

Verslag Tinlab Advanced Algorithms

J. I. Weverink

...

15 april 2021



Inhoudsopgave

1	Inleiding	3
2	Requirements	4
2.1	Requirements	4
2.1.1	Mode confusion	4
2.1.2	Automatisering paradox	5
2.2	specificaties	5
2.3	Het vier variabelen model	6
2.3.1	Monitored variabelen	6
2.3.2	Controlled variabelen	6
2.3.3	Input variabelen	6
2.3.4	Output variabelen	6
2.4	Rampen	7
2.4.1	Therac-25	7
2.4.2	Vlucht 1951	8
2.4.3	Tsjernobyl 1986	9
2.4.4	Kernramp Fukushima	10
2.4.5	vlucht 3054	10
2.4.6	Hoofddorp	11
3	Modellen	12
3.1	De Kripke structuur	12
3.2	Soorten modellen	12
3.3	Tijd	12
3.4	Guards en invarianten	12
3.5	Deadlock	13
3.6	Zeno gedrag	13
4	Logica	14
4.1	Propositielogica	14
4.2	Predicatenlogica	14
4.3	Kwantoren	14
4.4	Dualiteiten	14
5	Computation tree logic	15
5.1	De computation tree	15
5.2	Operator: AG	15
5.3	Operator: EG	15
5.4	Operator: AF	15
5.5	Operator: EF	15
5.6	Operator: AX	15
5.7	Operator: EX	16
5.8	Operator: $p \cup q$	16
5.9	Operator: $p \cap q$	16

5.10 Fairness	16
5.11 Liveness	17

1 Inleiding

Na grondige analyse van het Nederlandse sluisenpark is gebleken dat renovatie van een groot aantal sluisen noodzakelijk is. Hiervoor zal de sluis eerst gemodelleerd moeten worden. Dit verslag biedt als een basis voor de kennis omtrent modelleren. Dit verslag is gemaakt ter ondersteuning van de opdracht van het Ministerie van Infrastructuur en Waterstaat. Hier wordt verduidelijking gegeven in een aantal onderdelen met betrekking tot het ontwerp proces, modellen en logica.

2 Requirements

2.1 Requirements

Requirements zijn beschrijvingen over hoe een product zou moeten functioneren. Zo verandert de betekenis van een requirement als de machine in een andere omgeving wordt geplaatst. De requirements voor de verwarming van een ruimte bijvoorbeeld: Binnen moet het altijd warm zijn. In Nederland kunnen we zeggen dat 25°C als warm wordt aangezien. Terwijl op de noordpool dat op een lager punt zal zijn.

Anders gezegd zijn requirements geen harde eisen. Dit komt doordat de requirements zijn geformuleerd vanuit het perspectief van de opdrachtgever. De opdrachtgever kan de requirements geven zonder kennis te hebben van de machine die het moet gaan uitvoeren. De requirements die zijn opgesteld geven dan ook geen grenzen aan die overschreden kunnen worden.

Onder requirements zijn er verschillende soorten requirements. Zo zijn system requirements opgesteld voor het hele systeem en bevatten subsystemen die die kunnen bestaan uit software en hardware. Hier moet uiteindelijk alles ervoor zorgen dat deze requirement wordt gehaald. Software requirement zijn niet bedoeld voor het hele systeem, maar behappen alleen de software van het systeem. Software requirement zijn niet bedoeld voor het hele systeem, maar behappen alleen de software van het systeem. De software requirements kunnen gaan over de functionele eisen, gebruikers eisen en zakelijke vereisten. Requirements zijn onder te verdelen in verschillende delen:

- Functional Requirement
- Performance requirement
- Usability requirement
- User requirement
- Interface requirement
- Modes requirement
- Adaptability requirement
- Physical requirement
- Design requirement
- Environmental requirement
- Logistical requirement

Onder deze verschillende requirements zijn er nog twee soorten, functionele en niet-functionele requirements. Functionele requirements geven aan wat het systeem moet doen en kunnen. Niet-functionele requirements geven de eigenschappen aan van het systeem, zoals snelheid, veiligheid en bruikbaarheid. Met andere woorden functionele requirements geven informatie over het "wat". Niet-functionele requirements geven informatie over het "hoe".

2.1.1 Mode confusion

De naam van het begrip zegt het eigenlijk allemaal. Bij mode confusion maakt de gebruiker een vergissing in de huidige of geactiveerde modus van het systeem. De gebruiker denkt dat het systeem in modus A staat terwijl het werkelijk in modus B staat.

2.1.2 Automatisering paradox

Automatiseringsparadox. Wanneer een systeem is dat volledig geautomatiseerd moet worden is er altijd een wel een stap die dat nog niet is. Wanneer er een stap is geautomatiseerd moet er weer iets anders geautomatiseerd worden, om het hele systeem automatisch te krijgen.

2.2 specificaties

Specificaties zijn eigenlijk niet heel veel anders dan requirements. ze beschrijven beide een systeem of een deelsysteem. Het grote verschil tussen de twee is de grenzen die ze opleggen. Bij requirements is er ruimte voor interpretatie, bij specificaties is die ruimte voor interpretatie er niet.

De specificaties geven geen ruimte voor interpretatie, omdat ze 'meetbare' informatie bevatten. In specificaties worden meetbare eenheden gebruikt, zoals 10 meter of 10°C. Door dat de eisen een meetbare eenheid bevatten kan hiervan niet worden afgewezen. Deze specificaties zullen dan ook niet veranderen als het wordt gebruikt in een ander land, doordat de eenheden zijn gegeven.

Stel we nemen het eerder genoemde requirement voorbeeld: "Binnen moet het altijd warm zijn." Als we dit vertalen naar een specificatie wordt het: "Binnen moet het altijd minimaal 20°C zijn."

2.3 Het vier variabelen model

2.3.1 Monitored variabelen

Monitored variabelen zijn waarnemingen die kunnen worden gemeten vanuit de omgeving. De waarnemingen worden gemeten door sensoren, de gemeten data wordt als input gebruikt. Voorbeelden van zulke waarnemingen zijn:

- Temperatuur
- Licht intensiteit
- Luchtvochtigheid

2.3.2 Controlled variabelen

Controlled variabelen zijn waarnemingen die in zekere zin beïnvloed kunnen worden. Zo kunnen de controlled variabelen 'bestuurd' worden, er is controle over. Voorbeelden van controleerbare waarnemingen zijn:

- Temperatuur
- Licht intensiteit

2.3.3 Input variabelen

Input variabelen zijn de waardes, of de data, die een sensor doorstuurt naar de software die het systeem bestuurd. Deze data staat voor de waardes die de sensor heeft gemeten vanuit de omgeving. Omdat de sensor de gemeten waardes heeft omgezet in data kan dit worden gebruikt in de software, om bijvoorbeeld berekeningen mee te maken.

2.3.4 Output variabelen

Output variabelen zijn de 'uitkomsten' van de software die worden uitgelezen door de actuatoren. De actuatoren handelen aan de hand van de output van de software.

2.4 Rampen

2.4.1 Therac-25

Beschrijving

De therac-25 was een radiatie therapie apparaat. Het zou met behulp van x-rays of een straal van elektronen kanker cellen moeten doden. De goede cellen rondom de kanker cellen worden ook beschadigd, dit is heel normaal en gebeurt ook met andere behandelingen als chemo therapie. De therac-25 is 5 jaar in gebruikt geweest en heeft in 1986 en 1987 bijelkaar zes leven gekost.

Datum en plaats

De therac-25 is vanaf 1982 tot 1987. Zes ongelukken vonden plaats in het jaar 1986 en 1987.

Oorzaak

De doodsoorzaak in alle gevallen was een overdosis aan radiatie. Deze zijn veroorzaakt door een software fout, volgens experts. De fout zit in het instellen van de therac-25. Als x-ray wordt ingesteld dan moet de machine in positie komen. Als tijdens het handelen van de machine de electronodus selecteerd, dan zet de machine zichzelf niet in de goede positie.

Ik ben het hier eens dat dit een software matige fout is, maar er was nog een opmerkelijk onderdeel dat de doden had kunnen voorkomen. De stappen zijn gevolgd om dezelfde fout te creëren op de therac-20, hier zijn de test niet fout gegaan. Dit komt doordat de therac-20 een zekering in plaats heeft die als een hardware beveiliging geldt. Als de therac-25 deze ook had gehad zouden er geen zware ongelukken moeten zijn geweest.

Met de therac-25 had een ernstige fout die relatief makkelijk voorkomen had kunnen worden. De Monitored variabelen worden wel geaccepteerd, want de juiste stralingstechniek wordt gebruikt. Het is nu ter bespreking of het fout gaat bij de software, of de output variabelen, of bij de actuatoren, controlled variabelen. De vraag is echter niet moeilijk om te beantwoorden. Er was namelijk niks stuk aan de machine, de actuatoren waren in orde. Dit bevestigt dus dat er een fout zat in de software, wat betekent dat er iets niet klopte met de output variabelen.

Verder wil ik wel melden dat het ontwerp van de therac-25 slordig is uitgevoerd. De therac-20 had wel een hardware noodstop, de zekering, maar in de therac-25 is deze weggehaald. Dit wordt bevestigd in het onderzoek van N. G. Leveson en C. S. Turner *An Investigation of the Therac-25 Accidents* [?], waarin wordt vermeld dat het een grote fout is in engineering om te veel vertrouwen te hebben in de software van een product.

2.4.2 Vlucht 1951

Beschrijving

Een boeing 737/800 van Turkish Airline is neergestort op ongeveer 1.5km voor zijn bestemming, Amsterdam Schiphol Airport. Er waren 135 mensen aan boord, waarvan 128 passagiers en 7 crewleden. Hiervan zijn 9 mensen om het leven gekomen en 86 waren gewond geraakt tijdens de crash.

Datum en tijd

25 february 2009

Oorzaak

Een van de redenen van het neerstorten van de boeing is een kapotte hoogtemeter. Op een hoogte van 600 meter gaf de hoogtemeter een hoogte van -2 meter aan. Dit gaf als gevolg dat het systeem de piloten een signaal gaf dat er veranderingen aangebracht moesten worden, waaronder naar een lagere versnelling schakelen.

Het ongewone handelen van het vliegtuig is de piloten niet opgevallen. Ook de ongebruikelijke aankondigingen voor die hoogte zijn de piloten onopgemerkt gegaan. Als de piloten correct hadden gehandeld en de "foutieve" instructies van het systeem hadden opgemerkt. had dit ongeluk voorkomen kunnen worden.

Vanuit de ogen van het 4 variabelen model heeft dit ongeluk veel problemen in mijn ogen. De hoogtemeter gaf een incorrecte waarde aan, wat aangeeft dat een hoogtesensor defect was, waardoor er incorrecte input variabelen zijn doorgegeven. Hierdoor zijn er verkeerde aannames gemaakt in de software. Het opmerkelijke verschil in hoogte had in de software ook afgevangen kunnen worden, dit vind ik dan ook weer een software matige fout.

Verder is er ook door de piloten incorrect gehandeld. De piloten hadden de foute signalen van het systeem moeten opmerken. Wat kan worden vertaald naar de monitored variabelen, de piloten, die niet goed hebben gehandeld in deze situatie. De piloten worden uiteindelijk ook aansprakelijk gesteld in *Analysis of the crash of TK 1951 using CATS*[6] waarin de crash werd onderzocht.

2.4.3 Tsjernobyl 1986

Beschrijving

Tsjernobyl is de grootste kernramp aller tijden. Op de dag van het ongeluk werd er een test uitgevoerd. Doordat er een andere elektriciteits centrale was uitgevallen moest er worden getest of de reactor genoeg vermogen gaf voor de koelinstallatie, terwijl de noodaggregaten bezig waren met opstarten.

Datum en tijd

26 april 1986.

Oorzaak

Om meer vermogen te wekken zijn 20 van de 26 veiligheidsstaven omhoog gehaald. Dit heeft gewerkt, maar omdat er nog maar 6 staven van de bedoelde 26 in positie waren bleef het vermogen alleen maar stijgen. Op het moment dat de staven weer omlaag gehaald moesten worden ging het mis. De regelstaven hadden een ontwerp fout. De punt was zo gevormd dat het eerst het water aan de kant schoven voor ze een remmende effect konden uitoefenen. Hierdoor heeft het vermogen onder in de centrale op kunnen lopen, wat resulteerde in een explosie, die op zijn beurt ervoor zorgde dat de regelstaven klem kwamen te zitten. De regelstaven zaten op een derde van de oorspronkelijke diepte, hierdoor heeft het vermogen kunnen oplopen tot 10x de normale vermogen. De regelstaven konden dit niet meer aan en smolten waaruit nog een explosie ontstond.

Aan deze reeks van gebeurtenissen is een duidelijke fout te halen, de regelstaven. De regelstaven waren zo gevormd dat de het water werd wegzet wanneer ze terug op hun plaats werden gezet. Hierdoor zijn de reactie niet snel genoeg geremd. Hier is waarschijnlijk geen rekening mee gehouden in de requirements voor de regelstaven.

2.4.4 Kernramp Fukushima

Beschrijving

In Fukushima was er een kerncentrale aan de kust. Alle reactoren waren stilgezet omdat de centrale werd getroffen door een aardbeving. Doordat de aardbeving plaatsvond op zee volgde er na de aardbeving ook nog een tsunami die de centrale ook schade aanbracht.

Datum en tijd

11 maart 2011

Oorzaak

Door de aardbeving zijn de actieve reactoren stopgezet. Omdat er nu geen elektriciteit meer werd gewekt moest er nu overgeschakeld worden op noodstroomvoorzieningen om het koelsysteem draaiende te houden. Doordat de beschermingen tegen vloedgolven niet voldoende waren zijn gebouwen beschadigd, hierdoor kwamen reactoren 1 tot en met 4 zonder stroom. Hierdoor zijn bij elke reactor verschillende problemen ontstaan.

Dit is een duidelijk voorbeeld van een gebrek in de specificaties. De kerncentrale was voorzien van bescherming tegen aardbevingen maar niet goed genoeg tegen bescherming tegen vloedgolven. De vloedgolf was uitzonderlijk hoog, maar de beschermingen die aanwezig waren waren maar bestemd voor een derde van de werkelijke hoogte wat uiteraard veel te laag is.

2.4.5 vlucht 3054

Beschrijving

Linhas Aéreas-vlucht 3054 had mankementen tijdens de landing. Nadat het toestel succesvol de grond bereikte is het niet in staat gekomen om het toestel tot stilstand te krijgen, met grote gevolgen van dien. Nadat het vliegtuig doorschoot over de landingsbaan is het in brand gevlogen. In totaal zijn er 195 mensen omgekomen bij het ongeluk.

Datum en tijd

17 juli 2007.

Oorzaak

Nadat de landing is het systeem gefaald in het activeren van de spoilers, bovendien zijn de automatische remmen ook niet geactiveerd. Dit zijn beide cruciale onderdelen voor het tot stand komen van het toestel. De onderdelen ter sprake waren niet defect, hierdoor kan gezegd worden dat de output variabelen hier fout zetten. De onderdelen zijn niet aangezet terwijl dit wel had moeten.

2.4.6 Hoofddorp

Beschrijving

Doordat een intercity met een te hoge snelheid in de bocht is gegaan is de intercity ontspoord. Dit ongeval heeft in totaal 5 doden gekost en meer dan dertig gewonden.

Datum en tijd

30 november 1992.

Oorzaak

Door de werkzaamheden moesten treinen een S-bocht maken om de werkzaamheden te kunnen passeren. In deze S-bocht was de maximale snelheid 60km/h. Om een trein te laten ontsporen moet deze snelheid overschreden worden. Uit onderzoek is gebleken dat de trein tussen de 123 en 130km/h heeft gereden. De machinisten beweerde geen signaal te hebben gekregen voor een maximum snelheid, terwijl een snelheidbord met erop 60 langs de rails stond. Door de hoogte van de cabine waar de machinist in zat heeft hij de tekst op het bord niet kunnen zien. en ging er vanuit dat er 90 op stond.

De is misschien wel een vergezocht kritiek punt, maar blijkt uit dit voorbeeld iets wat toch cruciaal kan zijn. Een requirement van een verkeersbord, waarvan we kunnen uitgaan dat die er is, is dat een verkeersbord duidelijk zichtbaar moet zijn. hier is duidelijk dus niet aan voldaan vanuit de ogen van de machinist. Hier ben ik het mee eens, het verkeersbord moet goed zichtbaar zijn, ookal is de cabine hoger dan een gemiddelde cabine.

3 Modellen

Een model is een schematische weergave van de werking van een systeem. Elk soort systeem kan gemodelleerd worden, of het ingewikkeld of heel simpel is. Een model wordt altijd zo realistisch mogelijk gemaakt. Dat wil zeggen dat de belangrijke onderdelen in het model ook terug komen het "echte" systeem.

3.1 De Kripke structuur

Een kripke structuur bestaat uit binaire overgangsrelaties tussen state, die de werking van een systeem weergeven.[3] Aan deze overgangen en handelingen kunnen eigenschappen gekoppeld worden. Op deze manier kunnen de handelingen en eigenschappen gecontroleerd en verifieerd worden en het systeem getest worden of het op juiste wijze handelt. De acties binnen een kripke structuur lopen altijd synchroon.

3.2 Soorten modellen

Andere soorten modellen zijn discrete and continuous -time Markov chains, deze zogehete "Marko chains" zijn een vorm van stochastic model checking. Bovendien kosten continuous chains enorm veel kracht om te berekenen. Zoals A. Philippe en C. Robert het verwoorden in hun boek *Linking Discrete and Continuous Chains*[5] hebben deze vormen een gebrek aan intuïtieve basis achter de theorie van Markov chains.

3.3 Tijd

Tijd kan een belangrijke rol spelen in het modelleren van een systeem. De tijd die jij en ik bijhouden op een 24-uurs klok die is net zo van invloed op een systeem. De handelingen gebeuren immers in de loop van de tijd. In UPPAAL kan met het verloop van tijd ook gewerkt worden. De tijd verloop is altijd synchroon. Dit houdt in dat er geen 2 of meerdere acties tegelijk uitgevoerd kunnen worden.

3.4 Guards en invarianten

Guards zijn voorwaarden waaraan moet worden voldaan voordat een transitie genomen kan worden. Het is als het ware een "if statement", is de uitkomst niet true dan kan de transitie niet genomen worden.

Invariant theory is al heel oud en komt oorspronkelijk uit de wiskunde. Over de jaren heen heeft het verschillende betekenissen gekregen. Hermann Weyl legt dit uitgebreid uit in zijn boek *The Classical Groups*. [2] G. Rota haalt uit het boek van Weyl 2 beweringen. De eerste "all geometric facts are expressed by the vanishing of invariants". en de tweede "all invariants are invariants of tensors". Echter denk ik dat deze beweringen niet van toepassing zijn in deze context.

In de context van de opdracht zijn invarianten voorwaarden die kunnen worden vastgesteld aan een state. De state moet worden verlaten zodra de invariant niet

meer van kracht is (als de voorwaarde niet meer behaald is) en zal geforceerd worden een beschikbare transitie te nemen.

3.5 Deadlock

Een deadlock ontstaat als er een proces in groep processen een geheugen locatie bezet houden en bij een andere geheugen locatie wilt komen, maar deze wordt al bezet gehouden door een ander proces in de groep. Dit is een deadlock in de ogen van een programmeur.[9]

Een deadlock heeft echter een hele andere definitie als het neerkomt op modelleren. In de wereld van modelleren heeft een deadlock niets te maken met gebruik van geheugen locaties. Een deadlock kan alleen ontstaan als er restricties zijn gelegd aan transities. Deze restricties zorgen ervoor dat niet elke transitie altijd genomen kan worden. Als er een state is waar geen volgende transitie meer mogelijk is preken we van een deadlock.

3.6 Zeno gedrag

Zeno gedrag is een wiskundige term. het beschrijft een situatie waarin in een bepaalde tijd oneindig veel transities ontstaan.[10] Doordat er oneindig veel stappen worden gezet in een korte (of lange) tijd, moeten de stappen extreem klein zijn. Dat moet wel want er is immers geen einde aan te krijgen.

4 Logica

Logica is een de wetenschap die focussed op de regels van redeneren.

4.1 Propositielogica

Een propositie is een taaluiting ofwel een bewering of uitspraak uitgedrukt in een zin. De zin beschrijft een situatie of stand van zaken. Deze zin heeft een waarheidswaarde van 0 (nietwaar) of 1 (waar), het kan absoluut niet beide zijn. Bijv. "De auto staat stil." of

In de propositielogica worden deze proposities gebruik is samenwerking met logische operatoren zoals bijv. AND en OR.[7]

4.2 Predicatenlogica

Een predicaat is een eigenschap of kwaliteit dat over iets gezegd gezegd kan worden. Door gebruik te maken van predicatenlogica kan een zin met een predicaat vertaald worden op een dusdanige manier dat een (computer) systeem dit kan begrijpen. Het predicaat wordt vertaald naar een wiskundige formule. In deze formule zijn alle eigenschappen en relaties nogsteeds zichtbaar.

4.3 Kwantoren

Met kwantoren worden eigenschappen gekoppeld aan een onderwerp waarover ze informatie verschaffen. Stel de volgende kwantoren over rotten appels in een winkel. Een voorbeeld hiervan is:

$\exists xP(x)$: Er zijn appels die rot zijn.

$\forall xP(x)$: Alle appels zijn verrot.

Als we in deze kwantoren stellen dat P staat voor verrot en de x voor appels dan zijn de beweringen die erachter staan van kracht.

4.4 Dualiteiten

Dualiteit betekend dat van een stelling er een andere stelling bestaat met dezelfde betekenis.

$$\neg \forall xP(x): \exists x\neg P(x)$$

Als we weer het voorbeeld van de appels nemen staat hier effectief links niet alle appels zijn verrot. Aan de rechter kant staat Er zijn appels die niet verrot zijn.

5 Computation tree logic

5.1 De computation tree

Computation tree logic is een model waar de tijd een vertakkings structuur aan states heeft, waarin de toekomst nog onbepaald is. Er zijn verschillende wegen die afgelegd kunnen worden, waarvan er één mogelijk werkelijk genomen zal worden.

Een computation tree wordt opgebouwd vanuit de begin state. vervolgens komt er nog een laag aan de hand van alle transities die deze begin state kan nemen. De tweede laag bestaat dan uit alle states waar de transities van de begin state eindigen. Dit kan doorgaan tot er state tegen komt waar geen volgende transitie uit te volgen is.

5.2 Operator: AG

De betekenis van AG is makkelijk te onthouden A = Always, G = Globally. Dit houdt in dat het niet uit maakt waar je bent, je zal altijd van welke positie dan ook bij een gedefinieerde state uitkomen.

5.3 Operator: EG

De EG operator staat voor "Exists Globally". Dit betekent dat er van alle wegen een aantal zijn, niet allemaal, die leiden naar het gedefinieerde state.

5.4 Operator: AF

De AF operator kan worden samengevat als "Always Eventually". Dit kan worden vertaald naar het niet uit welke weg wordt genomen, er is altijd een moment waarop het gedefinieerde state is bereikt.

5.5 Operator: EF

De EF lijkt uiteraard op de AF operator, alleen geldt bij de EF operator dat **niet** bij alle wegen de gedefiniëerde state wordt bereikt. Met andere woorden op enkele wegen zal er uiteindelijk de gedefiniëerde state worden bereikt. Deze operator kan dan ook worden samen gevat als "Exists Eventually".

5.6 Operator: AX

De AX operator is anders dan de vorigen operatoren. Het heeft namelijk alleen maar betrekking tot de volgende state. De AX operator kan worden samengevat als "Always Next", wat dus inhoudt dat dus elk volgende state de gedefiniëerde state zal bevatten.

5.7 Operator: EX

Het spreekt voor zich dat de EX operator ook alleen voor de volgende state zal gelden. De EX operator staat dan voor "Exists Next". Voor de EX operator geldt dat er in enkele volgende states de gewenste state bereikt zal zijn.

5.8 Operator: $p \text{ U } q$

Bij deze operator draait het vooral om de U. De U kan worden verstaan als "Until", zo staat er hier p is waar tot dat q waar is. Anders gezegd is p waar en ophet moment dat dat stopt, p onwaar wordt, dan wordt q waar. Het is mogelijk voor p en q om uiteindelijk beide onwaar te worden, maar dit gebeurt pas nadat de "cyclus" van p naar q minimaal één keer is doorlopen.

Een raar maar simpel voorbeeld hiervan zou zijn een doos die wordt getild door persoon p, p is waar. Na enige tijd laat persoon p de doos vallen, die de doos wordt gevangen door persoon q. Op dit moment word q waar en p onwaar. Het komt dus niet voor dat de p en q samen waar zijn.

Deze operator kan genest worden in een Always of Exist operator. Door dit te doen komt er het volgende uit:

$A(p \text{ U } q)$: Voor deze samenstelling geldt dat op alle wegen de $p \text{ U } q$ operator van kracht is.

$E(p \text{ U } q)$: Hier geldt dat op enkele wegen de $p \text{ U } q$ operator van kracht is.

5.9 Operator: $p \text{ R } q$

De R operator is zeer gelijk aan de U operator. Ze bevatten beiden de eigenschap dat nadat p niet meer waar is dat q waar is. Er is een eigenschap die echter het verschil maakt. De R operator heeft een moment, een state, waarin zowel p als q waar zijn.

Om dit in hetzelfde voorbeeld te verduidelijken. Draagt persoon p nu weer de door, p is waar. Zodra persoon p moet wordt geeft hij de doos over aan persoon q. Let op dat tijdens het geven van de doos er een moment is waarin zowel persoon p als q de doos vasthebben. Dit is het verschil met de U operator, er is een state waarin beide p en q waar zijn.

Net als de U operator is de R operator ok te nestelen in de Always en de Exists operatoren. Hieruit volgt:

$A(p \text{ R } q)$: Voor deze samenstelling geldt dat op alle wegen de $p \text{ R } q$ operator van kracht is.

$E(p \text{ R } q)$: Hier geldt dat op enkele wegen de $p \text{ R } q$ operator van kracht is.

5.10 Fairness

De letterlijke vertaling van fairness is eerlijkheid. Dat is ook een beetje waar de fairness die hier wordt bedoeld op slaat. Fairness betekent namelijk dat alle taken in een systeem "aandacht" zullen krijgen. De zogeheten aandacht wordt eerlijk verdeeld.

Als we fairness uitdrukken in operatoren dan komen we uit op $AG(AF(p))$. Als wij stellen de p staat voor aandacht aan een process. Dan staat hier nu dat het niet uit maakt welke transitie route we afleggen uiteindelijk zal het process aandacht krijgen, zelfs als het al een keer aandacht heeft gehad. het is dus in feite een loop.

5.11 Liveness

Wat gezegd zou kunnen worden over liveness is dat het een vorm van fairness is. Dit komt door dat de operator schrijfwijze vrijwel gelijk is aan die van fairness. liveness wordt geschreven als $AG(r \rightarrow AF(p))$, wat lijkt op fairness $AG(AF(p))$. Wat "extra" is bij liveness is de $r \rightarrow$ genesteld in de AG. De pijl kan worden gezien als een if-statement, if $r = \text{true}$ dan AF, met de r als de bepalende factor. Als de r waar is dan is er een geval van welke weg, of transitie, we ook nemen daar komen we vroeg of laat in een state waar p geldt.

Zoals in het artikel van B. Alpern en B. Schneider[11] wordt gezegd is kan liveness voorkomen in 3 vormen starvation freedom, termination en guaranteed service. Deze vormen zijn ook weer terug te vinden in het boek *Synchronization Algorithms and concurrent Programming*[8] van G. Taubenfeld

Referenties

- [1] WEYL, H (1966). "*The Classical Groups: Their Invariants and Representations*". PRINCETON, NEW JERSEY: Princeton University Press.
- [2] V. Gupta, V Pratt (1993) "*Concurrent Kripke Structures*". Stanford University.
- [3] Kwiatkowska M., Norman G., Parker D. (2007) "*Stochastic Model Checking*". Bernardo M., Hillston J. Springer, Berlin, Heidelberg.
- [4] Philippe A., Robert C.P. (1998) "*Linking Discrete and Continuous Chains*". Robert, Christian P., Springer, New York, NY.
- [5] B.J.M. Ale, L.J. Bellamy, J. Cooper, D. Ababei, D. Kurowicka, O. Morales, J. Spouge, (2010). *Analysis of the crash of TK 1951 using CATS*
- [6] H. K. Büning, T. Lettman (1999). "*Proposition logic: deduction an algorithms*". Cambridge university press, Cambridge.
- [7] G. Taubefeld (2006). "*Synchronization Algorithms and Concurrent Programming*". Pearson Education Limited, Edinburgh.
- [8] S. Isloor, T. Anthony Marsland (1980) "*The Deadlock Problem: An Overview*".
- [9] A. D. Ames, A. Abate and S. Sastry (2005) "*Sufficient Conditions for the Existence of Zeno Behavior*". University of California, Berkeley.
- [10] B. Alpern, F. B. Schneider (1985) "*Defining Liveness*".
- [11] Rota G. -C. (2001). "*Algebraic Combinatorics and Computer Science: A Tribute to Gian-Carlo Rota*". Crapo, H. and Senato, D., Springer Milan.
- [12] N. G. Leveson, C. S. Turner (1993). "*An investigation of the Therac-25 accidents*"