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!

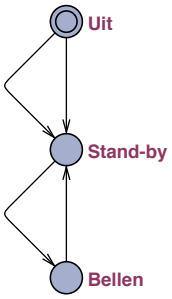
Een specificatie Om te kunnen bepalen of een programma zich gedraagt zoals wij willen, moeten we eerst heel precies opschrijven wat wij van het programma verwachten. Dit noemen we het specificeren van het programma.

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.

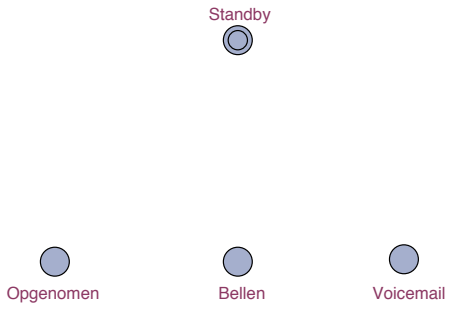
Omdat 'template' niet een erg handige naam is voor een model van een telefoon, willen we deze eerst veranderen. Dit kan je doen door bovenaan in het veld 'Name' een nieuwe naam in te voeren. Noem je model bijvoorbeeld 'Telefoon".

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.

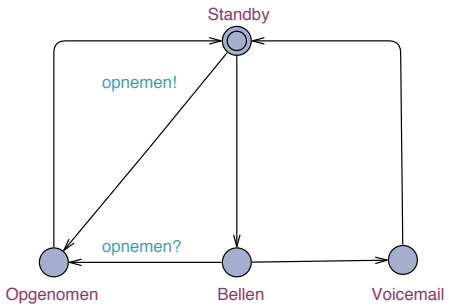
In Figuur 3 kan je zien hoe de overgangen eruit komen te zien. Let goed op waar ? en waar ! staat.

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.

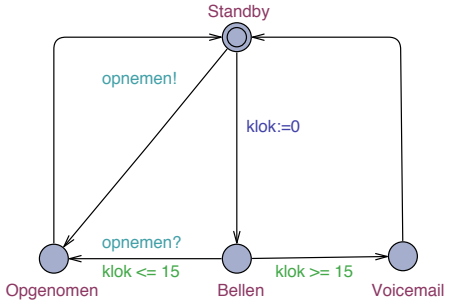
Vervolgens moeten we er voor zorgen dat de derde overgang (van 'Bellen' naar 'Opgenomen') alleen kan worden genomen als de klok op minder dan 15 seconden staat en dat de vierde overgang (van 'Bellen naar 'Voicemail') alleen wordt genomen als de klok op meer dan 15 seconden staat. Dit kunnen we doen door 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.