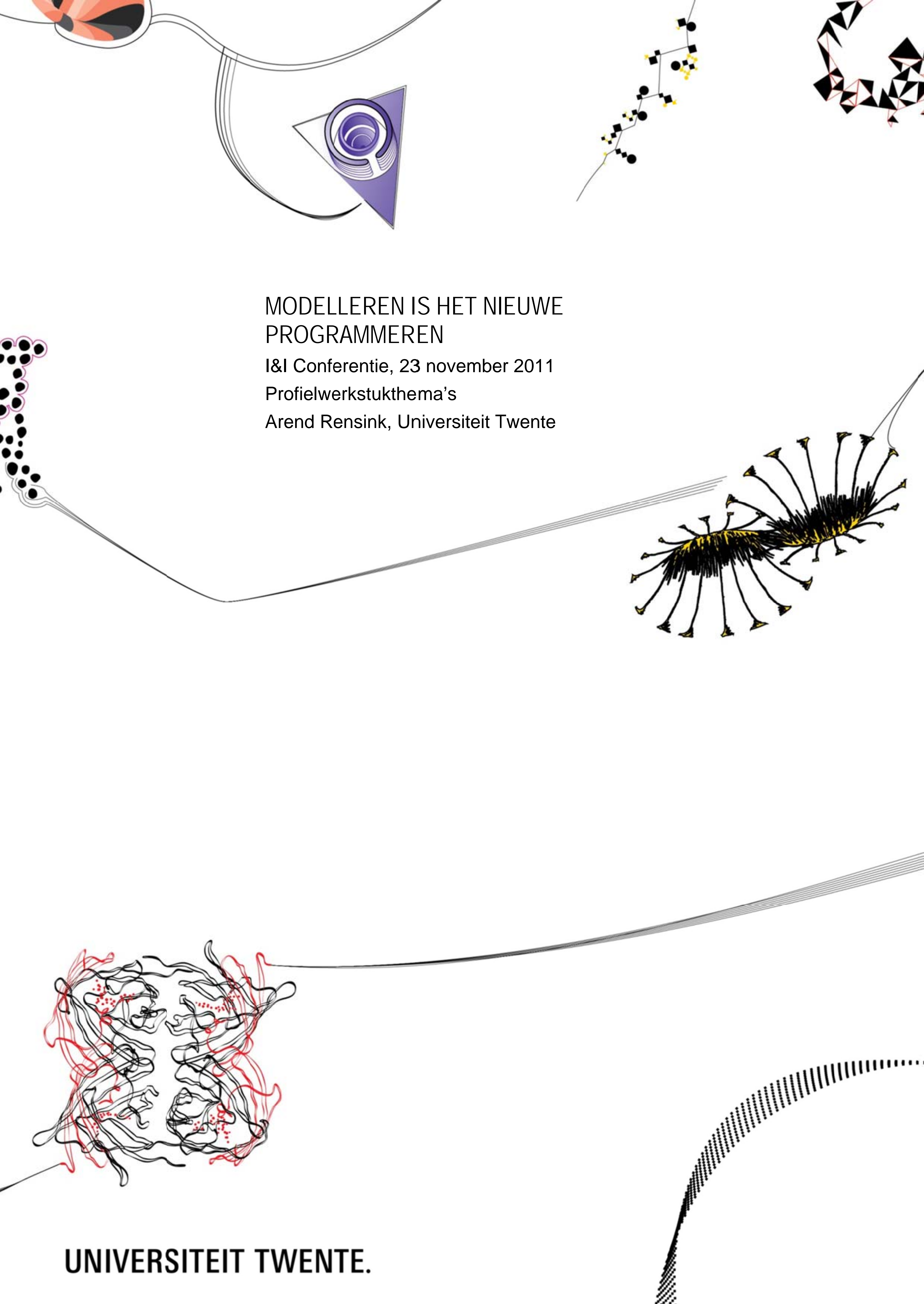


MODELLEREN IS HET NIEUWE PROGRAMMEREN

I&I Conferentie, 23 november 2011

Profielwerkstukthema's

Arend Rensink, Universiteit Twente

UNIVERSITEIT TWENTE.

Inhoudsopgave

Hersenkrakers (Graaftransformatie) A-1

Bellen zonder zorgen (Model checking) B-1

Gelijk oversteken (Roblox) C-1

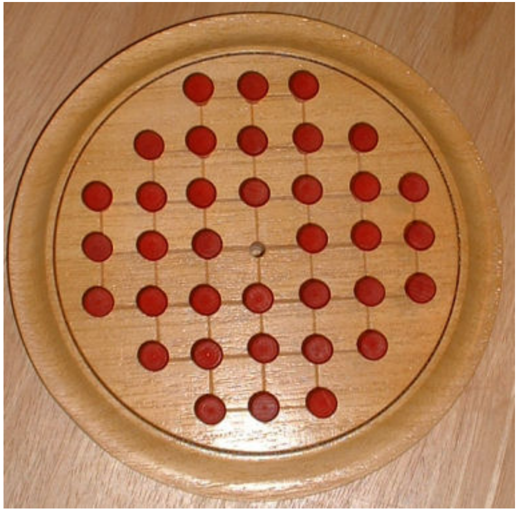
Zie ook: http://www.twenteacademy.nl/ (PWS hulp)

Hersenkrakers: De computer lost het voor je op

(Profielwerkstukthema gebaseerd op graaftransformaties)

Hoe zet je acht koninginnen op een schaakbord, zodat ze elkaar niet kunnen slaan? Of hoe zorg je dat je maar één pion in het midden overhoudt op het solitaire bord (Figuur 1, bij solitaire spring je met een pion over één andere heen naar een leeg vakje, en verwijder je de pion waar je overheen sprong).

Heb je altijd al het antwoord op dit soort puzzels willen weten, maar heb je nooit zin om er over na te denken, dan is er maar één oplossing: laat de computer het voor je doen.



Figuur 1: Het speelbord van Solitaire

1 Wat ga je doen?

Voor dit profielwerkstuk ga je je verdiepen in graaftransformatie. Dit is een techniek waarbij systemen, zoals de hierboven genoemde puzzels maar ook bijvoorbeeld ingewikkelde computerprogramma's, worden beschreven door een *graaf*. Met behulp van regels die we *transformaties* noemen kunnen we dan interessante eigenschappen van de graaf afleiden. Deze eigenschappen zeggen dan ook iets of het systeem of computerprogramma waar de graaf voor gemaakt is, en we kunnen die eigenschappen gebruiken om te kijken of dat systeem goed werkt, of om naar een bepaalde toestand van dat systeem te zoeken.

Omdat graaftransformaties van computerprogramma's erg ingewikkeld kunnen zijn, gaan we die niet voor dit profielwerkstuk gebruiken. In plaats daarvan gebruiken we simpele puzzels, en laten we in §4 zien hoe je met graaftransformaties zo'n puzzel kan oplossen.

In §2 vind je een korte introductie over grafen en graaftransformaties. Vervolgens staan in §3 een aantal suggesties voor onderzoeksvragen die je zou kunnen gebruiken voor je profielwerkstuk. Tot slot staat in §4 een klein voorbeeld uitgewerkt.

Aan het eind van de opdracht kan je een woordenlijst vinden met belangrijke en moeilijke begrippen en een korte uitleg daarvan vinden. Als je toch ergens niet uitkomt, kan je altijd hulp vragen bij de universiteit. Stuur daarvoor een mailtje naar Arend Rensink (rensink@cs.utwente.nl).

Figuur 2: Een voorbeeldgraaf: Het spoorwegennet

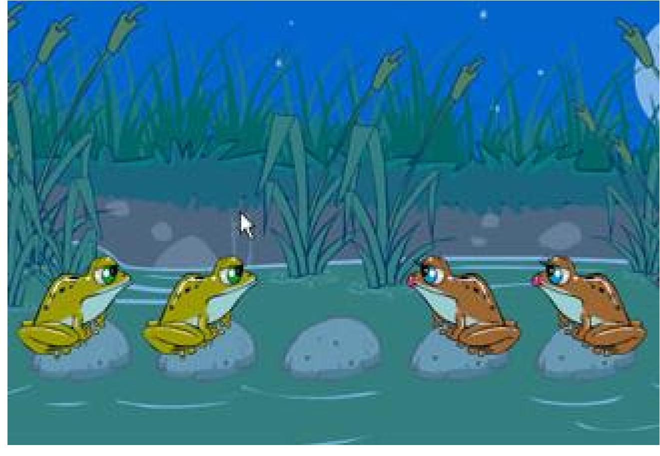
2 Wat zijn Graaftransformaties

2.1 Grafen

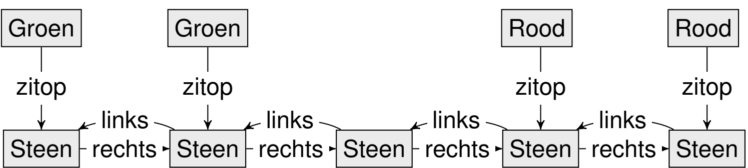
Een graaf in de wiskunde is een verzameling knopen (Engels: nodes). Om aan te geven dat er een bepaald verband is tussen twee knopen kunnen we een pijl (Engels: edge) trekken tussen deze knopen. De knopen kunnen nul, één of meer labels hebben; pijlen hebben altijd precies één label.

Als dit lasting klinkt, kijk dan bijvoorbeeld naar Figuur 2: dit is een spoorwegenkaart van de NS-website. De kaart is eigenlijk een graaf waarbij de knopen de treinstations en de pijlen de rails tussen de treinstations voorstellen. Behalve spoorwegen kunnen we met grafen nog veel meer situaties uit de praktijk op een simpele manier beschrijven.

Ter illustratie gebruiken we in de rest van deze toelichting de *kikkerpuzzel*. De beginsituatie van de kikkerpuzzel is zoals in Figuur 3. Er zijn vijf stenen: op de meest linker twee staat een groene kikker, en op de meest rechter twee staat een rode kikker. Het doel van de puzzel is om alle kikkers met alleen de volgende zetten naar de overkant te krijgen:



Figuur 3: Voorbeeldpuzzel: de kikkers



Figuur 4: Voorbeeldpuzzel: de graaf

- Als de steen rechts van een groene kikker leeg is, mag de groene kikker een stap naar rechts doen.
- Als op de steen rechts van een groene kikker een rode kikker zit, dan kan de groene kikker daar overheen springen. Dit mag alleen als de steen waarop de groene kikker land leeg is, en de groene kikker mag maar over één rode kikker tegelijk heen springen.
- Als de steen links van een rode kikker leeg is, mag de rode kikker een stap naar links doen.
- Als op de steen links van een rode kikker een groene kikker zit, dan kan de rode kikker daar overheen springen. Dit mag alleen als de steen waarop de rode kikker land leeg is, en de rode kikker mag maar over één groene kikker tegelijk heen springen.

Als we deze puzzel in een graaf willen beschrijven zullen we negen knopen nodig hebben; vijf knopen die de stenen voorstellen en vier knopen die de kikkers voorstellen. Om aan te geven op welke volgorde de stenen liggen, tekenen we links- en rechtspijlen tussen twee stenen die naast elkaar liggen. Tot slot tekenen we een aantal 'zitop' pijlen tussen de kikkers en de stenen om aan te geven op welke steen de kikkers zitten. De graaf ziet er dan uit zoals in Figuur 4

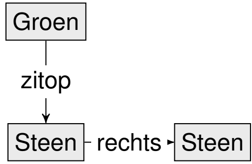
2.2 Transformaties

Als we eenmaal een graaf hebben en we bepaalde eigenschappen van die graaf willen onderzoeken, kunnen we daarvoor graaftransformaties gebruiken. Een graaftransformatie is niets anders dan een aanpassing van de graaf naar een nieuwe graaf. Dit kan bijvoorbeeld door het toevoegen van nieuwe knopen aan de graaf, of door het verplaatsen van pijlen. Door het na elkaar uitvoeren van verschillende transformaties kunnen we de graaf zo veranderen dat we gemakkelijk belangrijke eigenschappen kunnen aflezen.

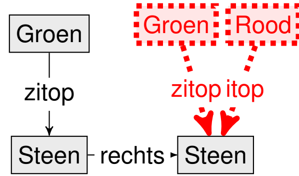
Denk bijvoorbeeld weer aan de voorbeeldgraaf van de kikkerpuzzel in Figuur 4. De graaftransformaties die we voor deze graaf willen uitvoeren zijn alle geldige zetten die we voor de kikkerpuzzel mogen doen. Door deze transformaties achter elkaar uit te voeren, kunnen we het maken van deze puzzel simuleren en zo naar de oplossing zoeken.

De zetten in de puzzel kunnen in verschillende volgordes worden uitgevoerd. In het begin kan je bijvoorbeeld de groene kikker een stapje naar rechts zetten, of de rode kikker een stapje naar links zetten. Uiteraard zullen we meer dan één zet nodig hebben om de puzzel op te lossen. Om een oplossing voor de puzzel te vinden moeten we dus naar alle mogelijk combinaties van zetten kijken, en kijken of er een combinatie tussen zit die tot de oplossing leidt. Omdat dit veel werk is gebruiken we hier een computerprogramma voor, namelijk GROOVE. In §2.3 vertellen we wat meer over GROOVE, en in §4 laten we zien hoe GROOVE gebruikt kan worden om de kikkerpuzzel op te lossen.

Voordat we verder gaan met uitleg over GROOVE, laten we eerst nog even zien hoe je graaftransformaties kan specificeren (specificeren betekent op een precieze en wiskundige manier opschrijven). Graaftransformaties worden uitgedrukt in *regels*: elke regel beschrijft één soort transformatie. Een (graaf)transformatieregel werkt met behulp van patroon herkenning. Dit betekent dat elke regel een patroon bevat, en dat een regel toegepast kan worden als de graaf aan het patroon dat in de regel staat voldoet.



Figuur 5: Transformatiepatroon: Groene kikker stap vooruit 1



Figuur 6: Transformatiepatroon: Groene kikker stap vooruit 2

Een voorbeeld van zo'n patroon kan je vinden in Figuur 5. Dit patroon is voor de zet waarbij de groene kikker één stap vooruit wordt gezet. Voor deze zet is niet de hele graaf interessant, dus beschrijft het patroon maar een klein deel van de graaf: de groene kikker die we willen verplaatsen, de steen waar de kikker op zit, en de steen waar we de kikker naar toe willen verplaatsen.

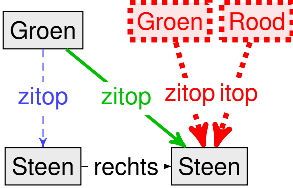
Denk eraan dat als we de groene kikker een stap vooruit willen zetten, dat de nieuwe steen waar de kikker op zal komen te staan wel leeg moet zijn. Dit moeten we expliciet in de transformatieregel zeggen, en dit doen we door rode knopen aan de graaf toe te voegen. Dit betekend dat deze regel alleen mag worden gebruikt als de zwarte knopen *wel* en de rode knopen *niet* in de graaf voorkomen. In ons geval betekent dat, dat er geen kikkers op de volgende steen mogen zitten, zie ook Figuur 6.

In Figuur 6 hebben we nu een patroon dat bepaalt wanneer we een transformatie mogen gebruiken, maar we weten nog niet wat we moeten aanpassen. Voor deze transformatie willen wij dat de groene kikker een stap naar voren zet, ofwel dat de 'zitop'-pijl naar de volgende steen wordt verplaatst. Dit doen we in de regel door aan te geven dat de oude 'zitop'-pijl moet worden verwijderd, en dat er een nieuwe 'zitop'-pijl moet worden toegevoegd. De regel ziet er dan uit zoals in Figuur 7. De oude 'zitop'-pijl is nu blauw gestippeld, wat betekend dat de pijl wel in de graaf voor de transformatie, maar niet in de graaf na de transformatie zit. De nieuwe 'zitop'-pijl is aangegeven met groen en zit in de nieuwe graaf, maar niet in de graaf vóór de transformatie.

Als we transformatieregels wat algemener bekijken zien we dat deze regels zelf ook weer een graaf zijn! De onderdelen in de regels kunnen vier functies hebben, die we allemaal met een eigen kleur aangeven:

Zwarte delen (smal dooregetrokken) Knopen en pijlen die voor én na de transformatie in de graaf voorkomen.

Blauwe delen (smal onderbroken) Knopen en pijlen die wel vóór de transformatie, maar niet erna voorkomen.



Figuur 7: Transformatiepatroon: Groene kikker stap vooruit 3

Groene delen (breed doorgetrokken) Knopen en pijlen die aan de graaf worden toegevoegd.

Rode delen (breed onderbroken) Knopen en pijlen die niet vóór de transformatie mogen voorkomen.

2.3 GROOVE

Voor deze opdracht zal je gebruik gaan maken van het computerprogramma GROOVE. GROOVE is speciaal gemaakt om grafen mee te tekenen, en om automatisch graaftransformaties uit te kunnen voeren. Van `http://sf.net/projects/groove/` kan je GROOVE downloaden. De download bevat ook een korte Engelse handleiding van het programma. In §4 zullen we ook laten zien hoe je GROOVE kan gebruiken om het kikkerprobleem van het voorbeeld uit §2 op te lossen.

Voordat je GROOVE kan gebruiken heb je ook Java nodig. Als je nog geen Java hebt geïnstalleerd, dan kan je de nieuwste versie downloaden via `http://www.java.com/en/download/installed.jsp`.

3 Onderzoeksvragen

In §4 laten we al zien hoe je met graaftransformaties een oplossing kunt vinden voor de kikkerpuzzel. Maar behalve de kikkerpuzzel zijn er nog veel meer puzzels waar je met behulp van graaftransformaties de oplossing voor kan vinden. Hier laten we een paar interessante zien.

De wolf, de geit en de kool

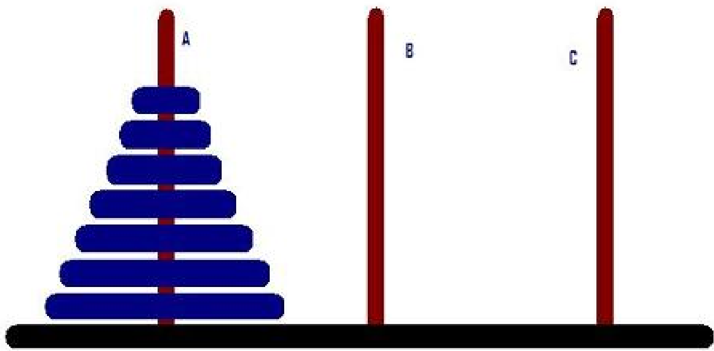
Een boer heeft een wolf, een geit en een kool. Aan het begin staan ze alle drie aan de kant van een rivier, en de boer wil ze met zijn boot naar de overkant brengen. De boot is alleen niet zo groot, dus hij kan maar één ding tegelijk meenemen. Maar als de boer de wolf en de geit alleen laat, dan zal de wolf de geit opeten. En net zo, als de boer de geit alleen laat met de kool, dan eet de geit de kool op. De vraag is nu: hoe kan de boer de wolf, de geit en de kool naar de overkant krijgen, zonder dat er eentje wordt opgegeten? Let op dat de boer ook met een lege boot naar de overkant mag, en dat hij dingen die al aan de overkant staan, ook weer mee terug mag nemen.

Om dit op te lossen met graaftransformaties, moet je eerst een startgraaf maken die beschrijft dat de wolf, de geit, de kool en de boer met zijn boot aan één kant van een rivier staan. Daarna moet je regels maken die beschrijven dat de boer iets naar de overkant van de rivier verplaatst en dat dingen elkaar kunnen opeten.

Drie missionarissen en drie kannibalen

In deze puzzel staan er drie missionarissen en drie kannibalen aan de kant van een rivier. Ze willen naar de overkant en hebben daarvoor een bootje waar twee mensen in kunnen. Het bootje kan niet leeg naar de overkant, er moet altijd minimaal één persoon in het bootje zitten. Als er aan één kant van de rivier meer kannibalen zijn dan missionarissen (de mensen die in het bootje zitten tellen ook mee, dus niet alleen de mensen die op de kant staan), dan worden de missionarissen opgegeten door de kannibalen. Hoe krijg je nu iedereen aan de overkant zonder dat er iemand wordt opgegeten?

Om dit op te lossen met graaftransformaties, moet je eerst een startgraaf maken die beschrijft dat de missionarissen en kannibalen aan één kant van een rivier staan. Daarna moet je regels maken de beschrijven dat het bootje met mensen naar de overkant gaat en dat de missionarissen kunnen worden opgegeten.



Figuur 8: De torens van Hanoi

De torens van Hanoi

In het Boeddhistische Hanoi zijn monniken al eeuwen bezig een toren van 64 schijven van oplopende grootte van de ene berg naar de andere te verhuizen, door de zware schijven één voor één te verplaatsen. Naast de beginplaats en de gewenste eindplaats is er een tussenstop waarop schijven tijdelijk kunnen worden neergelegd. Een schijf mag nooit op een kleinere schijf komen te liggen. Hoe moeten de monniken die doen?

Een kleinere toren, van 7 schijven, is schematisch afgebeeld in Figuur 8: A is de beginpositie, B de tussenpositie en C de gewenste eindpositie.

Het aantal stenen in de puzzel staat niet vast. Dit betekent dat je kan beginnen met een simpele versie van de puzzel (bijvoorbeeld met drie stenen), en daarna de puzzel kan uitbreiden met meer stenen.

4 Een simpel voorbeeld

In dit §gaan we het voorbeeld van de kikkerpuzzel uitwerken. De begin situatie van de kikkerpuzzel is zoals in Figuur 3 en het doel van de puzzel is om alle kikkers met alleen de volgende zetten naar de overkant te krijgen:

- Als de steen rechts van een groene kikker leeg is, mag de groene kikker een stap naar rechts doen.
- Als op de steen rechts van een groene kikker een rode kikker zit, dan kan de groene kikker daar overheen springen. Let op, dit mag alleen als de steen waarop de groene kikker landt leeg is, en de groene kikker mag maar over één rode kikker tegelijk heen springen.
- Als de steen links van een rode kikker leeg is, mag de rode kikker een stap naar links doen.
- Als op de steen links van een rode kikker een groene kikker zit, dan kan de rode kikker daar overheen springen. Let op, dit mag alleen als de steen waarop de rode kikker land leeg is, en de rode kikker mag maar over één groene kikker tegelijk heen springen.

De uitwerking van dit voorbeeld bestaat uit drie stappen: het maken van een graaf van de begin situatie; het maken van de transformatieregels; de transformatieregels toepassen om de oplossing te vinden. Voor het uitwerken zullen we gebruik maken van het programma GROOVE; in §2.3 staat waar je dit programma kan downloaden.

4.1 De startgraaf

Open GROOVE en begin met een nieuwe grammatica door bij 'File' op 'new Grammar' te klikken. Het onderste vak van de linker kolom is nu nog leeg, maar bevat straks de lijst met grafen die hebben gemaakt. Klik in dit blok op de nieuwe graaf knop (, let op de kleine G), en een nieuw scherm wordt geopend waarin we grafen kunnen tekenen.

Bovenin het scherm staan nu een aantal knoppen. Door te klikken en te slepen kan je nieuwe knopen en pijlen tekenen, en bestaande knopen verslepen. Als je een knoop of pijl een (nieuw) label wilt geven, moet je dit doen door erop te dubbelklikken.

Nu zou je zelf de startgraaf moeten kunnen tekenen, hij moet er ongeveer uitzien zoals in Figuur 4. Denk eraan dat als knopen of pijlen in de graaf hetzelfde label hebben, dit ook precies hetzelfde is gespeld, inclusief hoofdletters. Als je dit niet goed doet, zullen later de transformatieregels niet goed werken.

Denk er aan de graaf op te slaan als hij klaar is.

4.2 Transformatieregels

Nu we onze startgraaf hebben gemaakt, kunnen we beginnen aan de transformatieregels. De transformatieregels komen bovenaan in de linker kolom te staan. Als je daar op de nieuwe regel knop drukt (, let op de kleine 'R'), opent er weer een scherm waar je grafen kan tekenen. Dit scherm werkt hetzelfde als het scherm dat we gebruikte voor de startgraaf, je kunt nu dus zelf de knopen en de pijlen voor de regel in Figuur 7 tekenen.

Om de knopen en pijlen hun speciale betekenis (de kleuren) te geven, moeten we zogenaamde keywords toevoegen aan hun labels. De keywords die je nodig zou kunnen hebben kan je vinden in §6. Er zijn drie belangrijke keywords:

new: Betekent dat een knoop of pijl nieuw moet worden aangemaakt, een groene knoop of groene pijl dus.

del: Betekent dat een knoop of pijl verwijderd moet worden, een blauwe knoop of blauwe pijl dus.

not: Betekent dat een knoop of pijl helemaal niet mag voorkomen, een rode knoop of rode pijl dus.

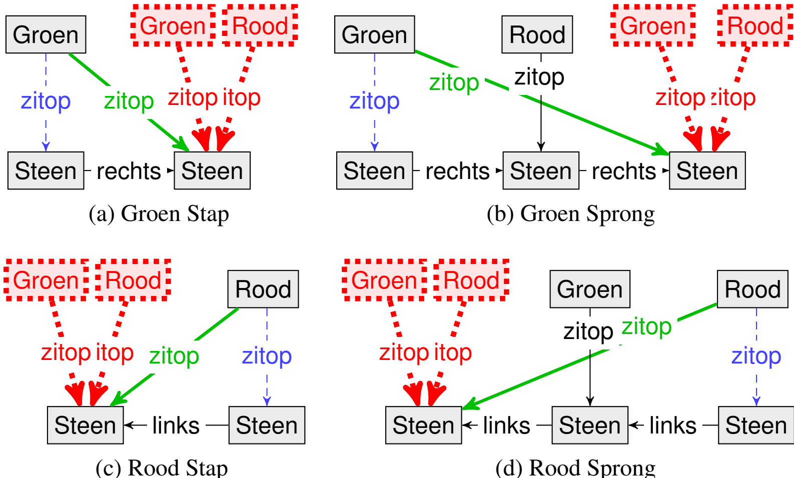
Let op! Als je aan een pijl een keyword wil toevoegen moet dit op één regel direct voor het label. Als je bij een knoop een keyword wil toevoegen moet dit op een aparte regel. Gebruik 'Shift+Enter' om in de editor een nieuwe regel in te voeren.

Omdat we vier soorten zetten mogen doen in deze puzzel, hebben we ook vier transformatieregels nodig. In Figuur 9 kan je vinden hoe deze regels eruit komen te zien.

4.3 Toestandsruimte

Nu we ook de transformatieregels hebben, kunnen we GROOVE laten zoeken naar een oplossing van de puzzel. Dit doen we door GROOVE alle toestanden van het systeem te laten uitrekenen, de zogenaamde *toestandsruimte*. De toestandsruimte bevat alle grafen die we kunnen maken met de startgraaf en de transformatieregels die we hebben gemaakt. In ons geval zal de toestandsruimte dus alle mogelijk combinaties van zetten bevatten, en bij elke combinatie de plaats waar de kikkers staan. Om de toestandsruimte te berekenen moet je (in GROOVE) op 'Explore' en dan op 'Explore State Space' klikken. In het tabblad 'LTS' zal nu de toestandsruimte verschijnen.

Als je alles goed hebt gedaan, zal de toestandsruimte 23 grafen bevatten. Als je één van de toestanden selecteert, kan je in het meest linker tabblad zien hoe deze graaf er uit ziet. De toestandsruimte bevat een paar speciale toestanden: de groene 's0' graaf is onze startgraaf, de oranje grafen zijn eindtoestanden, vanuit deze situaties kunnen we dus geen nieuwe transformaties (en dus zetten) meer doen. Als je wil



(a) Groen Stap (b) Groen Sprong

(c) Rood Stap (d) Rood Sprong

Figuur 9: Transformatieregels voor de kikkerpuzzel

weten welke transformaties (en dus zetten) nodig zijn om een bepaalde graaf te bereiken, moet je de pijlen vanuit 's0' naar die graaf volgen.

Nu gaan we in de toestandsruimte op zoek naar de graaf die de oplossing beschrijft. In deze graaf staan de rode kikkers helemaal links en de groene kikkers helemaal rechts, en kunnen we geen zetten meer doen. De oplossing zou dus een van de oranje grafen moeten zijn, en inderdaad graaf 's22' is onze eindsituatie. Om de oplossing te vinden hoeven we nu alleen maar de pijlen van 's0' naar 's22' te volgen en te kijken welke zet er bij elke pijl wordt uitgevoerd.

In plaats van zelf naar de gewenste eindtoestand te zoeken kan je dit ook weer aan GROOVE overlaten. Maak een nieuwe transformatieregel die test op het gewenste eindpatroon; in dit geval de graaf waarin de groene kikkers rechts en de rode links op het veld staan. De regel hoeft niets aan deze graaf te veranderen. Als je nu de toestandsruimte uitrekent kan je zien waar (d.w.z., in welke toestand) deze nieuwe regel toepasbaar is. Het pad in te toestandsruimte van de begintoestand naar deze eindtoestand is de oplossing van de puzzel.

5 Woordenlijst

Graaf Verzameling knopen en pijlen, die samen een toestand (situatie) beschrijven. Denk bijvoorbeeld aan een spoorwegenkaart; de knopen zijn de treinstations en de pijlen de rails tussen de treinstations.

Graaftransformatie Aanpassen van een graaf naar een nieuwe graaf. Geldige aanpassingen zijn: het toevoegen van nieuwe knopen, het verwijderen van knopen, het toevoegen van nieuwe pijlen, het verwijderen van pijlen en het verplaatsen van pijlen. Een graaftransformatie wordt beschreven met een (graaf)transformatieregel.

Grammatica Verzameling transformatieregels.

GROOVE GROOVE is een computer programma dat gebruikt kan worden voor het ontwerpen en uitvoeren van graaftransformaties.

Knoop (Engels: node): Element in een graaf waartussen pijlen getrokken kunnen worden. Een knoop heeft nul, één of meer labels.

Label Tekst op een knoop of pijl.

Pijl (Engels: edge): Gerichte verbinding tossen twee knopen, voorzien van een label.

Simuleren Bij een (computer)simulatie laten we de computer uitrekenen wat er allemaal kan gebeuren. In het geval van de puzzels in deze opdracht laten we de computer dus berekenen welke zetten er allemaal mogelijk zijn.

Specificeren Als we iets specificeren (bijvoorbeeld een transformatieregel), bedoelen we dat we het op een wiskundige en precieze manier opschrijven.

Toestandsruimte Als we een startgraaf hebben en een verzameling transformatieregels voor deze graaf, dan bevat de toestandsruimte alle nieuwe grafen die we kunnen maken met deze startgraaf en transformatieregels. Als de startgraaf een puzzel voorstelt en de transformatieregels de mogelijke zetten zijn, dan zal de toestandsruimte alle mogelijke tussenstappen van de puzzel bevatten.

Transformatieregel Wiskundige notatie voor het opschrijven van een bepaalde graaftransformatie.

6 GROOVE aanwijzingen

6.1 Keywords

In graaftransformaties is het mogelijk knopen en pijlen toe te voegen en te verwijderen. Met de volgende keywords kan je aangeven welke knopen of pijlen moeten worden aangepast. Als je een keyword aan een knoop toe wil voegen, moet dit op een nieuwe regel. In GROOVE ga je niet met ‘enter’ naar een nieuwe regel, maar met ‘shift + enter’.

Een nieuwe pijl Zet ‘new:’ vóór het label van de pijl.

Een nieuwe knoop Zet ‘new:’ in een apart label van de knoop.

Een pijl verwijderen Zet ‘del:’ vóór het label van de pijl.

Een knoop verwijderen Zet ‘del:’ in een apart label van de knoop.

Een verboden pijl Zet ‘not:’ vóór het label van de pijl.

Een verboden knoop Zet ‘not:’ in een apart label van de knoop.

6.2 Prioriteiten

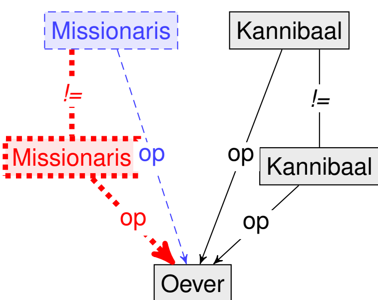
Normaal gesproken worden in alle grafen van de toestandsruimte alle regels uitgeprobeerd. Soms is dat niet de bedoeling en is een regel alleen geldig als een andere regel niet toepasbaar is. Bijvoorbeeld zouden we in de kikkerpuzzel als extra beperking kunnen invoeren dat een kikker niet mag stappen als er een (andere) kikker kan springen, of vice versa; of dat rood niets mag doen als er nog een groene kikker kan bewegen.

Dit kan in GROOVE worden aangegeven door sommige regels een hogere *prioriteit* te geven. Klik met de rechtermuisknop op een regel en kies “Rule Properties”; vul dan in de tabel bij “Priority” een positief getal in. Deze regel wordt nu als eerste uitgeprobeerd; andere regels komen alleen aan de beurt als geen van de regels met hogere prioriteit niet uitgevoerd kunnen worden.

Bijvoorbeeld kan je de kikker-sprong-regels prioriteit 10 geven, en de andere prioriteiten op 0 laten; of juist de stap-regels. (Ga na dat in het laatste geval de puzzel niet meer oplosbaar is.)

6.3 Discriminatie

Some is het nodig in een regel aan te geven dat twee knopen *niet* hetzelfde kunnen zijn. Dit kan je doen door een speciale pijl tussen die twee knopen te trekken met als label ‘!=’. Figuur 10 illustreert dit aan de hand van een regel die zegt dat een missionaris wordt opgegeten als er twee verschillende kannibalen en geen andere missionaris op dezelfde oever zijn.



Figuur 10: Twee kannibalen tegen één missionaris

Bellen Zonder Zorgen

Je hebt het vast wel eens gehad. Ben je lekker aan het werk op je computer loopt hij ineens vast! En natuurlijk heb je het werk niet opgeslagen. Je probeert nog van alles om te redden wat er te redden valt, maar uiteindelijk moet je toch opgeven en de computer opnieuw opstarten. Je kunt dan weer helemaal opnieuw beginnen.

Behalve dit zijn er nog veel meer problemen met computers. Denk bijvoorbeeld aan de belastingdienst die een paar jaar geleden ruim 700.000 mensen geen geld terug kon geven omdat de computer de aangiftes kwijt was geraakt (`http://www.vkbanen.nl/actueel/714010/Belastingdienst-raakt-730000-aangiftes-kwijt.html`). Op de universiteit wordt er daarom veel onderzoek gedaan naar verschillende manieren om fouten in computerprogramma's te kunnen vinden, op te lossen en zelfs te voorkomen. Deze opdracht gaat over één van die manieren, namelijk model checking.

1 Wat ga je doen?

Voor dit profielwerkstuk ga je je verdiepen in model checking. Dit is een speciale techniek waarbij je modellen maakt van een computerprogramma zodat je bugs (fouten) in het programma op kan sporen en verhelpen. In Hoofdstuk 2 vind je een korte introductie over wat model checking is. Vervolgens kan je in Hoofdstuk 3 een uitbreiding op het voorbeeld vinden die je voor je hoofdvraag kan gebruiken. Ook zijn er een aantal alternatieve hoofdvragen en onderwerpen vinden om je verder op weg te helpen. Tenslotte staat in Hoofdstuk 4 een voorbeeld van hoe model checking kan worden gebruikt voor het controleren van een simpel programma.

Aan het eind van de opdracht kan je een woordenlijst vinden met belangrijke en moeilijke begrippen en een korte uitleg daarvan vinden. Als je toch ergens niet uitkomt, kan je altijd hulp vragen bij de universiteit. Stuur daarvoor een mailtje naar Mariëlle Stoelinga (m.i.a.stoelinga@utwente.nl).

2 Wat is model checking?

'Model checking' is Engels en betekend letterlijk 'modelcontrole'. Het is een techniek die in de informatica is bedacht om te controleren of een computerprogramma zich gedraagt zoals wij willen. Dit doen we niet door direct een computerprogramma te controleren, maar door eerst een model van het programma te maken en dat te controleren. Het voordeel hiervan is dat de modellen vaak veel eenvoudiger zijn dan het computerprogramma, en daarom zijn de modellen ook gemakkelijker te controleren. Bij model checking hebben we een aantal belangrijke dingen nodig:

Een model Computerprogramma's zijn vaak erg ingewikkeld en zijn daarom moeilijk te controleren. Daarom maken we eerst een model van het computerprogramma, dat de belangrijkste eigenschappen van het programma bevat. Dat model gedraagt zich dan net zoals het computerprogramma, maar het is veel simpeler dus ook makkelijker te controleren!

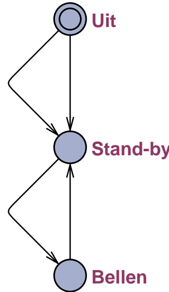
Een specificatie Om te kunnen bepalen of een programma zich gedraagt zoals wij willen, moeten we eerst heel precies opschrijven wat wij van het programma verwachten. Dit noemen we het specificeren van het programma.

Een model checker Een model checker is zelf een computerprogramma dat het belangrijke rekenwerk uitvoert. Dit gebeurt met behulp van het model en de specificatie, die eerder zijn gemaakt. De model checker gebruikt dan wiskunde om automatisch te bepalen of het model zich gedraagt volgens de specificatie die wij hebben gemaakt. Zodra er fouten in het programma zitten zal de model checker hier achter komen omdat het model niet aan de specificatie voldoet. Een programmeur kan dan de fout in het programma verbeteren.

Modellen

Om modellen te maken voor model checking wordt er vaak gebruik gemaakt van toestandsdiagrammen. Een voorbeeld van een toestandsdiagram van een simpele telefoon kan je vinden in Figuur 1. De cirkels in het diagram zijn de toestanden van de telefoon. Zoals je kunt zien zijn er drie toestanden, namelijk de telefoon staat 'Uit'; de telefoon staat op 'Stand-by'; de telefoon is aan het 'Bellen'. De pijlen tussen de toestanden noemen we overgangen, en ze geven aan hoe de toestanden kunnen veranderen. Er staat bijvoorbeeld een pijl tussen de toestand 'Uit' en de toestand 'Stand-by'. Dit betekent dat als de telefoon uit staat, je de telefoon kan aanzetten en dat hij daarna in stand-by gaat. Omdat er geen pijl van 'Uit' naar 'Bellen' gaat, kan de telefoon niet bellen als hij uit staat en moet je hem eerst aanzetten.

Het programma waar we een model van maken bestaat vaak uit meerdere delen. Denk bijvoorbeeld weer aan de telefoon. Als je belt is er nooit maar één telefoon, maar is er ook een tweede telefoon waar je naar toe belt. Die telefoons staan op hun beurt weer in verbinding met elkaar met behulp van telefooncentrales. Als we al deze delen in één model moeten zetten wordt het model vaak erg ingewikkeld en onoverzichtelijk. Daarom maken we vaak in plaats van één groot model verschillende deelmodellen. Op de computer kunnen we deze



Figuur 1: Voorbeeld Model: Een Simpel Telefoon Model

modellen dan automatisch samenvoegen tot één groot model. Dit samenvoegen wordt *synchronisatie* genoemd.

In het voorbeeld verderop in de opdracht maken we ook gebruik van synchronisatie; daar zal ook iets preciezer worden uitgelegd hoe dit werkt. In het kort is *synchronisatie* een manier voor modellen om met elkaar te kunnen communiceren. Een combinatie van gesynchroniseerde modellen wordt in Uppaal ook wel een systeem genoemd.

Specificatie

Als we een model hebben, dan kunnen we daarna ook het gewenste gedrag van het model specificeren. Omdat deze specificaties erg precies moeten zijn, is het niet genoeg om ze gewoon op te schrijven in 'natuurlijke taal' (gewoon Nederlands). Daarom wordt voor het opschrijven van de specificaties een speciale wiskundige taal gebruikt: logica.

Er zijn heel veel soorten logica. Voor dit werkstuk ga je gebruik maken van een logica die we CTL (Computation Tree Logic) noemen. CTL is erg uitgebreid, en voor deze opdracht zullen we maar een klein deel bekijken.

De CTL die wij gaan gebruiken zal altijd een van de onderstaande vormen hebben:

A[] eigenschap Deze constructie betekent dat de eigenschap altijd moet gelden. Stel je voor dat je altijd bereikbaar wil zijn, en dat je telefoon dus nooit uit mag staan. Dit betekend dat je altijd in de toestand 'Stand-by' of 'Bellen' zit. In CTL zou dat er dan zo uitzien A[] ('Stand-by' || 'Bellen'). In CTL staat || voor of, en deze stelling zegt dus dat de telefoon altijd of op 'Stand-by' of op 'Bellen' staat.

E<> eigenschap Deze constructie betekent dat de eigenschap ergens in de toekomst moet gelden. Als je zou willen weten of je ooit gebeld wordt, kan je dat dus zo zeggen in CTL: E<> 'Bellen' (ooit wordt er gebeld).

not eigenschap Not betekent dat de eigenschap juist niet moet gelden. In het eerste voorbeeld zeiden we dat de eigenschap dat de telefoon nooit uit stond zo kon worden beschreven: A[] ('Stand-by' || 'Bellen'). Maar als we not gebruiken kunnen we ook het volgende zeggen: A[] not 'Uit', wat precies hetzelfde betekent.

A[] not deadlock Dit is een speciale eigenschap die zegt dat er geen deadlocks in het programma zitten. Een deadlock is een situatie waarin een programma niets meer kan doen en dus is vastgelopen. Als je programma dus geen deadlocks heeft, zal het niet vastlopen.

Model Checker

Op het internet zijn veel model checkers te vinden (een aantal voorbeelden zijn: SPIN, JavaPathFinder en Uppaal). Voor deze opdracht gaan we Uppaal gebruiken. Het voordeel van Uppaal is dat het een grafische interface heeft. Dit betekent dat je op een simpele manier de toestandsdiagrammen in Uppaal kan tekenen. Ook kan je gemakkelijk de uitslag van je tests aflezen van je scherm en daardoor is Uppaal dus gemakkelijk om te leren voor beginners.

Uppaal kan je downloaden van `www.uppaal.com` en het werkt op Windows, Linux en OS X. Je hebt er wel minimaal Java versie 6 nodig. Als je dit nog niet hebt, dan kan je dat downloaden via `http://www.java.com/en/download/installed.jsp`.

Een uitgebreidere beschrijving van Uppaal kan je vinden in het voorbeeld van Hoofdstuk 4.

3 Onderzoeksvragen

Mobieltjes

In Hoofdstuk 4 staat een klein voorbeeld over mobieltjes met uitleg over hoe Uppaal werkt. Je kan dit voorbeeld overnemen en uitbreiden met wat extra opties:

- Als iemand na meer dan 15 seconden opneemt (dus als de telefoon al naar de voicemail is overgeschakeld) dan gaat de voicemail uit en wordt er alsnog een normaal telefoon gesprek gevoerd.
- Het voorbeeld heeft nu twee telefoons. Voeg daar wat meer telefoons aan toe, die allemaal met elkaar kunnen bellen.
- Echte mobiele telefoons bellen niet direct met elkaar, zoals in ons voorbeeld, maar via een zendmast. Je kan dus een zendmast te modelleren, zodat alle telefoon gesprekken via de zendmast lopen.

Voor je profielwerkstuk kan je dan een aantal interessante eigenschappen bedenken, en onderzoeken of ze geldig zijn voor jouw model. Een aantal voorbeelden van eigenschappen die je zou kunnen onderzoeken zijn:

- Een telefoon wordt soms opgenomen.
- Een telefoon schakelt soms door naar de voicemail
- Geen twee telefoons kunnen tegelijkertijd met dezelfde zendmast bellen.
- Alle telefoons komen een keer aan de beurt om te mogen bellen.

Master class Frits Vaandrager

Frits Vaandrager heeft een website over modelchecking voor 5 en 6 VWO, het adres is: `http://www.cs.ru.nl/~fvaan/Masterclass`. Op zijn website kan je een aantal voorbeelden vinden van modelchecking opdrachten. Je kunt deze gebruiken als basis voor je PWS.

4 Een Simpel Voorbeeld: Mobieltjes

In dit Hoofdstuk zullen we een klein voorbeeld laten zien zodat je een idee krijgt hoe Uppaal werkt. Als je ergens niet uitkomt, kan je in de handleiding voor beginners kijken (`http://www.mbsd.cs.ru.nl/publications/papers/fvaan/uppaalhandleiding/handleiding.pdf`), of een mailtje sturen naar Mariëlle Stoelinga (m.i.a.stoelinga@utwente.nl).

Het Voorbeeldsysteem

Als voorbeeld gaan we een aantal mobiele telefoons modelleren. Als de telefoons niets doen, dan staan ze in stand-by. Als een telefoon in stand-by staat, dan kan hij naar andere telefoon bellen, of een gesprek opnemen. Als een telefoon een ander belt en er wordt niet binnen 15 seconden opgenomen, zal de telefoon automatisch doorschakelen naar de voicemail.

Voor dit systeem beginnen we met het modelleren van de telefoons. In Uppaal moet je hiervoor het 'Editor' tabblad gebruiken.

De Telefoons

Omdat alle telefoons op dezelfde manier werken, hoeven we maar één Uppaal model van een telefoon te maken. Dit model kunnen we dan hergebruiken zodat één model meerdere telefoons kan beschrijven.

Open, om te beginnen aan dit model, Uppaal met een nieuw systeem. Je kunt een nieuw systeem aanmaken door op 'File → New System' te klikken. Selecteer nu aan de linkerkant van het scherm het template, dat is een nieuw model dat standaard door Uppaal wordt aangemaakt.

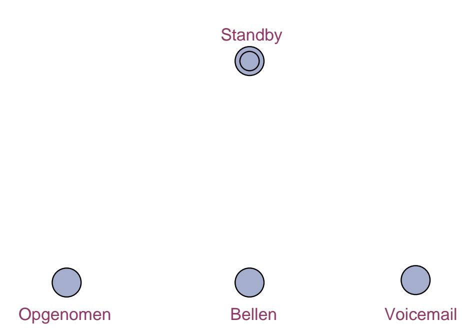
Omdat 'template' niet een erg handige naam is voor een model van een telefoon, willen we deze eerst veranderen. Dit kan je doen door bovenaan in het veld 'Name' een nieuwe naam in te voeren. Noem je model bijvoorbeeld 'Telefoon".

Nu kunnen we rechts het model van de telefoon gaan tekenen. We beginnen met de toestanden. Voor de telefoon kunnen we vier toestanden onderscheiden:

1. De telefoon staat in stand-by en wacht totdat hij een andere telefoon moet opbellen of totdat zelf wordt opgebeld.

2. De telefoon probeert iemand te bellen, maar we weten nog niet of er opgenomen gaat worden, of dat we naar de voicemail doorgeschakeld gaan worden.

3. De telefoon is in gesprek. Dit kan omdat wij zelf iemand hebben gebeld en zij hebben opgenomen, of omdat iemand ons heeft gebeld en wij hebben opgenomen.



Figuur 2: De toestanden van de telefoon

4. De telefoon is doorgeschakeld naar de voicemail. Dit gebeurt als de telefoon iemand probeerde te bellen, maar als er niet op tijd werd opgenomen.

Toestanden in Uppaal zijn blauwe cirkels. Je kunt ze aanmaken door op de middelste muisknop te drukken of met de toestandenknop ( ) boven in de werkbalk. De naam van een toestand kan je veranderen door te dubbelklikken op een toestand en dan bij ’Name’ een naam in te vullen. In Figuur 2 kan je zien hoe dit er dan uitziet. Hier is toestand 1 ’Standby’ genoemd, toestand 2 is ’Bellen’, toestand 3 is ’Opgenomen’ en toestand 4 is ’Voicemail’.

Let op dat toestand ’Standby’ een dubbele cirkel heeft. Dit is om aan te geven dat het model in deze toestand begint, ofwel als we de telefoon aanzetten start hij in stand-by. De begintoestand kan je instellen door op een toestand te dubbelklikken en het vinkje bij ’Initial’ aan of uit te zetten. Let op dat je in Uppaal meerdere begintoestanden tegelijk kan hebben. Dit is niet wat we willen, dus zorg ervoor dat bij alle toestanden behalve bij ’Standby’ het vinkje bij ’Initial’ uitstaat.

Nu gaan we de overgangen tussen de toestanden maken. Voor de telefoon zijn er de volgende overgangen:

1. Van ’Standby’ naar ’Bellen’: als we iemand willen bellen.
2. Van ’Standby’ naar ’Opgenomen’: als iemand ons belt, en we hebben opgenomen.
3. Van ’Bellen’ naar ’Opgenomen’: als we iemand bellen en er wordt opgenomen.

4. Van 'Bellen' naar 'Voicemail': als we iemand bellen, maar er wordt niet opgenomen.
5. van 'Opgenomen' naar 'Standby': als het gesprek is afgelopen en er wordt opgehangen.
6. van 'Voicemail' naar 'Standby': als een voicemailbericht is ingesproken en er wordt opgehangen.

Een overgang tussen twee toestanden kan je tekenen met de overgangen knop ( ) in de werkbalk, of door met de middelste muisknop eerst op de begintoestand en dan op de eindtoestand te klikken.

We willen straks het model van de telefoon hergebruiken, en we willen dat de verschillende telefoons met elkaar worden gesynchroniseerd. Om dit mogelijk te maken moeten we commando's toevoegen aan de overgangen die tegelijk uitgevoerd moeten worden. Deze kan je invullen door te dubbelklikken op een overgang en bij 'Sync' een naam in te vullen. Deze naam moet eindigen met een ? of een !. Een ? betekent 'ik wacht totdat ik dit commando krijg', en een ! betekend 'ik geef dit commando'.

Kijk bijvoorbeeld naar de tweede overgang van 'Standby' naar 'Opgenomen', daar geeft de telefoon een seintje dat hij gaat opnemen nadat iemand hem heeft gebeld. Daarom moet het commando bij deze overgang eindigen met een !. Bij de derde overgang van 'Bellen' naar 'Opgenomen' moet de telefoon een commando van een andere telefoon ontvangen dat er is opgenomen. Het commando bij deze overgang moet dus op een ? eindigen.

Voor de andere overgangen geldt dat we geen commando's nodig hebben. Als een telefoon iemand op belt (voordat er is opgenomen) of als we worden doorgeschakeld naar de voicemail, zijn we niet afhankelijk van een andere telefoon. Een telefoon moet deze acties dus zelfstandig, zonder synchronisatie uit kunnen voeren.

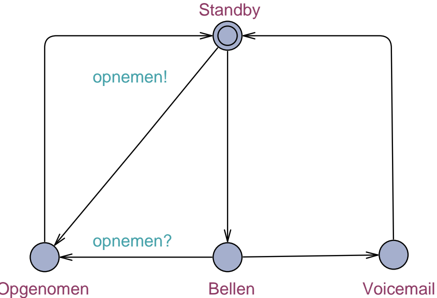
In Figuur 3 kan je zien hoe de overgangen eruit komen te zien. Let goed op waar ? en waar ! staat.

Tenslotte moeten we er voor zorgen dat de telefoon naar de voicemail gaat als er 15 seconden lang niet wordt opgenomen. Dit kunnen we doen met behulp van een klok. Deze klok begint te tikken na de eerste overgang (tussen 'Standby' en 'Bellen'). Daarom voegen we in het 'Update' veld van deze overgang 'klok:=0' toe. Dit betekent dat na deze overgang de klok vanaf 0 begint te lopen.

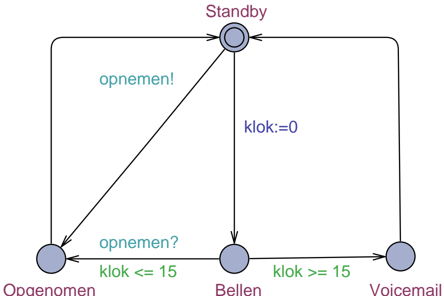
Vervolgens moeten we er voor zorgen dat de derde overgang (van 'Bellen' naar 'Opgenomen') alleen kan worden genomen als de klok op minder dan 15 seconden staat en dat de vierde overgang (van 'Bellen naar 'Voicemail') alleen wordt genomen als de klok op meer dan 15 seconden staat. Dit kunnen we doen door een 'guard' toe te voegen aan de overgangen. Een guard zorgt ervoor dat een overgang alleen kan worden genomen als er aan de voorwaarden in de guard is voldaan. Voeg dus bij de derde overgang de guard 'klok <= 15' toe en aan de vierde overgang de guard 'klok >= 15'. Nu kan er alleen in de eerste 15 seconden worden opgenomen, en anders kunnen we alleen nog maar naar de voicemail.

Als het goed is ziet je model er nu uit als in Figuur 4.

Om de commando's die we net in het model hebben gezet te kunnen gebruiken moeten we eerst aan Uppaal duidelijk maken wat de commando's en



Figuur 3: De overgangen van de telefoon



Figuur 4: Het model van de telefoon

klokken zijn. Dit wordt declareren genoemd. Omdat de commando's voor alle modellen (telefoons) hetzelfde zijn, doen we dit in het 'Declarations' bestand (let op niet in het 'System Declarations' bestand) dat bovenaan in de linker kolom staat. Een commando declareer je zo:

```
chan commandonaam;
```

dus bijvoorbeeld zo:

```
chan opnemen;
```

Behalve de commando's moeten we ook de klok declareren. Maar elke telefoon heeft zijn eigen klok, in tegen stelling tot de commando's die voor elke telefoon hetzelfde zijn. Daarom declareren de klok niet in het globale 'Declarations' bestand, maar in het 'Declarations' bestand van de telefoon. Dit bestand kan je vinden door naast de telefoon op het plusje te klikken. Een klok declareer je als volgt:

```
clock kloknaam;
```

onze klok wordt dus zo:

```
clock klok;
```

Synchronisatie

Tenslotte moeten we Uppaal vertellen dat er twee telefoons gesynchroniseerd moeten worden, dit gebeurt in het 'System Declarations' bestand. Om de twee telefoons te maken moet je de volgende regels kopiëren:

```
telefoon1 = Telefoon();
telefoon2 = Telefoon();
system telefoon1,telefoon2;
```

Deze regels zeggen dat we twee telefoons hebben, namelijk telefoon1 en telefoon2. Deze telefoons gedragen zich zoals het 'Telefoon' model. De laatste regel zegt dat deze twee dingen moeten worden gesynchroniseerd.

Simulatie

Nu is het model klaar. Als je nu op het tabblad 'Simulator' klikt zal het model in een simulator worden geladen. Je ziet dan rechts de toestandsdiagrammen van de twee telefoons. Door rechtsonder op 'random' te klikken start Uppaal een simulatie en zal je zien dat de toestanden van het systeem veranderen.

Door de simulatie een tijdje door te laten gaan kun je kijken of de telefoons zich gedragen zoals je verwacht. Maar om echt 100

Specificatie Maken en Controleren

Om te beginnen gaan we een simpele eigenschap controleren, namelijk de eigenschap dat de twee telefoons in ons systeem de mogelijkheid hebben om met elkaar te bellen. Denk eraan dat een telefoon is in gesprek als hij in de 'Opgenomen' toestand zit, alleen omdat we nu twee telefoons hebben heet deze toestand

nu 'telefoon1.Opgenomen' of 'telefoon2.Opgenomen'. Op deze manier kunnen we zien welke van de twee telefoons in de 'Opgenomen' toestand zit. Als de twee telefoons met elkaar in gesprek zijn, dan zitten ze allebei tegelijk in de 'Opgenomen' toestand. In Uppaal schrijven we dat zo op:

```
telefoon1.Opgenomen == telefoon2.Opgenomen
```

Dit betekend dat als 'telefoon1' in de 'Opgenomen' toestand zit, dan moet 'telefoon2' ook in de 'Opgenomen' toestand zitten, en andersom.

Nu hebben we dus de eigenschap die we willen testen. Maar als je Hoofdstuk 2 hebt gelezen, weet je dat de eigenschap alleen niet genoeg is, we moeten ook nog zeggen wanneer deze eigenschap geldt. Daarvoor hebben twee opties, namelijk: de eigenschap geldt altijd (A[]), of de eigenschap geldt ooit (E<>). We gebruiken nu de ooit eigenschap, omdat we willen kijken of het mogelijk is dat beide telefoons ooit met elkaar in gesprek komen. De complete specificatie ziet er dan zo uit:

```
E<> (telefoon1.Opgenomen == telefoon2.Opgenomen)
```

Ga nu naar het 'Verifier' tabblad. In dit tabblad kunnen we controleren of ons model aan onze specificatie (onze formule) voldoet. Kopiëer daarvoor de formule naar het 'query' veld en druk rechts op 'check'. Als het goed is geeft Uppaal nu een melding 'Property is satisfied', wat betekend dat de formule geldig is. De telefoons kunnen elkaar dus bellen.

In plaats van ooit, hadden we ook altijd in onze eigenschap kunnen gebruiken:

```
A[] (telefoon1.Opgenomen == telefoon2.Opgenomen)
```

Deze eigenschap betekent dat altijd als 'telefoon1' in gesprek is, 'telefoon2' ook altijd in gesprek is (namelijk met 'telefoon1'). Dit klinkt heel logisch, als je iemand belt dan ben je beide in gesprek. Maar als je deze eigenschap probeert te controleren, dan zul je zien dat deze niet geldt voor onze telefoons. Probeer nu zelf te ontdekken hoe het kan dat maar één van de telefoons tegelijk in gesprek is. Hiervoor kan je het simulatie tabblad gebruiken, door de simulatie stap voor stap uit te voeren. Als je het probleem hebt gevonden, dan kan je het als het goed is nu ook zelf oplossen.

Andere eigenschappen die je ook zou kunnen controleren zijn:

- Soms wordt de telefoon opgenomen.
- Soms wordt er doorgeschakeld naar de voicemail.
- Het systeem heeft geen deadlocks (het loopt nooit vast).

5 Woordenlijst

CTL CTL is een logica (een wiskundige taal) die we gebruiken om heel precies de eigenschappen van een systeem op te kunnen schrijven.

Deadlock Een deadlock is een speciale situatie in een computerprogramma waarin dat programma is vastgelopen en dus niets meer doet.

Model Een model is een beschrijving van een systeem door een toestandsdiagram.

Model Checker Een model checker is een computerprogramma dat met een model en een specificatie automatisch kan controleren of een ander computer programma goed werkt.

Natuurlijke Taal Natuurlijke taal is de technische term voor normale taal zoals mensen die gebruiken om met elkaar te praten en te schrijven. De tegenhanger van de natuurlijke taal is de wiskundige taal (de logica).

Specificatie Een serie regels in CTL (of andere wiskundige taal) die de eigenschappen van een computerprogramma beschrijven.

Systeem Een combinatie van gesynchroniseerde modellen in Uppaal.

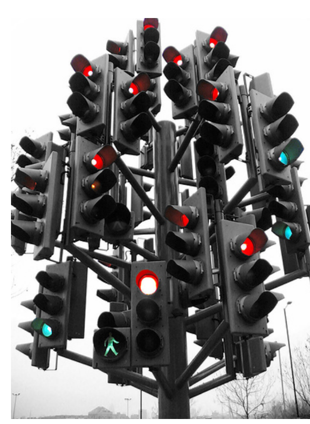
Toestandsdiagram Een toestandsdiagram is een beschrijving van een systeem. De toestanden in het diagram staan voor de verschillende situaties waarin het systeem zich kan bevinden. De overgangen tussen de toestanden geven aan hoe deze situaties kunnen veranderen.

Uppaal Uppaal is een model checker en is te downloaden via www.uppaal.com.

Gelijk oversteken
Een Roblox profielwerkstuk

1 Inleiding

Eerst naar links kijken, dan naar rechts kijken en dan voor de zekerheid nog een keer naar links kijken. Alleen als de weg dan nog vrij is, kun je veilig oversteken.



Als we dit bij elk kruispunt waar we met de auto, fiets of op de voet langskomen zouden moeten doen, dan zouden we de hele dag in de file staan. Om te helpen bij het oversteken hebben we daarom op veel kruispunten stoplichten gezet. Wil je weten hoe stoplichten veilig gemaakt worden, waarbij we ook nog zo kort mogelijk voor een rood licht staan? Kijk dan verder in deze opdracht.

2 Wat ga je doen?

Voor dit profielwerkstuk ga je kijken naar een computerprogramma dat een kruispunt met stoplichten simuleert. Met dit programma ga je onderzoeken wat de beste manier is om de stoplichten te gebruiken. In Hoofdstuk 3 kun je lezen wat simulatie is en waarom we het gebruiken. Ook zal het programma dat voor deze opdracht gebruikt wordt, kort worden uitgelegd. In Hoofdstuk 4 wordt vervolgens uitgelegd hoe je jouw onderzoek zou kunnen aanpakken. Tot slot staat in Hoofdstuk 5 uitgelegd hoe je een simpele simulatie kunt uitvoeren.

Als je iets in de opdracht niet snapt, kijk dan eerst in de woordenlijst aan het eind (Hoofdstuk 6). Hierin worden de belangrijkste begrippen uit de opdracht uitgelegd. Ook kun je op de websites kijken die in Hoofdstuk 7 staan genoemd. Als het dan nog steeds niet lukt, kun je ook een mailtje sturen naar de contactpersoon voor deze opdracht op de Universiteit Twente: Marieke Huisman (Marieke.Huisman@ewi.utwente.nl).

3 Simuleren

Zodra we een computerprogramma hebben gemaakt, willen we weten of het goed werkt. Normaal gesproken controleren we dit door het programma te testen; we starten het programma en kijken simpelweg of het doet wat we er van

verwachten. Maar als het programma ingewikkelder wordt, of wordt gebruikt in een ingewikkelde of gevaarlijke situatie, dan is simpel testen niet goed genoeg meer.

Een andere techniek die we kunnen gebruiken om te controleren of een computerprogramma goed werkt is simulatie. Bij simulatie wordt de omgeving waarin het computerprogramma gebruikt gaat worden nagebootst op de computer, en in die virtuele omgeving wordt getest of het programma doet wat het moet doen. Bij deze manier van controleren letten we dus niet zozeer op hoe het programma werkt, maar vooral op hoe de omgeving reageert op het programma.

Voor deze opdracht gaan we een aantal stoplichten op een kruispunt simuleren. De virtuele omgeving die we gaan gebruiken is een kruispunt met stoplichten. Het computerprogramma dat we willen testen is het programma dat bepaalt wanneer de lichten op rood of groen staan. We willen graag dat er geen ongelukken kunnen ontstaan bij het kruispunt, en ook dat er geen files ontstaan. Om dit uit te zoeken moeten we te weten komen hoe de omgeving op de stoplichten gaat reageren. Maar omdat we niet willen riskeren dat er ongelukken gebeuren, kunnen we de stoplichten niet zomaar ergens neer zetten en kijken of ze werken. Daarom is dit een typisch voorbeeld waarin we simulatie willen gebruiken.

3.1 Roblox

Om de simulatie simpel te houden hebben we alvast een virtuele omgeving gemaakt. Voor deze omgeving hebben we gebruikt gemaakt van Roblox, zie `http://www.roblox.com`. Dit is een programma dat bedoeld is om internet spelletjes mee te maken (en te spelen), maar dat ook gebruikt kan worden voor een simpele simulatie. Het voordeel van Roblox is dat je straks zelf makkelijk aanpassingen aan de omgeving kunt maken. In Figuur 3.1 kun je zien hoe de virtuele omgeving er uit ziet.

Om Roblox te installeren moet je het volgende doen:

1. Ga naar `http://www.roblox.com/Install/Download.aspx` en klik op download Roblox.

2. Als het bestand klaar is met downloaden, open het dan en volg de instructies om Roblox te installeren.

3. Om later programma's van de Roblox website te kunnen downloaden en te bewerken heb je een Roblox account nodig. Op `https://www.roblox.com/Login/Default.aspx` kun je zo'n account aanmaken (bereikbaar door in het hoofdmenu naar MyRoblox te gaan).

3.2 Hoe werkt het kruispunt?

Het kruispunt dat we gaan simuleren voor deze opdracht is al beschikbaar via Roblox. Je kunt dit kruispunt op de volgende manier downloaden:

1. Eerst moet je Roblox installeren en een account registreren. In Hoofdstuk 3.1 kun je vinden wat je hiervoor moet doen.

2. Open Roblox studio. Als je Roblox op de standaardlocatie hebt geïnstalleerd, dan kun je Roblox studio vinden in het Start menu bij: Programma's, Roblox en dan Roblox studio. Bij Roblox studio zie je bovenaan in het venster een aantal knoppen, terwijl de normale Roblox browser alleen een adresbalk heeft.

3. Log in met je Roblox account.

4. Ga naar de gebruikerspagina van de gebruiker 'superfmt': `http://www.roblox.com/User.aspx?ID=13432773` (of via het People menu zoeken op 'superfmt').

5. Rechts bij 'Active Places' staat het kruispunt, klik op de edit-knop. Let op: alleen als je bent ingelogd op Roblox, en met Roblox studio naar de website gaat zul je de edit-knop zien.

6. Het kruispunt is nu geopend in de editor, en je kunt het opslaan via het menu 'File' en dan 'Save As'.

Het kruispunt wat je net hebt gedownload bevat al een voorbeeldsimulatie. Je kunt deze simulatie op de volgende manier starten:

1. Start de simulatie door rechtsboven op de groene startknop te drukken (▶). Het kan even duren (± 5 seconden) voordat de auto's beginnen met rijden.

2. Zodra je de simulatie hebt gestart, verschijnt er boven in beeld een zwarte balk waarin staat hoeveel auto's tot op dat moment de stoplichten hebben gepasseerd.

3. Je kunt de simulatie tijdelijk op pauze zetten met de gele pauzeknop (⏸).

4. Als je klaar bent, kun je de simulatie resetten met de roze stopknop ( ).

4 Onderzoeksvraag

Voor deze opdracht is het de bedoeling dat je zelf verschillende schema's gaat bedenken voor de stoplichten. Met behulp van simulatie kun je daarna onderzoeken hoe goed de verschillende schema's werken.

Eerst zal je dus een aantal schema's moeten bedenken die je wilt gaan vergelijken. Je kunt daar het schema uit het voorbeeld voor gebruiken, maar bedenk tenminste één ander schema (bijvoorbeeld een schema waarbij alle rijbanen één voor één groen licht krijgen). Deze schema's moet je dan gaan programmeren in Roblox.

Zodra je de schema's hebt geprogrammeerd, moet je simulaties gaan gebruiken om te onderzoeken welke van de schema's het beste werkt. Daarvoor start je een simulatie en die laat je een paar minuten (bijvoorbeeld 5) lopen. Rechtsonderin, in de rand van het scherm, loopt de tijd mee (voorafgegaan door een 't'). Na afloop noteer je hoeveel auto's de stoplichten hebben gepasseerd, en of er ongelukken zijn gebeurd. Deze simulatie herhaal je voor elk schema dat je hebt bedacht.

Ook kun je het aantal auto's op de weg variëren. Door voor elk schema het aantal auto's op de weg steeds hoger te maken, kun je onderzoeken hoeveel auto's je schema maximaal aankan.

Als je al deze gegevens hebt verzameld, dan kun je ze samenvoegen in een tabel. In de tabel staat dan per schema en per drukte op de weg hoeveel auto's de stoplichten hebben gepasseerd. Aan de hand van deze tabel kun je dan bepalen welk schema in welke situatie het beste is.

Het aantal auto's dat een stoplicht kan passeren binnen een bepaalde tijd wordt wel de verwerkingscapaciteit (of *throughput*) genoemd. Het optimaliseren van de verwerkingscapaciteit is niet alleen bij kruispunten van belang, maar bijvoorbeeld ook bij communicatie via een computernetwerk.

4.1 Het kruispunt uitbreiden

Het kruispunt dat standaard in Roblox beschikbaar is, is een simpel kruispunt. De auto's rijden alleen maar rechtuit, en het gaat om een simpel éénbaanskruispunt. Als je een uitgebreidere versie van het kruispunt wilt simuleren kan dat ook, maar dan moet je zelf eerst de uitbreidingen maken.

Als je het kruispunt wilt gaan uitbreiden is het belangrijk dat je eerst begrijpt hoe het simpele kruispunt werkt. Hiervoor moet je de bestaande scripts bekijken en begrijpen wat ze doen (in Hoofdstuk 3.2 kun je lezen hoe je het kruispunt in de editor opent, zodat je de scripts kunt lezen). Om de scripts beter te kunnen begrijpen, is elke functie voorzien van commentaar dat beschrijft wat de functie doet.

Zodra je begrijpt hoe het kruispunt werkt, kun je beginnen met je eigen uitbreiding. Je zou bijvoorbeeld kunnen denken aan de volgende uitbreidingen:

- Zorg ervoor dat de auto's niet alleen maar rechtuit gaan, maar soms ook afslaan.
- Maak van het kruispunt een meerbaanskruispunt.

- Als je een meerbaanskruispunt hebt gemaakt, dan kun je die uitbreiden met voorsorteervakken.
- Voeg fietsers of voetgangers toe aan het kruispunt.
- Zorg ervoor dat nieuwe auto's na een willekeurige tijd verschijnen, in plaats van na een vaststaande constante tijd.

5 Voorbeeld

Als voorbeeld laten we zien hoe je een simpel schema maakt en kan instellen voor de simulatie.

Eerst gaan we een schema bedenken dat we willen testen. Voor dit voorbeeld staan de lichten 5 seconden op rood, 1 seconde op geel en 2 seconden op groen. Omdat de auto's niet afslaan, kunnen de auto's die in tegenovergestelde richting rijden tegelijkertijd groen licht hebben. De kleur in de naam van de stoplichten slaat op de kleur van de auto's. Dus `witLicht` is bijvoorbeeld de naam van het stoplicht voor de witte auto's.

Een stoplicht kan worden ingesteld met de functie

$$_G.zetLichtManager(vertraging, roodtijd, geeltijd, groentijd, stoplicht).$$

De argumenten *roodtijd, geeltijd, groentijd* zijn de tijden (in seconden) die het stoplicht op rood, geel en groen moet staan. Bij het argument stoplicht, geef je het stoplicht mee waarvoor dit schema moet gelden. Een voorbeeldaanroep van de functie ziet er zo uit:

```
_G.zetLichtManager(0,5,1,2,witLicht)
```

Deze functie zorgt ervoor dat het stoplicht eerst *roodtijd* op rood staat, dan *groentijd* op groen en tot slot *geeltijd* op geel. Het eerste argument van de functie, *vertraging*, kunnen we gebruiken om de eerste roodtijd van het stoplicht te verlengen, waarmee we de schema's per stoplicht van elkaar kunnen laten verspringen. In ons schema hoort het stoplicht voor de witte en blauwe auto's direct op rood te springen, dus hier is geen extra vertraging nodig. Het licht voor de paarse en bruine auto's moet pas na 4 seconden op rood te springen (let op dat het licht aan het begin van het schema niet rood is omdat het dan op rood springt, maar omdat het dan nog steeds op de begintoestand rood staat). Daarom gebruiken we voor het stoplicht voor de paarse en bruine auto's een vertraging van 4 seconden.

De functies die we nodig hebben om dit schema in te stellen zien er dan zo uit.

```
_G.zetLichtManager(0,5,1,2,witLicht)
_G.zetLichtManager(0,5,1,2,blauwLicht)
_G.zetLichtManager(4,5,1,2,paarsLicht)
_G.zetLichtManager(4,5,1,2,bruinLicht)
```

Om dit schema in de simulatie te gebruiken moeten deze functies gekopieerd worden naar het goede script. Dit gaat als volgt:

1. Kopieer het bestand *VoorbeeldConfig* en geef het een nieuwe naam, bijvoorbeeld *config1*.

2. Zoek in de nieuwe configuratie naar functies die het stoplichtschema instellen, en vervang die door de bovenstaande functies.
3. Onderaan in het bestand zie je de regel *_G.voorbeeld_config = config* staan. Verander hier *voorbeeld_config* naar de naam van jouw configuratie, en laat de rest van de regel staan. Het ziet er dan bijvoorbeeld zo uit: *_G.config1 = config*.
4. Open tenslotte het script *Main* en vervang op de laatste regel *voorbeeld_config* door de naam van jouw configuratie, dus: *_G.config1()*

Je hebt nu je stoplichten ingesteld. Met de startknop kun je nu de simulatie starten en kijken hoe goed jouw configuratie werkt.

6 Woordenlijst

Configuratie Een configuratie is een verzameling instellingen. In deze opdracht bestaat de configuratie uit de instellingen van de stoplichten en de auto's.

Roblox Roblox is een programma waarmee op een makkelijke manier spelletjes kunnen worden gemaakt. Voor deze opdracht gebruiken we het om simulaties mee te doen.

Simulatie Bij een simulatie laten we een echte situatie nabootsen op een computer. Daarmee kunnen we dan met de computer onderzoeken wat er in het echt zou gebeuren in die situatie.

Virtuele omgeving De situatie die we met simulatie willen controleren, wordt ook wel de omgeving genoemd. De omgeving die we op de computer hebben nagemaakt heet dan de virtuele omgeving.

7 Handige links

Het kan zijn dat je tijdens het maken van deze opdracht ergens niet uit komt. In dat geval kun je een kijkje nemen op één van de onderstaande websites. Je kunt ook een mail sturen naar Marieke Huisman (Marieke.Huisman@ewi.utwente.nl) van de Universiteit Twente.

Roblox forum Het Roblox forum kun je vinden op `http://www.roblox.com/Forum`. Vooral in het Help Center zijn veel Roblox programmeurs actief. Is er dus iets waar je niet uit komt, dan kun je deze programmeurs om hulp vragen.

Roblox wiki De Roblox wiki kan je vinden op `http://wiki.roblox.com/index.php/Tutorials`. Op deze wiki zijn veel handleidingen en tutorials te vinden over hoe Roblox werkt. Dit is dus een goede plaats om te beginnen met zoeken als je iets wil veranderen aan de simulatie, maar niet goed weet hoe je dat moet doen.

Lua wiki De script taal die door Roblox wordt gebruikt heet Lua. Op `http://lua-users.org/wiki/TutorialDirectory` kun je een uitleg vinden van de meeste dingen die je met Lua kan doen.