Een case-study naar het leren van real-time automaten

Marcel van der Made, 0814938

21 augustus 2012

Bachelorscriptie Radboud Universiteit Nijmegen, studiejaar 2011-2012

Begeleider: Frits Vaandrager

Met dank aan: Fides Aarts, Harco Kuppens

Samenvatting

Niet alleen mensen kunnen met apparaten leren omgaan door simpelweg te proberen, de zogenaamde trial-and-error methode. Ook computers hebben algoritmen om input te geven aan een *finite-state machine* en vervolgens het model hiervan te leren. Het aantal toestanden hiervan is eindig. Met de huidige technieken zijn deze algoritmen in staat modellen tot 30.000 toestanden te leren.

Wat nou als we deze algoritmen los willen laten op bijvoorbeeld een sensor? In elke seconde zijn een groot aantal metingen, en elk van deze metingen is te zien als een andere toestand. Het totaal aantal toestanden dat een dergelijk systeem heeft is dus veel groter dan de aantallen waar we in modellen mee willen werken. In deze case-study wordt gekeken of het mogelijk is een miniem *real-time* systeem zo aan te passen dat het aantal toestanden binnen de perken blijft en alsnog met het algoritme te gebruiken is. Vervolgens wordt gekeken waarom dit niet kan, of als het wel kan of de modellen nog wel op de originele modellen lijken.

Inhoudsopgave

1 Inleiding

Mensen leren vaak met apparaten omgaan door simpelweg maar wat te proberen, enkele knoppen indrukken en kijken wat er met het apparaat gebeurt door het gedrag ervan te observeren. Zonder een handleiding te raadplegen leren mensen zo omgaan met deze machines. We creëren in ons hoofd als het ware een model van het systeem, waarin we bijhouden wat er gebeurt wanneer we een bepaalde knop indrukken: in welke toestand de machine kan zijn en welke toestandstransities en outputs optreden bij bepaalde input.

Een dergelijke manier van leren bestaat er ook voor computers. Er zijn al, en er worden steeds nieuwe algoritmes ontwikkeld waarmee computers een model van een systeem kunnen leren, door input te genereren en output te observeren. De huidige technieken kunnen echter maximaal een model met 30.000 toestanden leren, terwijl de meeste systeemmodellen typisch meer dan 10^{1000} toestanden hebben.

Als we kijken naar een *timed automaton* dan neemt dit aantal toestanden nog meer toe. Elke tijdeenheid, neem bijvoorbeeld een miliseconde, zorgt namelijk voor een andere toestand. Elke seconde dat het systeem actief is komen er dan 1000 nieuwe unieke toestanden bij in het model! Wat houdt deze toestand precies in? Een computer ingenieur zou kunnen zeggen dat een transistor omslaat. Een natuurkundige zou kunnen zeggen dat de elektrische verbindingen op de printplaat ionen bevat, die op een bepaald moment een bepaalde lading heeft. In ons vakgebied van de ICT gaan we echter niet zo uitgebreid op het systeem in. We abstraheren van de fysieke verschijning en opbouw van de sensor, en beschouwen het systeem als een finite-state machine.

Sensors zijn als systeem een van de eenvoudigste soort *timed automata*. De sensor ontvangt een input en verwerkt deze input, en afhankelijk van de gelezen waarde geeft het systeem een output. Het aantal toestanden in een model van een dergelijk systeem is vooral afhankelijk van ontwerpkeuzes, zoals hoe lang moet de sensor wachten tussen twee metingen, hoeveel metingen worden er gedaan, hoe lang gaan de metingen door. Tijd is dan ook een belangrijke factor in een sensor systeem.

Een sensor heeft echter nog steeds het eerder genoemde probleem van tijd: het aantal toestanden wordt groter doordat tijd in het systeem een continue waarde is. Wat nou als we deze tijd discreet konden maken, en daarmee het aantal toestanden inperken? Dat zou het probleem oplossen, en de weg te openen om meer complexe *timed automata* te leren. We gaan dus proberen

een model te leren van een systeem waarvan we aannemen dat het een finite-state machine is. Om dit mogelijk te maken, wordt de tijd gediscretiseerd. Tijd zal niet meer continue oplopen, maar gesimuleerd worden met een input instructie. Hierbij moet wel rekening gehouden worden met tijdvoorwaarden, zoals de tijd tussen twee metingen. Voor x discrete tijdeenheden, komen er ook x toestanden bij.

Bij discretiseren verdwijnt er data. In dit geval verdwijnt er tijd. In Uppaal wordt een klok gebruikt om een niet-gespecificeerde tijdsduur te simuleren. Om dit toch duidelijk te simuleren is het belangrijk te weten welke waarde de klok heeft, en om hierover controle te hebben. Het is natuurlijk mogelijk om een reëel getal als klok te nemen en met elke tijdsverhoging deze klok 0,00...01 te verhogen om tijd realistischer te simuleren. Dan komen we echter bij het probleem dat dan het aantal toestanden van het model met deze klok onvoorstelbaar groot kan worden bij slecht gekozen constraints, zoals x $\geq$ 100.

Door te discretiseren verdwijnt er data. We nemen aan dat er op specifieke intervallen een handeling plaatsvindt. Mocht de sensor tussen deze intervallen een input krijgen, dan is niet duidelijk wat de sensor moet doen omdat dit niet hoort te gebeuren. We nemen daarom aan dat dit niet gebeurt, dat handelingen alleen voorkomen op de discrete intervallen. Bewijzen dat het continue model en het discrete model equivalent zijn ligt buiten de scope van dit onderzoek. Deze equivalentie nemen we aan.

In sectie 3 wordt uiteen gezet wat er precies onderzocht gaat worden. Sectie 4 behandelt omliggende theorie die relevant is aan het onderzoek. Hier staat ook beschreven wat precies gedaan wordt en hoe dit gedaan wordt. In sectie 5 worden alle resultaten samengevoegd en geanalyseerd, en vervolgens wordt in sectie 7 het onderzoek nog eens samengevat in een conclusie.

2 Verantwoording

2.1 Persoonlijke aanleiding

De aanleiding om hiernaar onderzoek te doen kwam van mijn begeleider Prof.dr. F.W. Vaandrager, die het Italia project is begonnen. Door een persoonlijke interesse voor embedded systems kwam ik al snel bij Frits terecht, die mij vertelde over het project. Het leek me interessant om iets toe te voegen aan een bestaand project, om kennis toe te voegen die gebruikt zal worden. Italia draait namelijk om het automatisch maken van model checking. Hiervoor moeten modellen eerst geleerd worden. Timed automata waren echter nog niet mogelijk omdat het aantal toestanden, zoals eerder gezegd, te groot werd voor de huidige technieken. Er is gerelateerd onderzoek gedaan naar dit onderwerp door Bengt Jonsson en Sicco Verwer. Het onderzoek van Jonsson richt zich vooral op de theorie van real-time systemen leren. Verwer houdt zich in zijn onderzoek bezig met passieve leeralgoritmen. Deze case-study is een eerste om te onderzoeken of het mogelijk is om *timed automata* zo aan te passen dat ze toch te leren zijn met actieve leeralgoritmen. Er wordt direct naar resultaten gezocht door te testen en daaruit conclusies te trekken.

2.2 Wetenschappelijke relevantie

Het Italia project [1] focust op het automatisch leren van complexe automaten. Deze automaten kunnen gezien worden als finite-state model. Als deze automaten via het model eenmaal geleerd zijn, kan men zich op de toepassing van *model inference technology* richten, en met name het testen van de systemen. Als een software component is geleerd, kunnen software-checking tools gebruikt worden om bijvoorbeeld beveiligingslekken te vinden. Ook wordt de model-based testing techniek gebruikt om automatisch test suites te genereren. Deze test suites worden onder andere gebruikt om te checken, (a) of er geen nieuwe fouten in een nieuwe versie van het component zitten (regression-testing), (b) of een alternatieve implementatie van een andere ontwikkelaar overeenkomt met een referentie implementatie, of (c) een nieuwe implementatie van verouderde software correct is.

Het leren van grote *state diagrams* is echter wel een probleem. Er is binnen Italia een recente doorbraak geweest om grote modellen te leren, in

samenwerking met prof. Jonsson van de Universiteit van Uppsala. Hierdoor was het mogelijk om modellen van realistische communicatieprotocollen te leren. Italia richt zich erop om deze techniek verder te ontwikkelen en toolsets te maken die volledig automatisch *state diagrams* leert, met maximaal 40 toestandsvariabelen.

Het idee om een sensor als een *timed automata* te leren, kwam naar aanleiding van het artikel van Bourke & Sowmya [2], waarin een infrarood sensor gemodelleerd met de tool Uppaal. Dit leidde tot de vraag of dergelijke systemen ook te leren zijn.

Real-time systemen worden vaak gerepresenteerd door een finite-state model met een groot aantal toestanden. Dit grote aantal toestanden is een belemmering om dergelijke systemen te leren. Als het mogelijk blijkt om continue tijd van een systeem om te zetten naar discrete tijd, waardoor dan het aantal toestanden afneemt, dan zal het ook mogelijk zijn om het leren van deze timed automata te automatiseren, doordat de belemmering van het grote aantal toestanden wordt weggenomen.

Het nut van het modelleren van het gedrag van een systeem, is dat software engineers in in grote lijnen dat in minder tijd betere software kunnen ontwikkelen. Met gedragsmodellen kunnen systemen gesimuleerd en geanalyseerd worden. Alle betrokkenen kunnen meedenken en deelnemen in het ontwikkelingsproces. Ze kunnen gebruikt worden om implementaties te testen en genereren, en zijn herbruikbaar.

In de praktijk is vaak het probleem dat software componenten geen of beperkte documentatie heeft. Italia geeft software engineers de mogelijkheid door volledig automatisch toestandsdiagrammen af te leiden door observaties, en vervolgens te testen met *black box reverse engineering*. Italia zal voornamelijk effectief zijn voor controle-geörienteerde systemen als embedded controllers en netwerk protocollen.

3 Probleemstelling

3.1 Onderzoeksvraag

De onderzoeksvraag die centraal staat in deze case-study is:

Is het mogelijk een extreem simpel real-time systeem te laten leren door Learnlib?

Deze case-study zal zich er dus op richten om te testen of dit mogelijk is, of waarom dit wel of niet mogelijk is.

3.2 Variabelen

- **extreem simpel** Een simpel systeem, is zo klein mogelijk, oftewel met zo min mogelijk toestanden. Een minimale functionaliteit impliceert een klein aantal toestanden. We zoeken dus een systeem dat weinig kan. We kiezen voor een infrarood-sensor. Deze geeft een hoog signaal, of een laag signaal af op een bepaald moment. Er gaat een input in, en er komt een input uit.
- **real-time** Een real-time systeem werkt met tijd. Op elk tijdstip verkeerd het systeem in een andere toestand, tijd is dus een continue waarde. Met betrekking tot de gekozen sensor betekent dit dat op elk tijdstip een signaal uitgelezen kan worden. Een tijdstip als trigger, en een binair signaal als output.

3.3 Methode

Om de onderzoeksvraag te beantwoorden, hebben we op de eerste plaats een sensor en een model daarvan nodig. Bourke en Sowmya [2] hebben in hun paper onderzoek gedaan naar een infrarood sensor. Ze hebben de sensor geanalyseerd en gemodelleerd in de Uppaal tool. Deze modellen zijn vrij beschikbaar, en daar maken we dankbaar gebruik van. Een uitgebreide beschrijving van deze modellen volgt in de volgende sectie.

Bourke en Sowmya modelleren het gedrag van de sensor aan de hand van twee modellen: een van de sensor, en een van de driver, waarmee de sensor communiceert. Om het gehele systeem te leren zullen we dus beide modellen moeten leren. Als learner is gekozen voor de tool LearnLib [3]. Voordat we

een model door Learnlib kunnen laten leren, moet het in een bepaald format komen te staan. Om dit te automatiseren is binnen het Italia project de tool Tomte ontwikkelt, die een concreet systeem omzet in een abstract systeem, dat geschikt is om met LearnLib te werken. Dit concrete model moet in de Uppaal syntax zijn gespecificeerd. Ook zijn er enkele eisen aan het model, voordat Tomte het model kan gebruiken: het model moet als een Mealy machine beschreven worden. Ook in de syntax moet een en ander aangepast worden, maar daarover meer in sectie 4.

Met behulp van Tomte en LearnLib kunnen we dus de modellen leren. We willen natuurlijk dat deze modellen equivalent zijn aan de originele modellen. Dit zullen we controleren met behulp van de CADP toolset. Mochten de modellen niet equivalent zijn, dan is er of iets fout gegaan in het modelleren naar Mealy formaat, of de modellen zijn niet leerbaar. Aan de hand van dit resultaat gaan we analyseren waarom het wel of niet mogelijk is om deze modellen te leren, en daarmee onze onderzoeksvraag beantwoorden.

4 Theoretisch kader

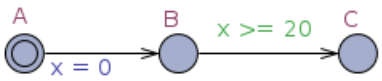
4.1 Toolsets

Uppaal Als modelleertool is gebruik gemaakt van Uppaal. Uppaal is een grafische omgeving waarin men real-time embedded modellen kan ontwerpen, simuleren en verifiëren. Voor een gedetailleerde beschrijving van Uppaal is de bijgeleverde handleiding meer dan geschikt. Ook zijn er vele tutorials over te vinden. Voor nu houden we het bij de basis.

De Uppaal syntax komt overeen met de syntax die gebruikt wordt in de automatentheorie: Een toestand wordt aangegeven met een cirkel. De begintoestand met een cirkel in de cirkel. Transities van een toestand naar een andere toestand worden aangegeven met een gerichte pijl; deze transities kunnen onder andere *guards* en updates bevatten.

In Uppaal is het ook mogelijk om tijdconstraints toe te voegen aan een model. Hiervoor is er een nieuwe type variabele ingevoerd genaamd clock. Een clock in een model wordt gedeclareerd met clock x;

In Uppaal nemen transities zelf geen tijd in. Er kan alleen tijd verstrijken als een model in een bepaalde toestand wacht. Om dit te modelleren in Uppaal worden er twee transities gebruikt. Stel we hebben een model met toestand A, B en C. De begintoestand is A, en de automaat moet 20 tijdeenheden wachten in B voordat hij naar C mag. Dit geven we aan door in de transitie van A naar B, de waarde van x op 0 te zetten, en de transitie van B naar C krijgt de guard $x \geq 20$. Deze guard zorgt ervoor dat de transitie alleen kan plaatsvinden als x de waarde 20 heeft, of meer. Zie figuur 1



Figuur 1: Voorbeeld

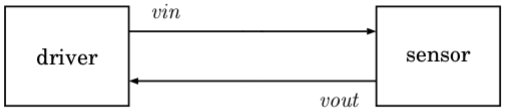
In de figuur is te zien dat bij de transitie van toestand A naar B, aan de klok x de waarde 0 wordt toegekend. De transitie van B naar C heeft een guard van $x \geq 20$, waardoor in het model wordt afgedwongen dat de transitie van B naar C pas genomen kan worden als de klok x de waarde 20 of hoger heeft. Op deze manier kan tijdsverloop in een model gemodelleerd worden.

CADP Construction and Analysis of Distributed Processes (CADP [4]) is een toolset die wordt gebruikt bij het het ontwerp van communicatieprotocollen en systemen. De toolset heeft vele functionaliteiten, maar degene waar in deze case-study gebruik van wordt gemaakt is een equivalentiechecker, die controleert of twee finite-state modellen (zonder klok) gelijk zijn aan elkaar. Hier kan als input een Uppaal model worden aangeleverd, of een .DOT file, wat als output wordt geleverd van ons leeralgoritme. Bij het checken wordt aangegeven of de modellen equivalent zijn, en als ze dit niet zijn wordt een tegenvoorbeeld gegeven in de vorm van een trace.

4.2 De sensor

De sensor die wordt gebruikt is de Sharp GP2D02 sensor. Het is een sensor die over het algemeen geïntegreert worden in embedded systemen. Het is een electro-optisch systeem die een toestand meet in de echte wereld en deze omzet in digitale waarden, die gebruikt worden in een groter systeem. Het is een relatief simpel apparaat en daardoor goed bruikbaar voor het onderzoek. Bij het beschrijven van het apparaat zal blijken dat ook tijdconstraints een factor spelen, en dat is precies wat we willen hebben.

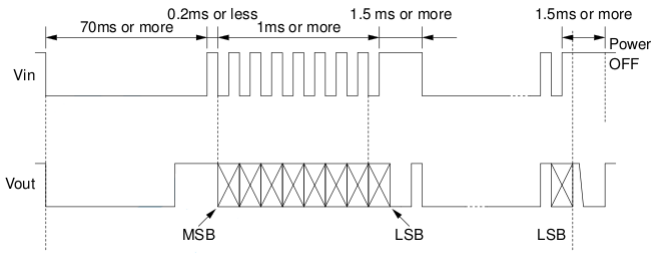
Overzicht De fysieke verschijning van de sensor is een klein ($14 \times 29 \times 14mm$) rechthoekig doosje met vier elektrische aansluitingen: Voltage, aarde, input*(vin)* en output *vout*. De sensor meet de afstand naar objecten door infrarood licht vanuit een diode te schijnen, wat wordt teruggekaatst door het voorwerp. De afstand van de sensor tot het object wordt geschat door te meten waar de teruggekaatste straal in een detectiegebied is. Dit detectiegebied is vlak naast de diode geplaatst. Er worden vervolgens meerdere metingen gestart en een 8-bit schatting wordt als output gegenereerd. Meer gedetailleerdere werking staat in de handleiding van de sensor [9] .



Figuur 2: Sensor toepassing

In een systeem, is de sensor gekoppeld aan een tweede apparaat, in ons geval de *driver*, zoals in figuur 2. De driver regelt het signaal op *vin* en de sensor regelt het signaal op *vout*. Door Bourke & Sowmya zijn meerdere manieren voor de modellen voorgesteld. Een systeem met driver en sensor samen, een model waar de driver in assembly-taal is gemodelleerd, en een methode waar de driver en sensor als afzonderlijke componenten functioneren. We hebben gekozen om te werken met de laatste methode. Dit om het aantal toestanden per model te beperken. Ook is het eenvoudiger twee kleine modellen te leren dan een groot model.

Timing Diagram In de handleiding van de sensor staat beschreven hoe de sensor precies werkt. Voor een beter inzicht in de modellen wordt dat hier ook uitgelegd.



Figuur 3: Timing diagram van de sensor [Sharp Coorporation 1997]

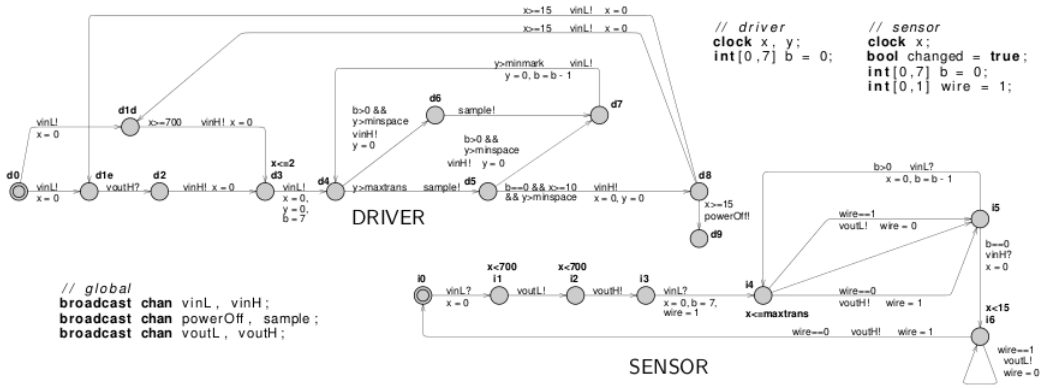
Het tijdsverloop diagram in figuur 3 is direct uit de handleiding overgenomen. De bovenste *Vin* lijn is het input signaal van de sensor, en de onderste *Vout* lijn wordt beschouwd als het output signaal van de sensor.

De eerste dalende lijn op *Vin* initiëert een afstandsmeeting die ten hoogste 70ms mag duren. Vervolgens wordt een serie pulsen toegepast om data uit de sensor te klokken. Als laatste moet er 1.5ms gewacht worden voor het proces wordt herhaald of wordt getermineerd.

Deze beide mogelijkheden zijn gemodelleerd in de figuur. In beide gevallen begint de vertakking met een pulse op *Vin* met het label *1.5ms or more*. De eerste puls is weer het begin van een nieuwe afstandsmeting. De tweede puls, met het label *Power OFF* wordt geïnterpreteerd als de keuze om de sensor uit te schakelen. Het onderscheid tussen deze vertakkingen wordt aangegeven door een stippellijn op het *Vin* signaal.

Tussen de twee signalen lopen vier verticale gestreepte lijnen. Deze lijnen geven synchronisatie momenten aan op de signalen. Deze momenten zijn van links naar rechts: begin een afstandsmeting, een verandering in *Vout* voor het meest significante bit (MSB), evenzo voor het minst significante bit (LSB), en vervolgens terugkeren naar een laag signaal op *Vout* na een specifieke vertraging. Als voorbeeld, in het model van de sensor is te zien, dat de toestand i2 pas na 700 clock-stappen (1ms = 10clockstappen in het model) verlaten kan worden, en dat dan de synchronisatie 'VoutH' wordt gegeven. Dit komt overeen met het timing diagram, waar het *Vin* signaal hoog wordt zodra de 70ms voorbij zijn.

De pulsen met een kruis in het *Vout* signaal, geven aan dat de data zowel een laag als hoog signaal kan zijn, en dat dit afhangt van de gemeten waarde van de sensor. Deze lezing heeft in de modellen het label 'sample'.



Figuur 4: De modellen van driver en sensor.

Modellen Doordat de modellen apart van elkaar worden behandeld is het gedrag van de individuele componenten beter te analyseren. Echter, de modellen werken wel parallel en zijn dus betekenisloos zonder elkaar. De modellen, in Uppaal syntax, staan in figuur 4. In deze subsectie wordt het gedrag van beide modellen bekeken en geanalyseerd.

De driver is verantwoordelijk voor het signaal op *vin*, net zo is de sensor verantwoordelijk voor *vout*. Door onderling te communiceren via kanalen in Uppaal, kunnen de componenten samenwerken. Dit betekent dat als het ene systeem pas verder zou kunnen als het via een specifiek kanaal bericht heeft

gekregen van het andere model dat het verder mag.

De modellen zijn ontworpen naar het timing diagram zoals afgebeeld in figuur 3. De modellen hebben synchronisatie momenten die wachten of het *Vin* of *Vout* signaal hoog of laag is, en op transities en toestanden worden beperkingen toegelegd, zodat aan alle tijdsvoorwaarden wordt voldaan.

Nu bekend is met welke modellen gewerkt wordt, kunnen we verder kijken wat nodig is om deze modellen te leren.

4.3 Mealy Machines

Uit de praktijk is gebleken dat de Mealy Machine een uitermate geschikt modelleer formalisme is voor reactieve systemen. Een Mealy machine is een specifiek soort automaat die onderscheid maakt tussen een input en een output alfabet. Een van de belangrijkste eigenschappen is dat een Mealy machine volledig gedefiniëerd is voor de input symbolen. Vanuit elke toestand is dus een transitie voor elke mogelijke input. Daarnaast is het gevolg van een input uniek bepaald, ook wel input determinisme genoemd.

Mealy machines lijken veel op deterministische finite-state machines: Mealy machines kunnen gezien worden als een deterministische finite-state automaat over de vereniging van het input alfabet en een output met slechts één afwijzende toestand. Ze hebben dus een partieel gedefiniëerde transitierelatie. Er wordt in een Mealy machine echter geen onderscheid gemaakt in afwijzende toestanden en accepterende toestanden. Het onderscheid is te vinden in de output van *runs*.

Formeel nemen we aan dat er een verzameling van input acties Σ is. En net zo nemen we aan dat er een verzameling van outputs Ω is. We refereren naar rijen van inputs (of outputs) met $w = \alpha_1..\alpha_n$ waar $\alpha_i \in \Sigma$ als woorden, die we zowel kunnen concateneren als splitsen in prefixes en suffixes, op dezelfde manier als in de talentheorie: we noteren $w = uv$ om aan te geven dat w opgesplitst kan worden in prefix u en suffix v. Dezelfde notatie gebruiken we om aan te geven dat u en v geconcateneerd kunnen worden tot w. Om nadruk te leggen op het concateneren, schrijven we ook wel $u \cdot v$, en het lege woord noteren we als ϵ.

Nu zullen we Mealy machine definiëren

Definitie 1. Een *Mealy machine* is gedefiniëerd als een tuple $\mathscr{M} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$ waar

- S is een eindige niet-lege verzameling van toestanden,
- $s_0 \in$ S is de begintoestand,
- Σ is een eindig input alfabet,
- Ω is een eindig output alfabet,
- $\delta : S \times \Sigma \to S$ is de transitiefunctie, en
- $\lambda : S \times \Sigma \to \Omega$ is de outputfunctie

Intuïtief gezien loopt een Mealy machine door de toestanden $s \in S$, en wanneer een input symbool $a \in \Sigma$ wordt toegepast, dan gaat de machine naar een nieuwe toestand in overeenkomst met $\delta(s, \alpha)$ en produceert een output in overeenkomst met $\lambda(s, \alpha)$. □

Nu we de Mealy machine en de modellen die we gaan gebruiken gedefiniëerd hebben, kunnen we onze sensor en driver modellen als een Mealy machine gaan definiëren, om daarna de volgende stap nar een test systeem te kunnen maken.

De sensor

Om de tijd discreet te maken in het model van de sensor, behandelen we de tijd niet als de gebruikelijke interne klok in het systeem, maar als een normale toestandsvariabele. Omdat elke toestand afhankelijk is van de toestandsvariabelen zullen we een functie moeten invoeren die van een toestand de correcte waarde toekent aan deze toestandsvariabelen. Deze functie noemen we f, en definiëren we intuïtief als volgt:

f : toestandsvariabele $\to$ waarde

De waarde is afhankelijk van de variabele. In het geval van de klok die we gebruiken is de waarde een natuurlijk geval ($\mathbb{N}$). De verzameling toestanden S bestaat dus niet uit enkel de toestandsnaam, maar uit een tuple (s_i, f) zodanig dat f aan alle toestandsvariabelen in s_i een waarde toekent. De individuele toestandsvariabelen zijn als volgt gedefiniëerd:

$f(Clock) \rightarrow \mathbb{N}$

$f(ClockX) \rightarrow \mathbb{N}$

$f(ClockY) \rightarrow \mathbb{N}$

$f(b) \rightarrow \mathbb{N}$

$f(minspace) \rightarrow \mathbb{R}$

$f(maxtrans) \rightarrow \mathbb{R}$

$f(minmark) \rightarrow \mathbb{R}$

$f(wire) \rightarrow 0, 1$

De precieze definitie van de functie f is dus afhankelijk van het aantal variabelen die in een specifieke toestand bestaan.

Nu kunnen we de definitie geven van de machines:

Sensor Het gedrag van de sensor kunnen we specificeren als de *Mealy machine* $\mathscr{M}_s = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$ waar

- S = $\{(i0, f), (i0a, f), (i0b, f), (i1, f), (i1a, f), (i1b, f), (i2, f), (i2a, f), (i2b, f), (i3, f), (i3a, f), (i3b, f), (i4, f), (i4a, f), (i4b, f), (i5, f), (i5a, f), (i5b, f), (i5c, f), (i5d, f), (i6, f), (i6a, f), (i6b, f), (i6c, f)\}$
- s_0 = (i0,f)
- Σ = VinL, VinH, Tick
- Ω = VoutL, VoutH, Ok

De transitie- en output-functies zijn gedefiniëerd zoals te zien in figuur 5. Alle toestanden die zijn toegevoegd bij de overgang van toestanden, de tussentoestanden, zijn genoemd naar de toestand van waaruit de transitie plaatsvind, en hebben een letter als subscript gekregen. De toestand namen zonder subscript komen dus overeen met de gelijknamige toestand in de finite-state automaat zoals beschreven in sectie 4.2. □

De driver

Evenals bij de sensor, hebben we bij de driver ook een extra functie f die we gebruiken om aan toestandsvariabelen een waarde toe te kennen. Deze functie zullen we wederom f noemen.

De definitie van de driver ziet er als volgt uit:

Driver Het gedrag van de driver kunnen we specificeren als de *Mealy machine* $\mathscr{M}_s = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$ waar

- S = $\{(d0, f), (d0a, f), (d1, f), (d1a, f), (d3, f), (d3a, f), (d3b, f), (d4, f), (d4a, f), (d4b, f), (d5, f), (d5a, f), (d7, f), (d7a, f), (d7b, f), (d8, f), (d8a, f), (d8b, f), \}$
- s_0 = (d0,f)
- Σ = VoutL, VoutH, Tick
- Ω = VinL, VinH, Ok, Sample

De transitie- en output-functies zijn gedefiniëerd zoals te zien in figuur 5. Evenals bij de sensor zijn tussentoestanden genoemd naar de toestand vanuit waar ze bereikt worden. De toestanden zonde subscript komen ook weer overeen met de gelijknamige toestanden in de finite-state modellen van sectie 4.2. □

4.4 Constructie van de machines

Nu de Mealy modellen van de originele modellen zijn gedefiniëerd, zullen we in deze sectie uitleggen hoe tot deze modellen gekomen is. Als input voor LearnLib is namelijk een Mealy machine nodig, maar die moet ook aan enkele syntaxvoorwaarden voldoen. Input en output van een transitie kan niet bij een overgang gerepresenteerd worden, en gebeurt dus met een tussentoestand. Stel we willen van toestand A naar C, met input p en output q. In toestand A treedt dus input p op. Dan is eerst een overgang van A naar tussen toestand B, en van deze tussentoestand is er een overgang naar toestand C met output q. Dit is nodig omdat deze inputs/outputs van de synchronisatie-variabele in Uppaal wordt gelezen. Deze variabele kan maar een waarde hebben, dus is gekozen voor deze aanpak. Alle namen van deze inputs en outputs beginnen respectievelijk met een I en een O, en eindigen met haakjes. Een andere

belangrijke voorwaarde is dat vanuit elke niet-tussentoestand een transitie is met elke mogelijke input. In het geval dat deze input in het oorspronkelijke model geen output heeft, dan wordt als output *Ok* gebruikt. Een andere voorwaarde is dat het model deterministisch moet zijn. Daarnaast zijn er nog enkele syntax voorwaarden, die verder beschreven staan op de website van Italia project [1].

Een van de problemen die aangepakt moest worden is de klok die gebruikt werd in het oorspronkelijke model. Als de automaat moet wachten in een toestand tot de klok een bepaalde waarde x heeft, dan wordt elke tijdseenheid tussen x_0 en x als nieuwe toestand gezien door LearnLib. Aangezien de klok een continue waarde is, is er besloten om in plaats van de ingebouwde klok een nieuwe integervariabele te definiëren, die als klok dient. Hierdoor wordt de klokwaarde discreet, waarmee het aantal toestanden van het door LearnLib geleerde model afneemt. Ook zorgt dit ervoor dat LearnLib ook daadwerkelijk met het model kan werken. Door een Tick input te geven, wordt de klok verhoogd in het model. Zo blijft de controle van de klok in eigen handen, en kan het model geleerd worden door LearnLib.

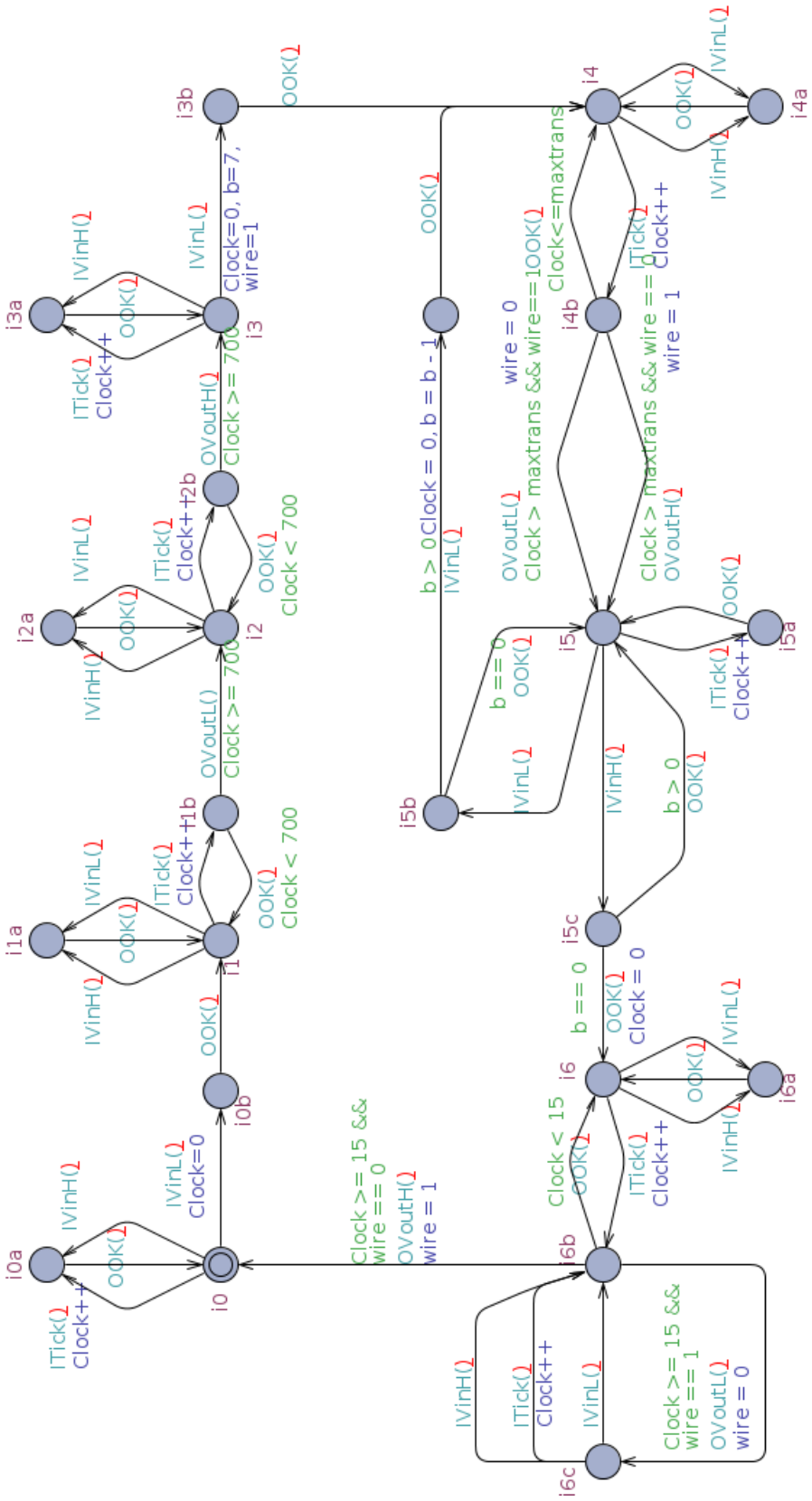
We zullen nu per model bekijken hoe het is opgebouwd. Per model zijn enkele keuzes gemaakt om het model kleiner te maken. Dit is noodzakelijk om aan de syntax voorwaarden van LearnLib te voldoen. In de modellen is de toestand met suffix *a* een tussentoestand voor inputs die geen betekenis hebben. De toestanden met suffix *b* zijn meestal tussentoestanden die overeenkomen met de transities in het oorspronkelijke model.

Mealy sensor Beginnend bij de sensor.

In figuur 5 is de sensor als Mealy machine afgebeeld, reeds met de syntax die LearnLib nodig heeft als input. Het sensor model heeft de minste problemen met omzetten naar een Mealy machine. Er was een situatie waarin non-determinisme voorkwam, namelijk de transities van toestand i4 naar i5 in het originele model (figuur 4). Deze transitie geeft een verandering van het signaal aan: een hoog signaal wordt een laag signaal, en een laag signaal wordt een hoog signaal. Dan is er nog een transitie waarin niets gebeurt, als er geen verandering in het signaal optreedt. Deze drie transities van i4 naar i5 zorgden voor non-determinisme. Er is gekozen deze laatste transitie weg te laten. Deze transitie van i4 naar i5 vindt plaats bij het *samplen* van het sensor signaal, om specifiek te zijn, het geeft de pulsen die de driver aangeeft wanneer het een sample kan nemen. Er wordt dus afgedwongen dat de pul-

sen in constant tempo doorgaan, zonder dat een puls langere tijd hetzelfde signaal (hoog signaal of laag signaal) behoudt. Het weglaten van de derde transitie verandert dus niet het gedrag van de sensor, slechts de tijdsduur tussen de samples wordt beïnvloed.

Figuur 5 is een rechtstreekse vertaling van het originele Uppaal model naar een Uppaal model dat voldoet aan de eisen die LearnLib stelt aan een inputmodel. Het model lijkt groter omdat nu alle transities uit de definitie van de Mealy machine aanwezig zijn. Dat de modellen equivalent zijn (op de klok na, die nu discreet is in plaats van continu) is te zien door alle ix toestanden te vergelijken in beide modellen. In figuur 4 treedt bijvoorbeeld tussen i2 en i3 een voutH op als output, en dit gebeurt pas als de klok kleiner is dan 700. Dit is ook terug te vinden in figuur 5: in toestand i2 gaan de transities die geen gevolgen hebben, IVinH() en IVinL() naar toestand i2a, en vervolgens met de betekenisloze output OOK() weer terug naar i2. De ITick() input zorgt ervoor dat de automaat naar toestand i2b gaat, waarmee de Clock variabele verhoogt wordt, en gecheckt wordt of de Clock groter is dan of gelijk is aan 700. Is dit niet het geval, dan gaat de automaat terug nar i2, is dit wel het geval dan gaat de automaat naar i3 en geeft de output OVoutH() overeenkomstig figuur 4. Op eenzelfde manier is voor elke toestand overgang te concluderen dat de modellen equivalent zijn. Ook wordt dit later nog aangetoont met de CADP toolset in de onderzoeksresultaten.



Figuur 5: De sensor als Mealy machine.

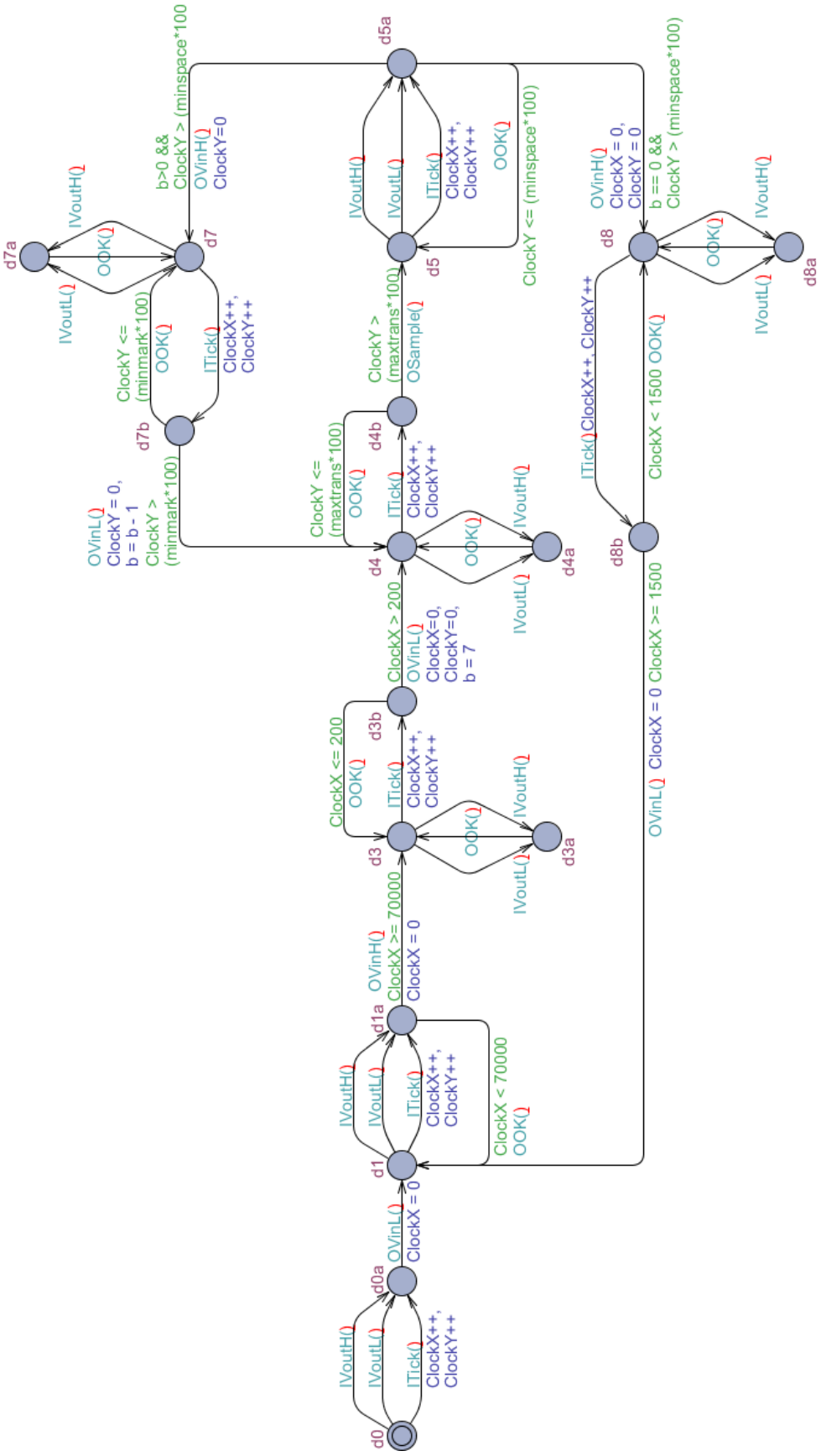
Mealy driver Dan de driver

Bij de driver zijn er meer problemen. Zo is er op drie plaatsen non-determinisme, namelijk in toestanden d0, d4 en d8. De twee routes van d0 naar d3 zijn volgens Bourke & Sowmya [2] twee manieren om het timing diagram te interpreteren. Een van beide routes kiezen zou een specifieker model van de sensor beschrijven. Een van beide routes nemen doet dus niet teniet aan het gedrag, het maakt slechts een specifieker model, dus dit levert geen problemen op. De route om mee te werken is de bovenste route: d0 - d1 - d3, omdat hier een tijdsbeperking in voorkomt. Tijd is het onderwerp van deze case-study dus dit leek de meest logische keuze.

Toestand d4 is het begin van de twee loops d4-d5-d7-d4 en d4-d6-d7-d4 (Zie figuur 4). In deze loops worden samples van de sensor uitgelezen. Het verschil tussen de loops is dat de instructies in omgekeerde volgorde staan. De eerste transitie uit de ene loop heeft dezelfde instructies als de tweede transitie uit de tweede loop en vice versa. Dit is waarschijnlijk omdat er slechts acht samples genomen worden, en in d5 wordt dan beslist of nog een keer gesampled wordt of dat de machine naar toestand d8 gaat (als variabele b gelijk is aan 0). Het weglaten van d4-d6-d7 dwingt de specifiekere volgorde af van eerst een sample uitvoeren en dan guards controleren, maar dit is niet van invloed op het gedrag van de sensor. Dit is eveneens specifieker maken van de volgorde van het gedrag, niet een verandering van het gedrag zelf.

Toestand d8, de laatste toestand met non-determinisme, kan ervoor kiezen om na 1.5ms nog een meting te doen (en dus het hele model nog een keer doorlopen), of om de sensor uit te schakelen. Wij hebben ervoor gekozen om de optie die de sensor uitschakelt weg te halen. Als de driver in de *PowerOff* toestand zou komen, zou de sensor ook niet meer verder kunnen doordat er geen synchronisaties meer zouden optreden. Gaat de driver wel door, dan blijft de sensor ook actief. Het is dus een logische keuze om de *PowerOff* toestand weg te halen, en beide modellen actief te houden bij het simuleren van de systemen. In LearnLib zit een parameter waarmee in te stellen is hoeveel toestanden er bekeken zullen worden, dus dat de modellen geen eindtoestand hebben zal geen probleem opleveren.

Net als de sensor zal in de onderzoeksresultaten met CADP aangetoond worden dat de modellen in LearnLib syntax en het Uppaal model van Bourke& Sowmya equivalent zijn. Ook is bij de driver, elke dx toestand uit figuur 6 te vergelijken met dezelfde dx uit figuur 4. Hieruit zal volgen dat beide modellen hetzelfde gedrag vertonen.



Figuur 6: De driver als Mealy machine.

5 Onderzoeksresultaten

De resultaten van de tests zullen in een tabel weergegeven worden per model. De tabel begint met de tijd-constraints uit de modellen. In de figuren 7 en 8 is te zien welke constraint waar aanwezig is in het model. Deze constraints zijn van invloed op het aantal toestanden van het geleerde model. De kolom met *Aantal states geleerd* geeft aan hoeveel toestanden het geleerde model heeft. Als dit getal 1 is, dan is er geen goed model geleerd.

De kolom *equivalent* geeft aan of het geleerde model en het origineel equivalent zijn volgens de CADP toolset. In beide modellen die vergeleken worden zijn de tijdconstraints gelijkgesteld. De parameters in LearnLib worden per model behandeld. De variabelen minSpace, maxTrans en minMark zijn allen gedeclareerd als 0, en zijn bedoeld om het gedrag van de sensor te fijn-tunen.

5.1 Driver resultaten

De resultaten van het onderzoek over de driver staan in de volgende tabel. In figuur 7 is nogmaals het model van de Mealy Driver te zien, dit keer niet met de standaard constraints, maar de variabelen A, B en C, om aan te geven welke transities variabel zijn. De hoofdletter B die een onderzoeksvariabele weergeeft is een andere dan de kleine b die in het model een teller bijhoudt.

De minTraceLength parameter van LearnLib kan het best hoger genomen worden dan het minimaal aantal toestanden dat doorlopen wordt bij het volledig doorlopen van het model. In het model van de driver zijn er in de kortste route 12 gewone transities, 7 keer een loop van 6 transities = 42, en de variabelen: 2A-2, 2B, 2C-2. Het minimaal aantal toestanden dat doorlopen wordt om het model volledig door te lopen is:

2A + 2B + 2C + 50

De originele waardes van de variabelen in de sensor zijn: A= 700, B = 2, C = 15. Om de grenzen op te zoeken van LearnLib wordt langzaam opgebouwd naar deze waarden om een beter overzicht te krijgen wat het programma doet.

In de tabel staan enkele opvallende resultaten. Zoals te zien blijken de eerste drie modellen niet alleen geleerd te worden, maar ook nog eens correct geleerd. Het eerste model heeft volgens de formule 4 + 4 + 4 + 50 = 62

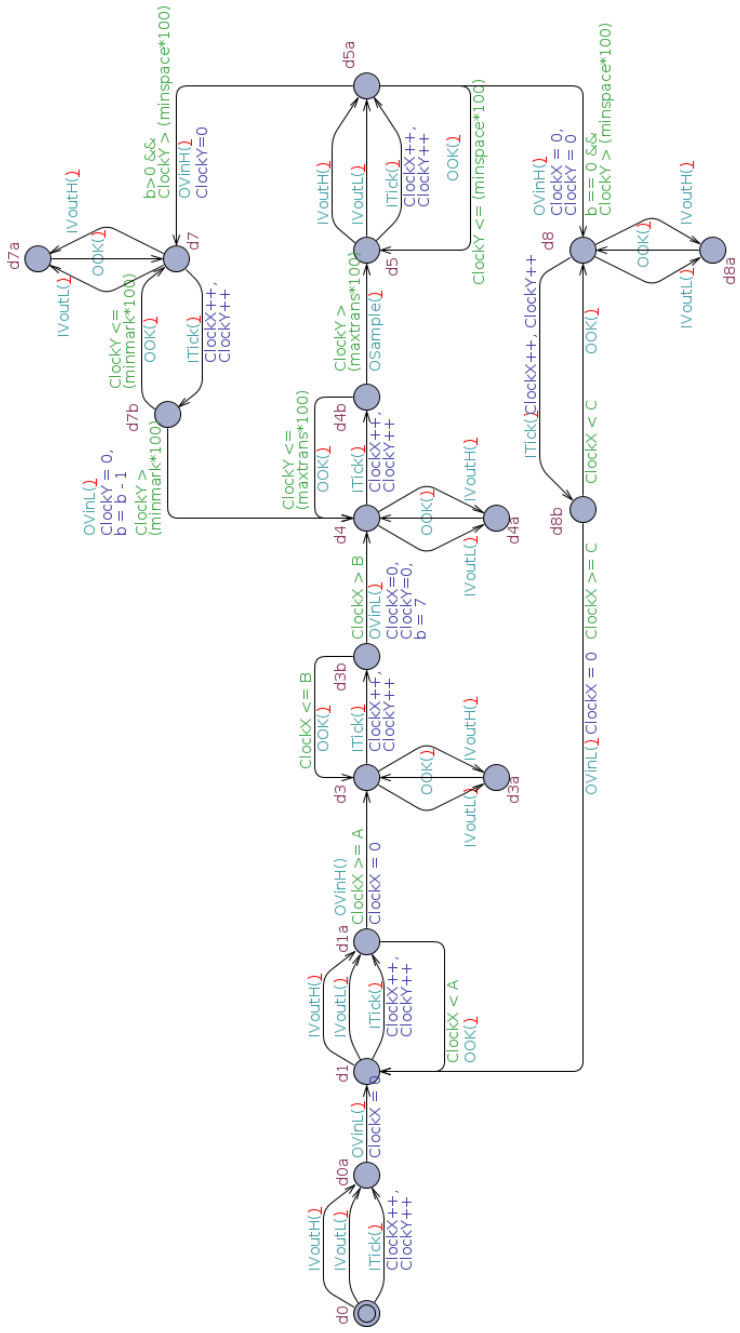
Tabel 1: Testresultaten van het driver model

Model	A	B	C	Aantal toestanden geleerd	equivalent
1	2	2	2	31	ja
2	2	2	15	44	ja
3	70	2	15	112	ja
4	700	2	15	0	nee

transities, precies het dubbele van het aantal toestanden. Net zo heeft het tweede model $4 + 4 + 30 + 50 = 88$ transities. En het derde model $140 + 4 + 30 + 50 = 224$ transities. Het vierde model, dat niet correct geleerd wordt, zou $1400 + 4 + 30 + 50 = 1484$ transities en dus 742 toestanden moeten hebben. Dit is niet het geval. 700 is blijkbaar een te hoge waarde voor LearnLib om mee om te kunnen gaan. Er zijn dan ook geheel geen outputbestanden gegenereerd om mee te testen. De resultaten van de voorgaande run waren nog aanwezig, wat bevestigd is door CADP.

Een mogelijke verklaring hiervoor is dat het aantal toestanden dat geleerd moet worden te groot is. Een meer simpele verklaring is dat de parameters van LearnLib niet goed gekozen zijn. Deze zijn echter met verschillende waarden gevariëerd, tot grote hoogte. Dit is dus onwaarschijnlijk.

Aannemelijker is dat LearnLib niet met dergelijke grote waardes bij dit model kan omgaan. Dit kan echter niet zomaar gezegd worden zonder meer tests uit te voeren op andere (simpele) systemen. Ook andere trucs zouden wellicht nog bedacht/toegepast kunnen worden om het aantal toestanden van real-time modellen te verminderen.



Figuur 7: De driver met tijdvariabelen A, B, C

Tabel 2: Testresultaten van het sensor model

Model	A	B	Aantal toestanden geleerd	equivalent
1	2	2	8	ja
2	2	15	21	nee, op diepte 14
3	70	15	89	nee, op diepte 14
4	700	15	0	nee

5.2 Sensor resultaten

De volgende tabel bevat de resultaten van de tests met het model van de sensor. In figuur 8 is nogmaals het model te zien met op de twee plaatsen waar tijd een rol speelt, de variabelen A en B. Ook hier is de hoofdletter B een andere variabele dan de letter b.

Bij de sensor zijn er in totaal 15 transities, met een 7-voudige loop met 5 transities, en de guards met 2A-2 en 2B-2. Het totaal aantal toestanden dat doorlopen moet worden komt dan uit op:

2A + 2B + 44

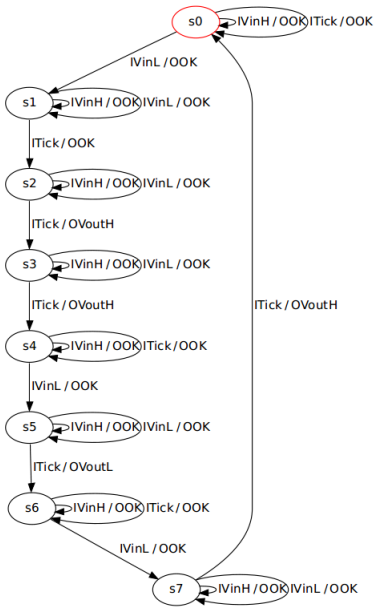
De originele waardes van de variabelen zijn A=700 en B = 15. Ook hier zullen we langzaam de waardes van de variabelen verhogen tot de gewenste waarde is bereikt.

Ook bij de sensor komen interessante resultaten naar voren. Model 1 heeft 8 toestanden met 52 transities, en is het enige model dat correct wordt geleerd. Model 2 heeft 21 toestanden met 78 transities, en model 3 heeft 89 toestanden met 214 transities. Dit wijkt totaal af van de regelmaat van de driver. Model 1 wordt correct geleerd, in tegenstelling tot model 2, 3 en 4. Model 2 en 3 wordt een tegenvoorbeeld voor gevonden met CADP, op diepte 14. Dit betekent dat de overgang van toestand i5c naar i6 een fout oplevert, wat niet hoort te gebeuren. Model 4 produceert net als de driver helemaal geen resultaat.

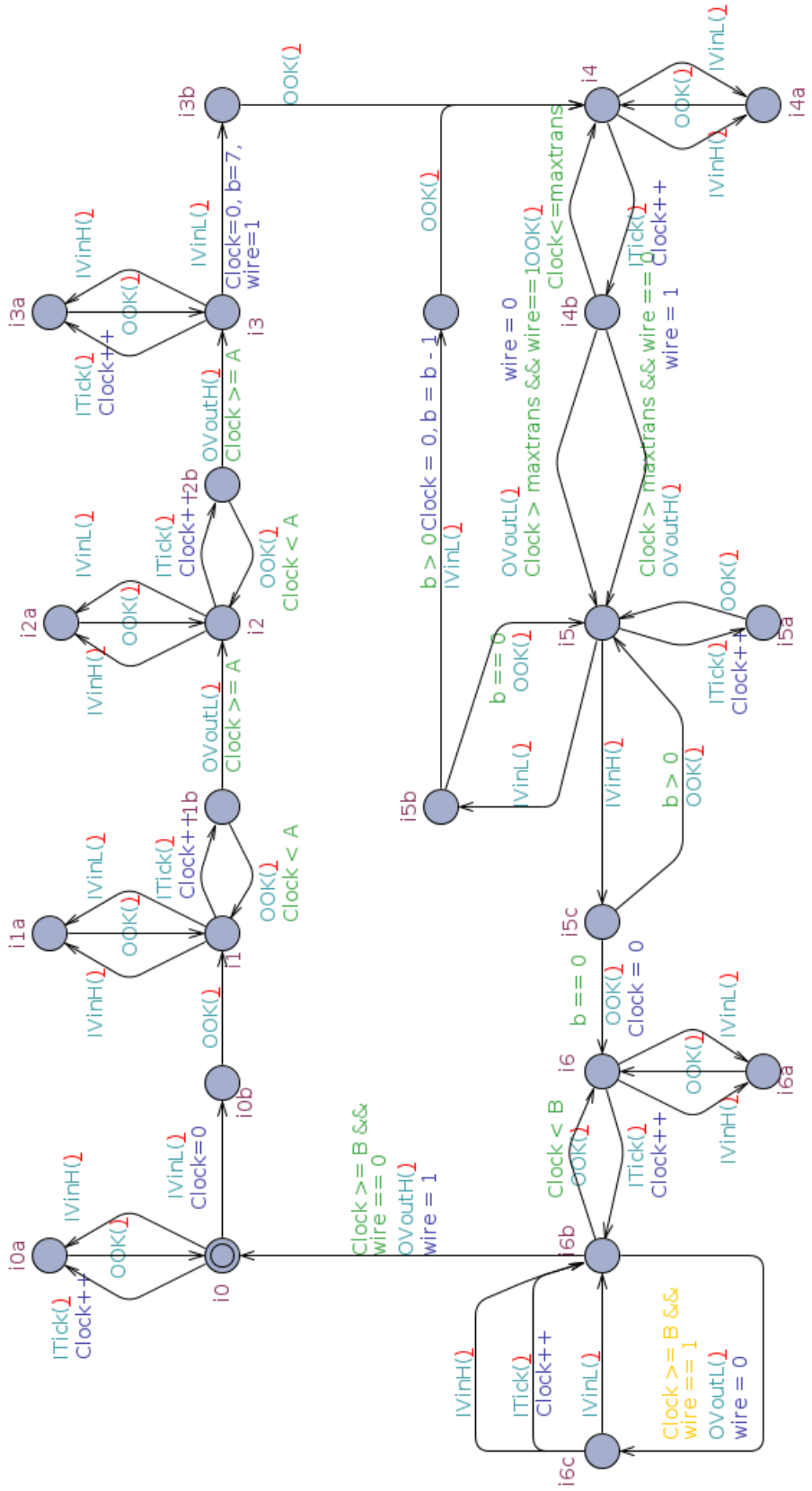
Bij deze resultaten een verklaring vinden is een stuk minder voor de hand liggend. Dat model 1 weinig toestanden heeft en geen tegenvoorbeeld, lijkt te zeggen dat er niet verder wordt gekeken dan toestand i5c. Bij het analyseren van het resultaat (figuur 10) blijkt dat het model in de i5-i5b-i4-i5 loop terecht komt, en ongeacht de parameters van LearnLib daarin blijft. Bij model 2 en 3 kan dit dan ook het geval zijn, dit zou verklaren dat de overgang van i5c

naar i6 in het geleerde model niet aanwezig is, maar in het Mealy model wel. Dit zou voorkomen moeten worden door de guard die variabele b aangeeft. Maar dit gebeurt niet.

Model 1 is dus zeer waarschijnlijk onterecht equivalent verklaard. Doordat er weinig toestanden zijn komt het eerste deel van het Mealy model overeen met het resultaat. Het is ook niet mogelijk dat na i5c eenzelfde subroute voorkomt als in de loop i5-i5b-i4-i5, wat rechtstreeks uit het model te lezen is. Waarom de driver wel geleerd wordt maar de sensor niet, is niet duidelijk zonder meer tests uit te voeren met verschillende modellen.



Figuur 8: Het geleerde model 1 van de sensor



Figuur 9: De sensor met tijdvariabelen A, B

6 Conclusie

We zijn begonnen met twee modellen die het gedrag van een simpele infrarood sensor beschrijven: de sensor en de driver die de sensor aanstuurt. In dit systeem speelt tijd een rol, omdat er bijvoorbeeld tussen twee metingen gewacht moet worden voor de volgende meting kan beginnen. Door dit aspect van tijd neemt het aantal toestanden toe. Als we een bepaalde tijd moeten afwachten in toestand x, hebben we een verzameling toestanden (x, t_1), $(x, t_2)...(x, t_n)$ waar de verzameling t bestaat uit elk mogelijk tijdstip vanaf dat er gewacht wordt, tot de bepaalde tijd t_n wordt bereikt. Afhankelijk van hoe je deze tijd kiest zijn dit een groot aantal toestanden..

Inderdaad afhankelijk van hoe we de tijd kiezen. Want we hebben besloten om de tijd discreet te maken in plaats van continue. Hierdoor is nog maar een beperkt aantal tijdstippen aanwezig tussen t_1 en t_n. De equivalentie tussen continue en discreet ligt buiten de scope van dit artikel, eveneens wat er zou gebeuren als er een handeling optreedt buiten de discrete intervallen.

Als input heeft LearnLib Mealy machines nodig, machines met een input en output alfabet. De sensor en driver modellen zijn dan ook omgezet in hun respectievelijke Mealy machine. Door de modellen te traceren is na te gaan dat de oorspronkelijke en het nieuwe Mealy model aan elkaar equivalent zijn. In deze Mealy machines komt goed na voren dat de modellen met discrete tijd werken: door middel van een *Tick* instructie wordt de klokwaarde in een toestand verhoogd.

Nu de Mealy machines compleet zijn, konden de guards in de modellen gevarïeerd worden om te onderzoeken wat allemaal mogelijk was in LearnLib. Er kwam naar voren dat lagere guards (en dus een kleiner aantal toestanden) meer garant staan voor een correct geleerd model dan guards met hoge waardes. Ook werd duidelijk dat het model van de driver eenvoudiger was dan het model van de sensor. Hieruit is te zeggen dat het mogelijk is dat het ontwerp van een model meespeelt in de leerbaarheid ervan.

De onderzoeksvraag luidde: **Is het mogelijk een extreem simpel real-time systeem te laten leren door Learnlib?**
Aan de hand van onze resultaten kunnen we zeggen: Ja, dit is mogelijk. In het geval van de driver is het mogelijk bij laag gekozen waardes voor de guards. Er zijn echter ook situaties waarin het leren niet mogelijk is. Waarom

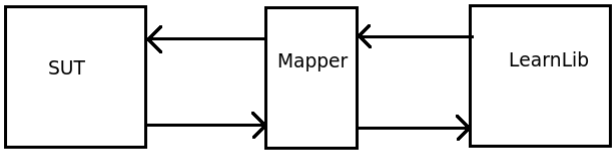
precies dit niet mogelijk is, is meer onderzoek voor nodig, een enkel systeem kan daar weinig over zeggen.

Bij de driver is te zien dat lage waardes zorgen voor een correct geleerd model. Zodra deze waardes echter boven een bepaalde grens komen, weet LearnLib niet meer met de modellen om te gaan. Bij de sensor weet LearnLib in zijn geheel niet met het model om te gaan. Na de loop in het model loopt LearnLib vast waardoor er geen model geleerd wordt. Dit kan liggen aan het ontwerp van het model. Ook kan het zijn dat het aantal toestanden toch te groot is.In de volgende sectie worden enkele suggesties gedaan om deze hindernissen te ontdekken en uiteindelijk te kunnen nemen.

7 Vervolgonderzoek

Uit de resultaten is gebleken dat het gedrag van systemen met tijd, inderdaad te leren is met LearnLib. Maar deze sensor is slechts een miniem systeem, en bij het grootst geleerde model werd al een aantal van 112 toestanden bereikt. Bij grotere systemen zal het aantal toestanden nog meer toenemen, zoals te zien is in de stijgende lijn in aantal toestanden bij deze modellen. Het zou veel mooier zijn als de toestanden die ontstaan door het wachten voor guards, konden worden samengevoegd tot een enkele toestand, met als label het aantal *Ticks* dat is gewacht. In het hier onderzochte model zou dat al gauw meer dan 700 toestanden besparen!

Een voorstel voor vervolgonderzoek is dan ook om een mapper module tussen LearnLib en de SUT te implementeren en testen. Het idee van deze mapper is dat het alle 'gewone' input acties (in ons geval de VinL etc.) maar dat hij wordt ingeschakeld bij de Tick instructie. In plaats van dat LearnLib nu voor elke Tick instructie een nieuwe toestand aanmaakt, houdt de mapper dit tegen, en telt het aantal Tick instructies. Zodra een transitie naar een verdere toestand plaatsvindt, zal de mapper uitgeschakeld worden, en als output het aantal Ticks aan LearnLib geven, die vervolgens een enkele toestand met dit getal als label kan toevoegen. Bij de guards van 700 in ons model zouden dan geen 700 toestanden, maar slechts 1 toestand worden toegevoegd.



Figuur 10: Een schematische weergave van de mapper

Een andere volgende stap is om het discretiseer principe op andere gelijkwaardige modellen toe te passen en de resultaten te analyseren. In deze case-study is gebleken dat de driver wel, maar de sensor niet leerbaar is. Waaraan ligt dat? Er zijn geen vergelijkbare artikelen om op terug te vallen, dus meer tests met meer modellen zijn nodig.

Referenties

[1] http://www.italia.cs.ru.nl

[2] T. Bourke, A. Sowmya - Analyzing an Embedded Sensor with Timed Automata in Uppaal. 2011

[3] http://www.learnlib.de

[4] http://www.inrialpes.fr/vasy/cadp/

[5] Olga Grinchtein, Bengt Jonsson, Martin Leucker; Learning of event-recording automata; Theor. Comput. Sci. 411(47): 4029-4054. 2010

[6] Olga Grinchtein, Bengt Jonsson, Paul Pettersson; Inference of Event-Recording Automata Using Timed Decision Trees; In Proc. CONCUR 2006, 15th Int. Conf. on Concurrency Theory, volume 4137 of Lecture Notes in Computer Science, pages 435-449, 2006

[7] Sicco Verwer and Mathijs de Weerdt and Cees Witteveen; Efficiently learning simple timed automata; In Induction of Process Models (IPM), pp. 61-68. University of Antwerp. 2008

[8] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen; An algorithm for learning real-time automata; In Benelearn, pp. 128-135 (2007).

[9] SHARP CORPORATION; GP2D02: Compact, high sensitive distance measuring sensor. 1997

[10] Bernhard Steffen, Falk Howar, & Maik Merten; Introduction to Active Automata Learning from a Practical Perspective; SFM 2011, LNCS 6659, pp. 256296, 2011