Verslag Tinlab Advanced Algorithms

A. Almeida Mendes
176-671

15 april 2021

Inhoudsopgave

1 Inleiding

In dit verslag behandel ik 6 rampen waarbij ik analyseer waar in het vier variabelen model er iets mis is gegaan dat tot deze ramp heeft geleid. 3 van deze rampen zijn vooraf bepaald (de eerste 3) en de 3 andere rampen heb ik zelf aangeleverd om dat deze mij fascineren. Ik heb hier voor rampen/ongelukken gekozen die ontstaan zijn door ogenschijnlijk kleine software- en modelfouten. Verder behandel ik de belangrijkste termen die ook in de les voorbij zijn gekomen en geef ik bij ieder van deze termen uitleg.

1.1 Requirements

De requirements zijn de eisen waaraan een product moet voldoen. Het is een lijst die puntsgewijs aangeeft wat een product moet kunnen of wat een product juist niet moet doen. Requirements zijn doelen die de opdrachtgever stelt aan het begin van een project. Deze requirements/eisen horen niet in te technische taal genoteerd te worden omdat de opdrachtgever dit wellicht niet begrijpt. Requirements worden dus altijd beschreven in taal die de opdrachtgever/klant snapt. Met enige regelmaat zijn requirements informeel en worden dan niet precies geformuleerd.

1.2 specificaties

Specificaties worden ook weleens software requirements genoemd. Specificaties zijn preciezer geformuleerd dan requirements en zijn doorgaans meetbaar. Een specificatie van een machine is een uitspraak over een doel dat de machine moet bereiken middels de onderdelen waaruit deze machine bestaat. Een case fan van een computer heeft bijvoorbeeld de specificatie "1500RPM". Doormiddel van het motortje van de ventilator (een onderdeel van de machine) bereikt hij het doel (1500RPMM). Dit is vervolgens ook weer exact te controleren door het aantal omwentelingen van de fan per minuut te tellen/meten.

1.3 Het vier variabelen model

Elk systeem dat wij mensen bouwen valt in het vier variabelen model uit te drukken. Het vier variabelen model is bedacht door Parnas en Madey en wordt gebruikt bij de ontwikkeling van systemen in verschillende industriën. Het model maakt niet expliciet gebruik van software requirements maar verbind deze doormiddel van het weergeven van de systeem eisen en de in- en output interfaces van het systeem [7]

1.3.1 Monitored variabelen

Door sensoren waargenomen fenomenen uit de omgeving.Voorbeelden van waar te nemen femomenen door sensoren zijn: licht, druk, geluid, snelheid, versnelling etc. Met deze sensoren neemt het systeem de omgeving en gebeurtenissen in de omgeving waar. De sensoren kwantificeren de fenomenen. Dit wil zeggen dat

de fenomenen omgezet worden in getallen of een bepaalde schaalverdeling. De monitored variabelen behoren tot de system requirements.

1.3.2 Controlled variabelen

Door actuatoren bestuurde fenomenen uit de omgeving. Deze worden in het Engels ook wel "the movers"genoemd. Denk hier bijvoorbeeld aan een robotarm of een motor die gaat draaien. Deze actuatoren kunnen mechanisch zijn, maar ook elektrisch, pneumatisch etc. Kortweg zorgen de actuatoren van het stysteem dus voor een verandering in de buitenwereld. De controlled variabelen behoren tot de system requirements.

1.3.3 Input variabelen

Data die de software als input gebruikt. Deze data is afkomstig van de sensoren van het systeem. De software leest bepaalde geheugenlocaties waar de sensoren hun gekwantificeerde fenomenen hebben opgeslagen. Zo kan de software aan de slag met de waargenomen data uit de omgeving en deze dus gebruiken als input. De input variabelen behoren tot de software requirements.

1.3.4 Output variabelen

Dit is de data die de software levert als output. De software bewerkt de input data die van de sensoren is verekregen op de manier waarop de software geprogrammeerd is. De actuatoren kunnen met deze output vervolgens aan de slag om fenomenen uit de omgeving te 'besturen'. De output variabelen behoren tot de software requirements.

1.4 Rampen

In technische systemen kunnen fouten in hardware of software fatale gevolgen hebben. In dit hoofdstuk onderzoek ik 6 rampen waarbij er ergens in het systeem een fout heeft plaatsgevonden die grote gevolgen gehad heeft voor de betrokkenen.

1.4.1 Therac-25

Beschrijving De Therac-25 was een bestralingsapparaat voor kankerpatiënten die tumoren kon vernietigen en minimale schade aanrichtte aan gezond omringend weefsel. De machine is ontworpen door het Canadese bedrijf AECL die eerder al de Therac-6 en Therac-20 ontwikkelde met het Franse bedrijf CGR. Het verschil van de Therac-25 met zijn voorgangers zit hem vooral in de manier waarop safety-critical onderdelen van het systeem werden gemonitored. Zo had de Therac-20 onafhankelijke berschermingscircuits en mechanische vergrendelingen die ervoor zorgden dat de elektronenstraal op de juiste en veilige manier kon werken. Om geld te besparen of omdat de ontwerpers meer vertrouwen hadden in de computer, werden de mechanische vergrendelingen niet

overgenomen bij de bouw van de Therac-25. In plaats van deze vergrendelingen controleerde de hoofdcomputer de werking van de hardware. Er werd dus ook geen gebruik meer gemaakt van onafhankelijke circuits die het hoofdcircuit konden controleren. Tussen 1985 en 1987 hebben er meerdere incidenten plaatsgevonden met deze machine. [3] Het eerste ongeluk vond plaats in 1985 in Marietta. Een 61-jarige vrouw werd daar nabehandeld voor borstkanker. Toen de machine werd aangezet voelde de patiënt een extreme hitte. Wanneer de dokter de kamer binnen kwam zei ze: "je hebt me verbrand". De dokter zei dat dit niet mogelijk kan zijn maar voelde toch een warme plek op de patiënt haar schouder. Voor de zekerheid belde hij met de makers van de Therac-25 om te vragen of het mogelijk was dat de machine in de elektronen modus stond zonder eerst de scanner op de juiste positie te hebben gezet (de scanner wordt gebruikt om de straal uit te spreiden over een groter oppervlak). Hij kreeg een aantal dagen later antwoord dat dit niet mogelijk kan zijn. Een aantal weken later moest de borst van de patiënt geamputeerd worden en kon de vrouw haar schouder en arm niet meer bewegen. Daarnaast had ze ook heftige pijn. Uiteindelijk werd geschat dat de vrouw door een systeemfout met 15.000 tot 20.000 rads bestraald is terwijl ze maar 200 rads hoorde te krijgen. Na dit ongeluk hebben er nog 5 andere ongelukken plaatsgevonden. Bij 1 van de incidenten kreeg een patiënt in minder dan 1 seconde een dosis van 16.500 tot 25.000 rads over een oppervlakte van 1 centimeter. De weken na dit ongeluk kreeg de man tal van ernstige klachten zoals: verlamming van de linker arm en benen, misselijkheid, blaasproblemen, niet meer kunnen praten etc. 5 maanden na het ongeluk is de man overleden aan de complicaties van de extreme overdosis aan bestraling.

Datum en plaats Tussen juni 1985 en januari 1987 in Amerika en Canada.

Oorzaak De oorzaak van de 6 incidenten komt voort uit 2 softwarefouten [2] die ervoor zorgden dat de patiënt met een bundel van hoog vermogen röntgenstraling bestraald werden. Een van de softwarefouten zorgden ervoor dat wannneer de bestuurder van de Therac-25 perongeluk de röntgenmodus inschakelde en vervolgens binnen 8 seconde overschakelde naar de elektronenmodus, de elektronenstraal toch ingesteld werd voor röntgenstraling alleen stond het speciale röntgendoel niet op de juiste plek. De patiënt werd dus direct bestraald door de elektronenstraal die veel te veel vermogen had. De tweede softwarefout zorgde ervoor dat de elektronenstraal werd geactiveerd als eigenlijk het verkenningslicht moest worden geactiveerd. De positie van de draaiende tafel bepaalt welk type bestraling mogelijk is. Bij de oudere versies van de Therac werd gebruik gemaakt van mechanische vergrendelingen zodat zelfs bij een softwarefout de patiënt niet foutief bestraald kon worden. Bij de Therac-25 werden deze hardwarematige vergrendelingen niet meer gebruikt. In plaats daarvan werden 3 schakelaren in de draaiende tafel verwerkt om de positie van de instrumenten vast te stellen. Wanneer de tafel niet juist stond afgesteld kon de bestraling niet doorgaan. Wanneer er iets mis was in het circuit van 1 van de schakelaren kon deze een foute waarde doorgeven aan de computer. Dit is dus een fout in de controlled variabele en vervolgens de input

variabele. Vervolgens zou de computer ook met een 1-bit error vast moeten kunnen stellen dat de tafel niet in de juiste positie staat, er werd namelijk gebruik gemaakt van 3 schakelaren. Toch kon de computer hier soms niet goed mee omgaan en werd vervolgens een verkeerde output naar de actuatoren gestuurd, waardoor bestraling dus toch mogelijk werd terwijl dit niet zou moeten kunnen.

1.4.2 Vlucht 1951

Beschrijving Turkish Airlines vlucht 1951 [8], ook wel aangeduid met TK1951, was een Boeing 737-800 die voor de landingsbaan van Schiphol verongelukte. Het vliegtuig vertrok op 25 februari 2009 vanaf luchthaven Istanbul Atatürk met aan boord 128 passagiers en 7 bemanningsleden waaronder 3 piloten. De vlucht verliep zonder problemen totdat het toestel zich boven Lelystad bevond. Hier gaven de linker en rechter radiohoogtemeter 2 verschillende waardes weer aan de piloten. Dit was nog geen groot probleem want bij zo'n storing hoort de automatische piloot over te schakelen naar de juist werkende meter. Dat gebeurde echter niet waardoor de systemen de waarde van de kapotte hoogtemeter gebruikte. De automatische piloot kwam van een zogenaamde 'altitude hold' state in een 'retard flare' state terecht. Hierdoor werd het vliegtuig voor een landing voorbereidt terwijl er nog een stuk te vliegen was. Toen de piloten eenmaal doorkregen dat zij te laag zaten en een stall warning kregen, duwde de eerste officier de stuurkolom naar voren alsmede de gashendel. De gezagvoeder reageerde ook direct op de waarschuwing door de besturing over te nemen. Dit heeft ervoor gezorgd dat de selectie van de stuwkracht door de eerste officier werd onderbroken en het vliegtuig uiteindelijk overtrok (stall). Het vliegtuig storte neer op een akker kort voor de landingsbaan en brak vervolgens in 3 stukken. 9 mensen kwamen bij dit ongeluk om het leven.

Datum en plaats 25 februari 2009, Amsterdam

Oorzaak De linker radiometer was kapot en gaf een hoogte aan van -8 voet terwijl de rechter meter de juiste waarde aangaf. Dit type vliegtuigen hebben 2 hoogtemeters, waarvan primair de linker wordt gebruikt [8]. Wanneer er echter een storing is met de linker hoogtemeter hoort het systeem automatisch over te schakelen naar de werkende rechter hoogtemeter. De piloten gingen er van uit dat de automatische piloot gebruik maakte van de rechter meter omdat een waarde van -8 duidelijk incorrect is. Het systeem herkende dit echter niet als een foute waarde en gebruikte daarom deze data voor diverse andere vliegtuigsystemen. Een van deze systemenen was de autothrottle waardoor het vliegtuig veel langzamer begon te vliegen. De rechter automatische piloot gebruikte wel de juiste hoogtewaarde en zag dat het vliegtuig snel daalde en probeerde dit te compenseren door de neus van het vliegtuig omhoog te trekken. Hierdoor had het vliegtuig uiteindelijk te weinig lift en kwam het in een stall situatie terecht. Het probleem begint hier bij de monitored variabele,

namelijk de kapotte hoogtemeter. Hierdoor krijgt het systeem verkeerde input data, hoewel de andere hoogtemeter wel de juiste input data geeft. Het systeem maakt uiteindelijk de verkeerde keuze om de linker (kapotte) hoogtemeter te gebruiken. De output naar de actuatoren (onder andere de motoren) zorgde ervoor dat de stuwkracht te laag was en dat de neus van het vliegtuig te ver omhoog gericht was.

1.4.3 Tsjernobyl

Beschrijving In de nacht van 26 april werd in CNPP (Chernobyl Nuclear Power Plant) een test uitgevoerd waarbij gekeken werd of de koelinstallatie genoeg koeling zou leveren als de stroom even weg zou vallen. Het vermogen van de reactor werd door de crew echter zo verlaagd dat het vermogen in deze staat niet meer goed te regelen viel [9]. De test slaagte echter alsnog en de staff kreeg de opdracht om de reactor volledig uit te zetten. Toen de beschermingsstaven echter werden teruggeplaatst om de reacter helemaal uit te zetten ontstond juist bij dit lage vermogen ineens een toename van vermogen. Hierdoor werd een kettingreactie in gang gezet die uiteindelijke leidde tot een explosie met een brand die 7 doden [10] tot gevolg had en op lange termijn vele duizenden. Van de 237 brandweermannen en CNPP medewerkers die de dagen na de ramp onderzocht werden, vertoonden 134 van hen verschillende vormen van stralingsziekten. 28 van hen stierven binnen 4 maanden ondanks alle therapiën en zelfs beenmergtransplantaties. De ramp had een erg grote omvang, zo werden er radioactieve isotopen gemeten in zowel Europa, Amerika en Azië.

Datum en plaats 26 april 1986

Oorzaak De oorzaak van deze ramp is een combinatie van meerdere omstandigheden. Zowel menselijk handelen als ontwerpfouten zijn de oorzaak geweest van de explosie. Zo werd de test uitgevoerd bij een veel te laag vermogen [9]. Nadat de medewerkers de waterpompen hadden aangezet, daalde het vermogen van reactor 4 nog verder. Om dit te compenseren werden er veiligheidsstaven uit het reactorbad gehaald. Toen vervolgens de waterturbines gestopt werden omdat er opdracht was gegeven om de reactor af te sluiten, warmde het water snel op waardoor het slechter neutronen kon absorberen. Toen de medewerkers doorhadden dat het vermogen nu wel heel snel toenam, drukte een van hen de noodknop [10] in. Hoewel de noodknop er eigenlijk voor moet zorgen dat de beschermingsstaven binnen 3 seconden in het reactorbad liggen, duurde dat bij deze kerncentrale bijna 20 seconden. Dit was overigens geen fout in de aandrijving van dit noodsysteem maar een design flaw. Doordat de reactie niet snel genoeg kon worden afgeremd, ontstond er een kettingreactie. De reactor bereikte hiermee een vermogen van 30GW terwijl dit in normale omstandigheden rond de 3GW ligt. Als we naar het 4 variabelen model kijken dan zijn de actuatoren hier toch wel de schuldigen doordat het plaatsen van de beschermingsstaven zo lang duurde.

1.4.4 De crash van het AT&T netwerk

Op 15 januari 1990 crashte het netwerk van AT&T in Amerika. AT&T die verantwoordelijk was voor 70% van de langeafstandsgesprekken en reclame maakte met het meest betrouwbare en veilige netwerk lag deze dag voor 50% stil [1]. Het netwerk van AT&T bestond uit 114 computer-operated switches (4ESS) die per stuk zo'n 700.000 telefoontjes per uur konden afhandelen. De 114 4ESS's hielden van elkaar in de gaten of zij het druk hadden of dat ze telefoontjes door konden verbinden. Op die manier werd het netwerk niet op een punt belast maar werd de werkdruk verspreid. AT&T adverteerde zelfs met hun 'paranoid democracy' systeem wat inhield dat de switches bij elkaar checkten of het werk wel goed werd uitgevoerd of dat een switch bijvoorbeeld gedeeltelijk kapot was. Toch gebeurde het in de ochtend van 15 januari dat er een stuk hardware in een van de switches om onduidelijke reden kuren begon te vertonen [5]. Dit was opzich nog geen probleem omdat de 4ESS self-correcting software had. De betreffende 4ESS stuurde naar alle andere switches in het netwerk een mededeling dat hij voor de komende paar seconden geen telefoontjes kon doorverbinden, omdat de self-correcting software aan het werk was. Nadat de switch zichzelf hersteld had, gaf hij aan de dichtstbijzijnde switch (B) aan dat hij weer telefoontjes kon ontvangen. Door een recente software update kreeg switch B van switch A direct meerdere telefoongesprekken doorgeschakeld terwijl switch B nog bezig was met het bericht van switch A dat hij weer online was. Door een foute regel code kon switch B hier niet mee omgaan en moest zichzelf resetten. Het bericht dat hij offline ging stuurde hij weer naar een naastgelegen switch. Dit werd op deze manier een kettingreactie van de ene switch naar de andere en het hele netwerk werd hierdoor geraakt. Toen ontdekt werd wat de oorzaak van de storing was, heeft AT&T ervoor gezorgd dat switches die zichzelf aan het resetten waren geen berichten konden ontvangen die deze reactie op gang brachten. Dit probleem begon duidelijk bij de input variabele want er werden al telefoontjes doorgeschakeld terwijl de andere switch nog aan het werk was om het voorgaande bericht te verwerken.

1.4.5 Ariane 5 explosie

Op 4 juni 1996 vond de eerste lancering van de nieuwe Ariane 5 raket plaats. De omstandigheden waren die ochtend gunstig in Kourou alleeen het zicht was niet voldoende dus werd de lancering een uur uitgesteld. Iets na half 10 's ochtends vond de lancering plaats waarbij tot 37 seconden na de lancering alles verliep zoals gepland. Echter na 37 seconden wijzigde de raket abrupt zijn oriëntatie waardoor deze in stukken brak [4]. Toen het systeem doorkreeg dat de raket in stukken was gebroken, werd de self-destruction functie geactiveerd om schade op de grond te voorkomen. Direct na het ongeluk werden de eerste stukken data geanalyseerd om de oorzaak te achterhalen. Er bleek iets mis te zijn gegaan in het Inertial Reference System (Gekgenoeg afgekort met SRI en niet IRS). De raket was zo'n 4 kilometer boven de grond ontploft en de brokstukken lagen verspreid over 12 vierkante kilometer. Er werden zoveel mogelijk brokstukken veiliggesteld voor verdere analyse, waaronder de 2 Inertial Reference Systems. In deze raket werden namelijk meerdere

belangrijke onderdelen waaronder bijvoorbeeld de On-Board Computer (OBC) in tweevoud aangebracht (redundantie). Het ene stuk hardware is dan actief en het andere stuk hardware is 'hot'. Deze is dus ook actief maar wordt niet door de OBC gebruikt als er geen fouten worden waargenomen. Helaas had deze redundantie in dit geval weinig zin doordat het een softwarefout betrof in de SRI's die dus in beide units voorkwam en geen hardwarefalen in een van de twee units. Door een fout in de code werd een 64 bit float omgezet in een 16 signed integer. 37 Seconden na de lancering werd dit getal te groot en kon het niet meer worden opgeslagen binnen de integer, wat een overflow error tot gevolg had. Dezelfde error vond ook plaats in de andere IRS want deze draaide zoals eerder genoemd dezelfde software. Doordat de omzetting van de float naar een integer niet beschermd was (wat bij de andere omzetting wel het geval was) zorgde dit ervoor dat de mondstukken van de thrusters hun richting veranderde en de raket uit balans raakte. Het probleem begon bij de input variabele die niet zo groot verwacht werd door de software. Vervolgens werd door de overflow error verkeerde output geleverd aan de actuatoren (de mondstukken van de thrusters) die het ongeluk uiteindelijk veroorzaakte.

1.4.6 Mars Climate Orbiter crash

Op 11 december 1998 werd de Mars Climate Orbiter gelanceerd door NASA [6]. De Orbiter zou de eerste weersatelliet worden die het weer van een andere planeet kon analyseren. Ook had deze satelliet als taak om gegevens te verzamelen die kunnen verklaren hoe en of er ooit water op Mars heeft gestroomd. De hele reis verliep maanden lang precies zoals gepland totdat NASA het contact met de satelliet op 23 september 1999 verloor. De 125 miljoen dollar kostende missie ging in vlammen op toen de satelliet verbrande in de atmosfeer van Mars. Uit nader onderzoek is gebleken dat de laatst uitgevoerde manoeuvre om de satelliet op de juiste hoogte te krijgen mislukt was. De satelliet kwam door deze verkeerde stuurbeweging in een veel lagere baan terecht waardoor deze vernietigd werd. De manoeuvre mislukte doordat bij het ontwerpen van de satelliet de ingenieurs van Lockheed Martin de verkeerde eenheden gebruikten. Zij gebruikte de Brits-Amerikaanse eenheden terwijl NASA sinds 1990 al gebruik maakte van het metrieke stelsel. Door deze fout werd 4.5 maal zoveel stuwkracht gebruikt dan dat de bedoeling was. Deze menselijke fout komt in het vier variabelen model terecht bij de input variabele. De software kreeg namelijk als input van de het NASA mission control center metrische waarde ingegeven terwijl de software imperial units verwachtte. Hierdoor werd er vervolgens verkeerde output naar de boosters (actuatoren) gestuurd waardoor de satelliet in de verkeerde baan kwam. De fout ligt hier vooral in het samenwerken tussen de verschillende teams. Het opstellen van duidelijke requirements en specificaties met daarin ook de juiste eenheden gespecificeerd. Met het juist modelleren van het systeem en model checking had deze fout voorkomen kunnen worden maar dit is door bezuinigingen [6] niet volledig gebeurt.

2 Modellen

2.1 De Kripke structuur

De Kripke structuur die gebruikt wordt voor modelchecking is bedacht door Saul Kripke. De structuur maakt gebruik van verschillende states, ookwel toestanden genoemd, en overgangen tussen deze states. De structuur is een type state transition diagram. De Kripke structuur wordt gebruikt om het gedrag van safety-critical systems te controleren en te verifiëren. Belangrijk om te begrijpen is het dat states niet een moment in de tijd zijn maar een staat waarin een systeem zich bevindt. Een state kan dus een langere tijd beslaan. De toestand bevat waarden van alle variabelen van het systeem gedurende dat tijdsinterval. Wanneer een systeem 'werkend' is, modelleert men dit middels het doorlopen van meerdere toestanden. De overgang tussen deze states worden transities genoemd.

2.2 Soorten modellen

Onder de state transition diagrams heb je veel verschillende soorten. Zoals eerder genoemd de Kripke structuur maar ook labeled state transition diagrams, timed state transition diagrams, input-output state transition diagrams en combinaties hiertussen. De normale state transition diagram is de 'kale versie'. Wanneer er ook labels gebruikt worden om de namen van de toestanden aan te geven heet dit een labeled state transition diagram. Er bestaan ook structuren waar tijd een belangrijk rol speelt en dit zijn dan timed state transition diagrams. Een combinatie tussen de 2 eerder genoemde states is een labeled timed state transition diagrams. Vervolgens bestaan er ook structuren waar rekening gehouden moet worden met input en output, deze structuren heten heel toepasselijk input-output state transition diagram.

2.3 Tijd

In Uppaal wordt de tijd bijgehouden doormiddel van klokken die beginnen vanaf 0. Het is niet mogelijk om als begintijd bijvoorbeeld 5 te kiezen. Al deze klokken lopen even snel en kunnen worden uitgelezen en verwerkt door invarianten en guards. Klokken kunnen tevens ook gereset worden in transities. De tijd die gebruikt wordt, wordt gezien als een continu verschijnsel. Het is dus ook mogelijk om slechts een stukje van de tijdseenheid te gebruiken. Tijd verstrijkt in Uppaal alleen in toestanden en dus niet in transities. De concrete simulator is binnen Uppaal een handige tool als je wat meer wilt focussen op de tijd waarin states geactiveerd worden etc.

2.4 Guards en invarianten

Invarianten en guards zijn condities. Een invariant is een conditie die geldt in een bepaalde state en die altijd waar is wanneer een systeem zich in die state bevindt. Een invariant is dus eigenlijk een uitspraak die waar moet zijn wanneer het systeem zich in die betreffende toestand bevindt. Er kan dus nooit een transitie plaatsvinden

tussen state A en state B als niet voldaan wordt aan de invariant. Een invariant verandert niet en er zijn ook geen uitzonderingen op. Invarianten kunnen meerdere variabelen gebruiken. Een guard is een conditie die geldt tijdens een transitie. De transitie kan dus alleen plaatsvinden als de guard geldig/waar is. Het verschil tussen de guards en de invarianten is dat invarianten gelden voor de state zelf en de guards voor de transitie naar een state toe.

2.5 Deadlock

Een deadlock is een situatie waarin een systeem vast loopt. Een deadlock kan ontstaan wanneer de code in een systeem bijvoorbeeld een bepaalde variabele nodig heeft om een bepaalde bewerking uit te kunnen voeren. Wanneer deze variabele niet bestaat of niet beschikbaar is, kan de bewerking niet worden uitgevoerd. Het programma komt op zo'n moment tot stilstand en kan niet verder. In model checking zit dit iets anders in elkaar. In Uppaal geldt dat deadlocks ontstaan wanneer een combinatie van invarianten en guards het verstrijken van tijd verhindert. Wanneer een invariant bijvoorbeeld aangeeft dat een systeem uit de state vertrokken moet zijn terwijl de klok kleiner dan 2 is maar vervolgens de guard zegt dat de transitie alleen plaats kan vinden als de klok groter is dan 2. Dit type situatie heet een deadlock omdat het systeem door tegenstrijdigheden niet verder kan.

2.6 Zeno gedrag

Zeno gedrag, vernoemd naar de Griekse filosoof Zeno van Elea, houdt in dat er binnen een systeem een oneindig aantal states doorlopen kan worden in een eindige hoeveelheid tijd. Zeno gedrag kan voorkomen worden door gebruik te maken van klokken en guards. De guard kan bijvoorbeeld alleen waar zijn wanneer de klok een gehele waarde geeft en vervolgens kan de state de klok weer resetten.

3 Logica

3.1 Propositielogica

Proposities zijn uitspraken (in dit geval over een systeem) die waar of onwaar zijn. Het probleem bij ons gebruik is dat een bepaalde propositie P in de ene state waar kan zijn maar in de andere state niet. Proposities zijn niet 'sterk genoeg' om uitspraken te doen over gehele systemen waarin dingen steeds veranderen. Propositielogica werkt wat dat betreft alleen met statische systemen.

3.2 Predicatenlogica

Predicatenlogica is een uitbreiding op de propositielogica. Zo bevat de predicatenlogica functiesymbolen en variabelen. Verder bevat de predicatenlogica 2 kwantoren die niet bestaan in de propositielogica.

3.3 Kwantoren

$\forall$: universele kwantor
$\exists$: existentiële kwantor
Kwantoren zijn onmisbaar in de logica. Hieronder heb ik de 2 belangrijkste voor dit moment genoteerd met een voorbeeld.

$\forall$xP(x): "Voor alle x geldt P(x).Oftewel: Voor alle X in een verzameling van x'en, is predicaat P van toepassing.

$\exists$xP(x): "Er is een x, waarvoor geldt: P(x).Oftewel: Voor een aantal/sommige X in een verzameling van x'en, is predicaat P van toepassing.

3.4 Dualiteiten

In kwantoren zijn dualiteiten aanwezig. Dit wil zeggen dat de ene kwantor op te bouwen is uit meerdere andere kwantoren. Aan de hand van een voorbeeld:

$\neg\forall$xP(x) $\equiv$ $\exists$x$\neg$P(x) betekent bijvoorbeeld: Niet alle vissen (x) zijn goudvissen, er geldt dus een vis (x) die geen goudvis is.

$\neg\exists$xP(x) $\equiv$ $\forall$x$\neg$P(x) betekent bijvoorbeeld: Er is niet een docent (x) perfect, voor alle docenten (x) geldt dat ze niet perfect zijn.

4 Computation tree logic

Computation tree logic, afgekort Ctl, is een vorm van temporele/tijdelijke logica. Temporele logica is een type logica die zich bezig houdt met systemen waarin veranderingen plaatsvinden. Uppaal werkt met een subset van Ctl. Ctl maakt het mogelijk om logische uitspraken te doen over de computation tree van een Kripke structuur. Met computation tree logic is het mogelijk om een systeem volledig geautomatiseerd te laten checken.

4.1 De computation tree

Een computation tree is een boomachtige structuur die het pad volgt dat het systeem doorloopt en deze states achter elkaar plaatst. Wanneer het systeem erg simpel is en slechts een paar states kent die achter elkaar doorlopen worden, ziet de sturctuur eruit als een lange sliert. Een vertakking wil zeggen dat er meerdere opties of paden zijn die genomen kunnen worden binnen het systeem. Doordat we reactieve systemen modelleren is de computation tree oneindig groot. Kortgezegd is een computation tree een logische weergave van een systeem in werking.

4.2 Operator: AG

A: selecteer alle paden (all ways)
G: alle states in het pad (globally)
AG(P): Pak alle paden uit de computation tree en pak op elk van die paden alle states, daarin is P geldig. P is dus altijd overal geldig in het systeem.

4.3 Operator: EG

E: sommige paden (exists)
G: alle states in het pad (globally)
EG(P): Pak niet alle paden / sommige paden uit de computation tree en pak op elk van die paden alle states, daarin is P geldig. P is dus nooit overal geldig in het systeem.

4.4 Operator: AF

A: selecteer alle paden (all ways)
F: op een gegeven moment op het pad (eventually)
AF(P): Pak alle paden uit de computation tree en pak op elk van die paden ergens een state, daarin is P geldig. P is dus in alle paden een keer geldig in het systeem.

4.5 Operator: EF

E: sommige paden (exists)
F: op een gegeven moment op het pad (eventually)
EF(P): Pak niet alle paden / sommige paden uit de computation tree en pak op elk van die paden ergens een state, daarin is P geldig. P is dus in sommige paden een keer geldig in het systeem.

4.6 Operator: AX

A: selecteer alle paden (all ways)
X: de eerst volgende state (next)
AX(P): Pak alle paden uit de computation tree en pak op elk van die paden niet de begin state maar de eerst volgende state, daarin is P geldig. P is dus in elk pad na de beginstate, een state (de eerste na de beginstate) geldig in het systeem.

4.7 Operator: EX

E: sommige paden (exists)
X: de eerst volgende state (next)
EX(P): Pak niet alle paden / sommige paden uit de computation tree en pak op elk van die paden niet de begin state maar de eerst volgende state, daarin is P geldig. P is dus in sommige paden na de beginstate, een state (de eerste na de beginstate) geldig in het systeem.

4.8 Operator: p U q

p: propositie p
q: propositie q
U: totdat (until)
propositie p is geldig totdat q geldig is. Een voorbeeld is A(p U q): Voor elk pad is p geldig totdat q ergens in dat pad geldig is. Q hoeft in het betreffende pad daarna niet eindeloos geldig te zijn.

4.9 Operator: p R q

p: propositie p
q: propositie q
R: aflossen (releases)
propositie p is geldig totdat na een tijd p en q een state geldig zijn. Na deze aflossings state is alleen q nog maar geldig. Een voorbeeld is A(p R q): Voor elk pad is p geldig totdat q en p ergens in dat pad geldig zijn (voor de duur van 1 state). Hierna is q in het betreffende pad geldig en p niet meer. Q hoeft in het betreffende pad daarna niet eindeloos geldig te zijn. Dit alles lijkt een beetje op soldaten die elkaar aflossen bij de wacht. Soldaat A staat op wacht en soldaat B komt soldaat A aflossen. Op een gegeven moment staan de soldaten dus samen. Na dit korte samenkomen gaat soldaat A weg en neemt soldaat B de wacht over.

4.10 Fairness

A: selecteer alle paden (all ways)
G: alle states in het pad (globally)
F: op een gegeven moment op het pad (eventually)
AG(AF(P)): in welke state een systeem zich ook bevindt, in alle richtingen kom je vroeg of laat een state tegen waarin p geldig is. p blijft steeds gebeuren, p vindt dus oneindig vaak plaats. Fairness betekend letterlijk dat elke state in een systeem aandacht zal krijgen. Wanneer deze state 'aandacht' heeft gekregen zal dit in de toekomst nog vaker plaatsvinden. Fairness geldt bijvoorbeeld voor besturingssystemen en wordt ook wel freedom of starvation genoemd.

4.11 Liveness

A: selecteer alle paden (all ways)
G: alle states in het pad (globally)
F: op een gegeven moment op het pad (eventually)
AG(p $\rightarrow$ AF(q)): altijd en overal geldt p $\rightarrow$ AF(q) (als p geldt dan geldt vroeg of laat q). Welk pad er ook genomen wordt: als p geldig is, is vroeg of laat q geldig. Dit wordt liveness genoemd. Voorbeeld:
- Vul voor p 'true' in
- we krijgen nu AG(true $\rightarrow$ AF(q))

- en dus AG(AF(q)) (fairness)
Te concluderen valt dat liveness een generalisatie is van fairness.

Referenties

[1] Dennis Burke. All circuits are busy now: The 1990 at&t long distance network collapse. *California Polytechnic State University*, 1995.

[2] N. Leveson. *Medical Devices: The Therac-25*. University of Washington, Washington, USA, 1995.

[3] Nancy G Leveson and Clark S Turner. An investigation of the therac-25 accidents. *IEEE computer*, 26(7):18–41, 1993.

[4] Prof. J. L. LIONS. Ariane 5 flight 501 failure report by the inquiry board, 1996.

[5] Peter G. Neumann. Cause of at&t network failure. *The Risks Digest*, 9(62):11, 1990.

[6] J. Oberg. Why the mars probe went off course [accident investigation]. *IEEE Spectrum*, 36(12):34–39, 1999.

[7] Lucian M. Patcas, Mark Lawford, and Tom . Maibaum. *Electronic Communications of the EASST Volume 66*. McMaster University, Department of Computing and Software, McMaster University, Canada, 2013.

[8] mr. J.W. Selles prof. mr. Pieter van Vollenhoven, mr. J.A. Hulsenbek. *Neergestort tijdens nadering, Boeing 737-800, nabij Amsterdam Schiphol Airport, 25 februari 2009*. Onderzoeksraad, Anna van Saksenlaan 50, 2593HT, Den Haag, 2009.

[9] Yu. Ustyantsev V. Kortov. *Chernobyl accident: Causes, consequences and problems of radiation measurements*. Physical & Engineering Institute, Ural Federal University, , 19 Mira Str., 620002 Ekaterinburg, Russia, 2012.

[10] A. Tsyb T. Bogdanova M. Tronko Yu. Demidchik S. Yamashita V. Saenko, V. Ivanov. *The Chernobyl Accident and its Consequences*. Elsevier Ltd., 2011.