Verslag Tinlab Advanced Algorithms

J. I. Weverink
...

15 april 2021

Inhoudsopgave

1 Inleiding

Zie hier een referentie naar Royce [?] en nog een naar Clarke [?]...

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 geplaats. De requirements voor de verwaring 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 van het perspectief van de opdrachtgever. De opdrachtgever kan de requirtements 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 overscheden 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 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 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
- Enviromental 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 denk 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 interpetatie, bij specificaties is die ruimte voor interpetatie er niet.

De specificaties geven geen ruimte voor interpetatie, 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 variablen 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 variablen 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 bijvoordbeeld 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

Datum en plaats

Oorzaak

3 Modellen

Een model is een schematische weergaven 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 wieze 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 handellingen 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 ïf 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 haal uit het boek van Weyl 2 beweringen. De eerste äll geometric facts are expressed by the vanishing of invariants". en de tweede äll 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.[8]

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 ristricties 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.[9] 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.[6]

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 zei 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 Evetually". 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 operatoo, alleen geld 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 bereik. 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 inhoud at dus elk volgende state de gedefinieerde 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 geld dat er in enkele volgende states de gewenste state bereikt zal zijn.

5.8 Operator: p 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 moeglijk voor p en q om uiteindelijk beide onwaar te worden, maar dit gebreurt 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 U q): Voor deze samenstelling geld dat op alle wegen de p U q operator van kracht is.
E(p U q): Hier geld dat op enkele wegen de p U q operator van kracht is.

5.9 Operator: p 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 moe 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 R q): Voor deze samenstelling geld dat op alle wegen de p R q operator van kracht is.
E(p R q): Hier geld dat op enkele wegen de p 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 edoeld 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 aflegen 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 lijk 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 = 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 geld.

Zoals in het artikel van B. Alpern en B. Schneider[10] wordt gezegd is kan liveness voorkomen in 3 vormen starvation freedom, termination en guarenteed service. Dez vormen zijn ook weer terug te vinden in het boek *Synchronization Algorithms ans concurrent Programming*[7] van G. Taubenfeld

Referenties

[1] Knuth: Use of Tabular Expressions for Refinement Automation, https://www.researchgate.net/figure/4-Variable-Model-of-Parnas-Madey_fig3_270733268

[2] WEYL, H (1966). *The Classical Groups: Their Invariants and Representations.*. PRINCETON, NEW JERSEY: Princeton University Press.

[3] V. Gupta, V Pratt (1993) *Concurrent Kripke Structures*. Stanford University.

[4] Kwiatkowska M., Norman G., Parker D. (2007) *Stochastic Model Checking*. Bernardo M., Hillston J. Springer, Berlin, Heidelberg.

[5] Philippe A., Robert C.P. (1998) *Linking Discrete and Continuous Chains*. Robert, Christian P., Springer, New York, NY.

[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 Californioa, Berkeley.

[10] B. Alpern, F. B. Schneider (1985) *Defining Liveness*.

[11] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[12] Rota G. -C. (2001). *Algebraic Combinatorics and Computer Science: A Tribute to Gian-Carlo Rota*. Crapo, H. and Senato, D., Springer Milan.

[13] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.

[14] Knuth: Computers and Typesetting, http://www-cs-faculty.stanford.edu/~uno/abcde.html