Amsterdam. Het idee om het voorbeeld van de successor functie te gebruiken voor een demonstratiemodel voor gebruik op practicum dagen en wetenschapsmarkten is in 1992 opgekomen en door de faculteit gerealiseerd. Het model van de puzzel is ontworpen door de auteur en uitgevoerd door mw. A. Houtkoper-Visser, met medewerking van dr. E.H. Dooijes.

Bij deze brochure is gevoegd een zevental bladen met de verschillende tegeltypes in een zwart-wit uitvoering. Met het gegeven materiaal (twee keer vel A en vel B en drie keer vel C) kunt U zelf een exemplaar van de puzzel construeren.

Referenties

[1] R. Berger, *The undecidability of the domino problem*, Mem. Amer. math. Soc. 66 (1966).

[2] S.A. Cook, *The complexity of theorem proving procedures*, Proc. ACM STOC 3 (1971) 151–158.

[3] M.R. Garey & D.S. Johnson, *Computability and intractability; A guide to the theory of NP-completeness*, W.H. Freeman and Company, 1979.

[4] D. Harel, *Algorithmics; the spirit of computing*, 2nd edition, Addison Wesley, 1992.

[5] R.M. Karp, *Reducibility among combinatorial problems*, R.E. Miller & J.W. Thatcher (eds.), Complexity of computer computations, Plenum, New York 1972, pp. 85–103.

[6] L.A. Levin, *Universal sequential search problems*, Problems of information transmission 9 (1973), 265–266.

[7] H.R. Lewis, *Complexity of solvable classes of the decision problem for predicate calculus*, Proc IEEE FOCS 19 (1978), 35–47.

[8] H.R. Lewis & C.H. Papadimitriou, *Elements of the theory of computation*, Prentice Hall, 1981.

[9] R.M. Robinson, *Undecidability and non periodicity for tilings of the plane*, Invent. Math. 12 (1971) 177-209.

[10] M.W.P. Savelsberg & P. van Emde Boas, *BOUNDED TILING, an alternative to SATISFIABILITY?*, G. Wechsung (ed.), Proc. 2nd Frege Memorial Conference, Schwerin (GDR), Sep 1984, Akademie Verlag, Mathematische Forschung 20, pp. 401–407, 1984. preprint: rep CWI-OS-R8405.

[11] A.M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. (2) 42 (1936) 230–265; correction, ibidem 43 (1937) 544–546.

[12] Peter van Emde Boas, *Dominoes are forever*, L. Priese (ed.), Proc. 1st GTI workshop, Paderborn, Oct 1982, Rheie Theoretische Informatik UGH Paderborn, 13 pp. –.–. Preprint: Rep. SFW-UvA-83-03.

[13] Peter van Emde Boas, *Machine models and simulations*, J. van Leeuwen (ed.), Handbook of Theoretical Computer Science, Vol. 1, Algorithms and Complexity, Elsevier Science publ. 1990, pp. 3–66.

[14] Hao Wang, *Proving theorems by pattern recognition - II*, Bell System Tech. Journal, 60 (1961) 1–41.

[15] Hao Wang, *Games, Logic and Computers*, Sci. American. 213 (Nov. 1965), 98–106.