

# Verslag Tinlab Advanced Algorithms

**Galvin Bartes 0799967**  
176-671

8 december 2023



# Inhoudsopgave

|          |                                                            |           |
|----------|------------------------------------------------------------|-----------|
| <b>1</b> | <b>Inleiding</b>                                           | <b>2</b>  |
| <b>2</b> | <b>Theoretisch kader</b>                                   | <b>3</b>  |
| 2.1      | Begrippen, tools en literatuur . . . . .                   | 3         |
| 2.1.1    | Requirements . . . . .                                     | 6         |
| 2.1.2    | Specificaties . . . . .                                    | 10        |
| 2.2      | Requirements en specificaties . . . . .                    | 11        |
| 2.2.1    | Safety critical systems . . . . .                          | 11        |
| 2.3      | Rampen . . . . .                                           | 12        |
| 2.3.1    | Therac-25 . . . . .                                        | 12        |
| 2.3.2    | Ethiopian Airlines Flight 302,boeing 737 crashes . . . . . | 13        |
| 2.3.3    | China explosie 2015 Tianjin . . . . .                      | 14        |
| 2.3.4    | schipholbrand . . . . .                                    | 15        |
| 2.3.5    | 1951 . . . . .                                             | 15        |
| 2.3.6    | slmramp . . . . .                                          | 16        |
| 2.3.7    | Tsjernobyl . . . . .                                       | 16        |
| 2.4      | De Kripke structuur . . . . .                              | 17        |
| 2.4.1    | CTL . . . . .                                              | 17        |
| 2.5      | Guards en invarianten . . . . .                            | 17        |
| 2.6      | Deadlock . . . . .                                         | 18        |
| 2.7      | Zeno gedrag . . . . .                                      | 18        |
| <b>3</b> | <b>Logica</b>                                              | <b>18</b> |
| 3.1      | Propositie logica . . . . .                                | 18        |
| 3.2      | Predicatenlogica . . . . .                                 | 19        |
| 3.3      | Kwantoren . . . . .                                        | 19        |
| 3.4      | Dualiteiten . . . . .                                      | 20        |
| 3.5      | Operator: AG . . . . .                                     | 20        |
| 3.6      | Operator: EG . . . . .                                     | 21        |
| 3.7      | Operator: AF . . . . .                                     | 21        |
| 3.8      | Operator: EF . . . . .                                     | 21        |
| 3.9      | Operator: AX . . . . .                                     | 21        |
| 3.10     | Operator: EX . . . . .                                     | 21        |
| 3.11     | Operator: $p \cup q$ . . . . .                             | 21        |
| 3.12     | Operator: $p \cap q$ . . . . .                             | 22        |
| 3.13     | Fairness . . . . .                                         | 22        |
| 3.14     | Safety . . . . .                                           | 22        |
| 3.15     | liveness properties . . . . .                              | 22        |

# 1 Inleiding

In dit verslag ga ik in op de kennis en achtergrondinformatie die nodig is voor het toepassen van Time-based model technieken. Door de toenemende complexiteit van systemen is het gebruik van modellen en de toepassing van timebased model checking op industriële controle systemen een manier van modelleren van het systeem en de requirements zodat er een bijdrage kan worden geleverd aan de acceptatie van simulatie-/modeltechniek voor de industrie.[?]. De behandelde onderwerpen zijn requirements-engineering, computational Tree Logic, propositielogica en predicaatlogica.

## 2 Theoretisch kader

In dit hoofdstuk houdt ik me bezig met de bestudering van rampen aan de hand van het vier-variabelen model maakt het analyseren mogelijk van rampsituaties. Van een aantal rampen is een beschrijving gegeven met datum, plaats en oorzaak. De analyse van de 4-variabelen modellen zal gebruikt worden voor de requirementsdefinitie, ontwerp en ontwikkeling van het sluismodel.

### 2.1 Begrippen, tools en literatuur

**Wat is uppaal** Uppaal is een geïntegreerde toolomgeving voor het modelleren, simuleren en verifiëren van real-time systemen, gezamenlijk ontwikkeld door Basic Research in Computer Science aan de Universiteit van Aalborg in Denemarken en de afdeling Informatietechnologie aan de Universiteit van Uppsala in Zweden. Het is geschikt voor systemen die kunnen worden gemodelleerd als een verzameling niet-deterministische processen met een eindige controlestructuur en klokken met reële waarde, die communiceren via kanalen of gedeelde variabelen. Typische toepassingsgebieden zijn met name real-time controllers en communicatieprotocollen, waarbij timingaspecten van cruciaal belang zijn.

**Wat is statistical model checking?** Dit verwijst naar verschillende technieken die worden gebruikt voor de monitoring van een systeem. Daarbij wordt vooral gelet op een specifieke eigenschap. Met de resultaten van de statistieken wordt de juistheid van een ontwerp beoordeeld. Statistisch model checking wordt onder andere toegepast in systeembioologie, software engineering en industriële toepassingen. [?]

**Waarom gebruiken we statistisch model checking?** Om de bovenstaande problemen te overwinnen stellen we voor om te werken met Statistical Model Checking, een aanpak die onlangs is voorgesteld als alternatief om een uitputtende verkenning van de toestandsruimte van het model te vermijden. Het kernidee van de aanpak is om een aantal simulaties van het systeem uit te voeren, deze te monitoren en vervolgens de resultaten uit het statistische gebied te gebruiken (inclusief het testen van sequentiële hypothesen of Monte Carlo-simulaties) om te beslissen of het systeem aan de eigenschap voldoet of niet. mate van vertrouwen. Van nature is SMC een compromis tussen testen en klassieke modelcontroletechnieken. Het is bekend dat op simulatie gebaseerde methoden veel minder geheugen- en tijdsintensief zijn dan uitputtende methoden, en vaak de enige optie zijn. [?] Alternatieve tools voor Uppaal zijn Asynchronous Events, Vesta en MRMC.

**MODE CONFUSION** Mode confusion treed op als geobserveerd gedrag van een technisch systeem niet past in het gedragspatroon dat de gebruiker in zijn beeldvorming heeft en ook niet met voorstellingsvermogen kan bevatten.

**Wat is automatiseringsparadox** Gemak dient de mens. Als er veel energie wordt gestoken in de ontwikkeling van hulpmiddelen die taken van werknemers overemen heeft dat tot resultaat dat veel productieprocessen worden geautomatiseerd. De vraag is dan of vanuit mechnisch wereldpunt de robot niet de rol van de mens overneemt en of de mens nog de kwaliteiten heeft om het werk zelf te doen. [?]

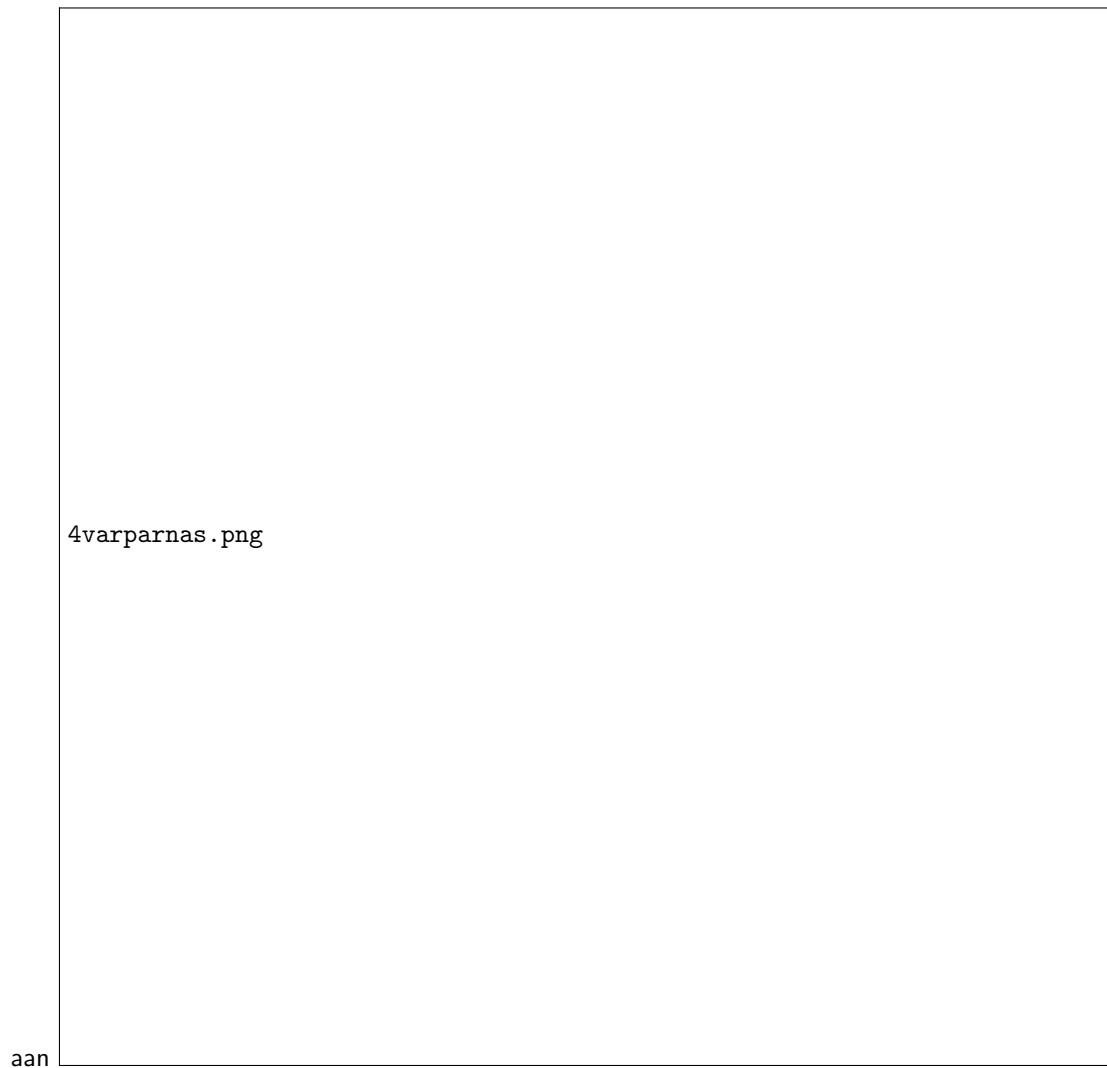
**4 variabelen model** Er zijn veel veiligheidskritische computersystemen nodig voor het bewaken en besturen van fysieke processen. Het viervariabelenmodel, dat al bijna met succes in de industrie wordt gebruikt veertig jaar, helpt het gedrag van en de grenzen tussen het fysieke te verduidelijken processen, invoer-/uitvoerapparaten en software. [?]

Het 4 variabelen model kort toegelicht Monitored variabelen: door sensoren gekwantificeerde fenomenen uit de omgeving, bijv temperatuur

Controlled variabelen: door actuatoren bestuurde fenomenen uit de omgeving Gecontroleerde variabelen kunnen bijvoorbeeld de druk en de temperatuur zijn in een kernreactor, terwijl gecontroleerde variabelen ook visuele en hoorbare alarmen kunnen zijn als het uitschakelsignaal dat een reactorsluiting initieert; wanneer de temperatuur of druk bereikt abnormale waarden, de alarmen gaan af en de uitschakelprocedure wordt gestart

Input variabelen: data die de software als input gebruikt Hier modelleert IN de ingangshardware-interface (sensoren en analoog-naar-digitaal-omzetters) en relateert waarden van bewaakte variabelen aan waarden van invoervariabelen in de software. De invoervariabelen modelleren de informatie over de omgeving die beschikbaar is voor de software. Bijvoorbeeld, IN zou een druksensor kunnen modelleren die temperatuurwaarden omzet in analoge spanningen; Deze spanningen worden vervolgens via een A/D-omzetter omgezet in gehele waarden die zijn opgeslagen in een register dat toegankelijk is voor de computer software.

Output variabelen: data die de software levert als output De uitgangshardware-interface (digitaal-naar-analoog converters en actuatoren) is gemodelleerd door OUT, dat waarden van de uitvoervariabelen van de software relateert aan waarden van gecontroleerde variabelen. Een uitvoervariabele kan bijvoorbeeld een Booleaanse variabele zijn die door de software is ingesteld met de begrippen dat de waarde true aangeeft dat een reactorsluiting zou moeten plaatsvinden en de waarde false geeft het tegenovergestelde



**6 Variable model** Een uitbreiding van een 4-variabelen model is het 6-variabelen model. Optitatie statements omschrijven de omgeving zoals we het willen zien vanwege de machine. Indicatieve statements omschrijven de omgeving zoals deze is los van de machine. Een requirement is een optitatie statement omdat ten doel heeft om de klantwens uit te drukken in een softwareontwikkel project. Domein kennis bestaat uit indicatieve uitspraken die vanuit het oogpunt van software ontwikkeling relevant zijn. Een specificatie is een optitatie statement met als doel direct implementeerbaar te zijn en ter verondersteuning van het nastreven en realiseren van de requirements. Drie verschillende type domein-kennis: domein eigenschappen, domein hypothesen, en verwachtingen. Domein eigenschappen zijn beschrijvende statementsover een omgeving en zijn feiten. Domein hypothesen zijn ook beschrijvende uitspraken over een omgeving, maar zijn aannames. Verwachtingen zijn ook aannames, maar dat zijn voorschrijvende uitspraken die behaald worden door actoren als personen, sensoren en actuators.


### 2.1.1 Requirements

1.5 Definition of Requirements Engineering “Requirements engineering [DoD 1991] involves all lifecycle activities devoted to identification of user requirements, analysis of the requirements to derive additional requirements, documentation of the requirements as a specification, and validation of the documented requirements against user needs, as well as processes that support these activities.” Note that requirements engineering is a domainneutral discipline; e.g., it can be used for software, hardware, and electromechanical systems. As an engineering discipline, it incorporates the use of quantitative methods, some of which will be described in later chapters of this book. Whereas requirements analysis deals with the elicitation and examination of requirements, requirements engineering deals with all phases of a project or product life cycle from innovation to obsolescence. Because of the rapid product life cycle (i.e., innovation→development→ release→maintenance→obsolescence) that software has enabled, requirements engineering has further specializations for software. Thayer and Dorfman [Thayer et al. 1997], for example, define software requirements engineering as “the science and discipline concerned with establishing and documenting software requirements.”

Schermafbeelding 2023-12-08 133303.png



Schermafbeelding 2023-12-08 132959.png



Schermafbeelding 2023-12-08 132840.png

1.7 Characteristics of a Good Requirement Requirements characteristics are sometimes overlooked when defining requirements processes. They can be an excellent source of metrics for gauging project progress and quality. One question we typically ask organizations when discussing their quality processes is, "Given two requirements specifications, how would you quantitatively determine that one is better than the other?" This question may be answered by looking at the IEEE 830 Standard [IEEE 1998]. The characteristics of a good requirement, as defined by the IEEE, are listed next, with several additional useful ones. It is important to distinguish between the characteristics of a requirement and the characteristics of a requirements specification (a set of related requirements). In some cases a characteristic can apply to a single requirement, in some cases to a requirements specification, and in other cases to the relationship of two or more requirements. Furthermore, the meaning may be slightly different when referring to a requirement or a specification. Care must be taken, therefore, when discussing the characteristics described here to define the context of the attributes.

## 1.9 Quality and Metrics in Requirements Engineering

A quality attribute requirement (QAR) is specified in terms of observable, usually measurable, characteristics of the system that indicate its fitness for use. Quality attributes may be thought of as modifiers of the functional requirements that indicate how they are achieved. Quality attribute requirements address all “uses” of the system, including those where the system is a passive object rather than an active participant, such as when the system is being sold or when the next version of the system is being developed. Examples of quality attributes include: capacity, security, usability, cost, modifiability, and fault tolerance. The term nonfunctional requirement, although still commonly used, has become a synonym for quality attribute requirement.

**5.3 Quality Attribute Requirements** The road to understanding quality attribute requirements starts with a brief detour into the fundamentals of system quality. The quality of a system, in general, is its fitness for its intended uses. ISO Std. 9126-1 defines a quality model with four linked topic areas of quality: • Process quality Quality of the process that is producing the product • Internal quality Quality of the intermediate work products (some of which may also be deliverable work products) • External quality Quality of the finished product, before delivery • Quality in use Quality of the larger processes in which the delivered product is used A quality attribute is a system or process property indicative of quality in any of these quality topic areas. Note that, for our purposes, the “process” in “process quality” includes not only software development, but all the business functions surrounding the product, including marketing, sales, planning, maintenance, installation, customer support, and preparing to develop the next version. Naturally, quality in use is the most important area of quality, but it is also the latest one to measure, because it cannot be measured until the product is delivered. Fortunately, quality attributes in the other topic areas give us useful indications of what the quality in use will be; i.e., we say that such quality attributes are “indicative of” quality in use.

With these examples in mind, we summarize four related terms: • Quality Fitness for one or more defined uses • Quality attribute A property of the system or the process that is indicative of quality • Quality attribute measure A way of measuring a quality attribute for a specific system or process • Quality attribute requirement A requirement expressed in terms of one or more quality attribute measures Table 5.1 gives a broad sampling of other quality attribute topics you might want to consider, with an example measure pertaining to each topic. The topics are drawn from ISO/IEC Std. 9126, but there are many other good sources of topics available on the Internet. The quality attribute measures you use will be very specific to your project, as you will see when we discuss quality attribute workshops.

### 2.1.2 Specificaties

Formal Model: Specification is used to give a formal description of the system which is being analysed. The formal specification considers a language with a mathematically defined syntax and semantics. In this paper we represent the system behaviour as a network of timed automaton i.e., a finite-state machine extended with clock variables [11]. A system is modelled as a network of timed automata in parallel. The model is extended with bounded discrete variables that are part of the state. A state of the system is defined by the location of all automata, the clock values and the values of the discrete variables. Automaton may fire an edge (i.e., perform a transition) separately or synchronise with another automaton with the aim to lead a new state.

Below we itemize the timed automata distinctive elements: Guard: a particular expression aimed to verify a boolean expression evaluated on edges; Invariant: a condition that indicates the time that

can be spent on a node; Channel: considered for synchronizing the progress of two or more automata. Temporal Logic: to define properties we need a precise notation.

Formal Verification Environment: once defined the model and the temporal logic properties, we need something enabling us to check whether the timed automata based model satisfies the defined properties. To this aim we consider formal verification, a system process exploiting mathematical reasoning to verify if a system (i.e., the model) satisfies some requirement (i.e., the timed temporal properties).

## 2.2 Requirements en specificaties

Requirements en specificaties! Assuming that the requirements are feasible with respect to the environment, Parnas and Madey [3] proposed the following acceptability condition for the software:  $NAT (NAT;SOF;OUT) REQ$  Here, the operator  $;$  is the usual composition of relations. A system implementation  $sys = IN;SOF;OUT$ ; is then acceptable if and only if it satisfies the following condition:  $NAT SYS REQ$  <https://www.sciencedirect.com/science/article/pii/S0167642315001033> Functional vs non functional requirements

### 2.2.1 Safety critical systems

De kennis van de mens is verantwoordelijk voor het scheppen van kennis. Opdoen van kennis van veiligheidsintegriteit ten overstaan van onwetendheid. Kennisdoelen zijn daarom altijd belangrijk en kunnen worden onderscheiden in 3 domeinen: cognitief, affectief en psychomotorisch.[?] Een Veiligheidsanalyse is een manier voor het evalueren van ongelukken en risico's op verschillende niveaus. Met een veiligheidsanalyse wordt gekeken naar mensen en systemen die worden blootgesteld aan een gevaar/risico en de mogelijkheid deze te minimaliseren. Veiligheid is moeilijk te definiëren omdat het een vaag concept is. Een veiligheidsanalist moet kennis hebben van het systeem, maar ook ervaring met het systeem en vooral hoe een fout in het systeem zich voordoet. Volgens Stoney[?] is veiligheid een aspect dat de veiligheid van mensenlevens of de omgeving niet in gevaar brengt. De interactie tussen systeem en omgeving levert kennis op door ad-hoc ervaring. Maar ook is kennis van abstractie en systeemdoelen belangrijk. Het kwalificeren van informatie moet ook op waarde geschat worden Een verkeerde schatting kan fataal zijn. Zelfs als een veiligheidsanalyse is gedaan waarbij risico's zijn gedefinieerd en geprioritiseerd is een definitie van een veiligheidsniveau nodig. Soms zijn hier richtlijnen voor en soms ook niet. Er zijn voorbeelden van tools en technieken te gebruiken voor een veiligheidsanalyse. En dat zijn

1. checklists voor critical safety items
2. fault tree analysis
3. event tree analysis
4. failure modes en critical effects analysis (FMECA, FMET)
5. hazard operability studies

[?]



Schermafbeelding 2023-12-08 144723.png

[?] [?] [?] [?] [?] [?].

## 2.3 Rampen

Voor deze studie is onderzoek gedaan naar verschillende rampen aan de hand van het vier variabelen model. Elke ramp op deze manier categoriseren kan ons helpen te bepalen in hoeverre requirements een rol kunnen spelen in de veiligheid van ons model.

### 2.3.1 Therac-25

**Beschrijving** In de periode van Juni 1985 and Januari 1987 zijn er meerdere ongelukken met dodelijke afloop door de implementatie van de Therac-25 bij de behandeling van huidkanker. Dit apparaat gebruikt

elektronen om stralen met hoge energie te creëren die tumoren kunnen vernietigen met minimale impact op het omliggende gezonde weefsel.

**Datum en Plaats** De ongelukken vonden plaats in de periode van juni 1985 tot en met januari 1987.

**Oorzaak** Onderzoekers constateren dat er fouten zijn gemaakt tijdens de (her-)implementatie van systemen uit eerdere productiemodellen. Terwijl de therac 20 afhankelijk was van mechanische vergrendelingen werd er bij de therac-25 software gebruikt. Onderzoekers komen daarom tot de volgende conclusies Software problemen zijn onder andere:

- slechte software engineering/designing praktijken
- de machine is afhankelijk van software voor veiligheidsoperaties
- de fout in de code is niet zo belangrijk als een geheel onveilig ontwerp
- het reinigen van de buigmagneetvariabele in plaats van aan het uiteinde van het frame
- raceconditionering om aan te geven dat het invoeren van het recept nog steeds aan de gang is
- reactie van de gebruiker
- slechte subroutines voor schermvernieuwing die foutieve informatie op de werkende console achterlieten
- Problemen met het laden van tapes bij het opstarten, waarbij het gebruik van photom-tabellen werd uitgesloten om het interlock-systeem te activeren in het geval van een laadfout in plaats van een checksum

Onderzoekers zijn van mening dat de tekortkomingen in medische apparatuur niet geheel en altijd te verwijten zijn aan softwareproblemen. Zo is er ook een rol weggelegd voor fabrikanten en overheden. Klankbordgroepen klagen over het tekort aan software-evaluaties en een tekort aan hard-copy audit trials om foutmeldingen in beeld te krijgen. [?]

### 2.3.2 Ethiopian Airlines Flight 302, boeing 737 crashes

**Beschrijving** Ethiopische Airlines-vlucht 302 was een geplande internationale passagiersvlucht van Bole International Airport in Addis Abeba, Ethiopië naar Jomo Kenyatta International Airport in Nairobi, Kenia. Op 10 maart 2019 stortte het Boeing 737 MAX 8-vliegtuig dat de vlucht uitvoerde zes minuten na het opstijgen neer nabij de stad Bishoftu, waarbij alle 157 mensen aan boord omkwamen.

Vlucht 302 is het dodelijkste ongeval van Ethiopië Airlines tot nu toe en overtreft de fatale kaping van vlucht 961, resulterend in een crash nabij de Comoren in 1996. Het is ook het dodelijkste vliegtuigongeluk dat zich in Ethiopië heeft voorgedaan, en overtreft de crash van een Antonov An-26 van de Ethiopische luchtmacht in 1982, waarbij 73 mensen omkwamen.

Dit was het tweede MAX 8-ongeluk in minder dan vijf maanden na de crash van Lion Air-vlucht 610. Beide crashes waren aanleiding voor een twee jaar durende wereldwijde langdurige aan de grond houden van het vliegtuig en een onderzoek naar de manier waarop het vliegtuig werd goedgekeurd voor passagiersvervoer.

**Datum en Plaats** De ramp vond plaats op 10 maart 2019 nabij de stad Bishoftu

**Oorzaak** De technische oorzaak ligt bij het MCAS flight control system. Dit systeem werd geïmplementeerd om kosten te reduceren en opleidingen voor piloten in te korten. Deze single point of failure [?] werd getriggerd door een enkele angle-of-attack sensor[?]. Bij de nieuwe boeing 737 max model werden tests uitgevoerd in volledige flight simulators. Nieuwe regels van het FAA instituut vereisten ondersteuning bij het uitvoeren van enkele manoeuvres. Tijdens testvluchten uitgevoerd binnen een jaar voor certificatie werd het pitch-up fenomeen geconstateerd waarop het mcas systeem werd aangepast. In het systeem zaten nu de volgende fouten.

- MCAS wordt getriggerd voor enkele sensor zonder vertraging
- Het ontwerp staat toe dat in situaties waar de angle-of-attack fout is de mCAS wordt geactiveerd
- systeem kreeg onnodig bevoegdheid controle om de neus bij te sturen
- Waarschuwingslicht bij fouten in de angle-of-attack werkte niet door softwarefout. Het werd ook niet kritisch bevonden door ethiopian airlines. geplande updates door boeing pas in 2020
- een losstaande fout in de microprocessor van de controle computer kan vergelijkbare situaties doen voorkomen zonder dat mcas wordt geactiveerd

Fouten vielen niet op omdat FAA test uitbesteedde aan Boeing. Contact tussen de organisaties FAA en Boeing verliep op management niveau. Boeing instrueerde niet alle piloten over MCAS. En het MCAS systeem werd gezien als een achtergrond systeem.

Uit onderzoek is op te maken dat problemen bij boeing toestellen te verwijten zijn aan:

- marktdruk
- slapshot engineering
- missiekritische toepassing van technologie
- geen regelgevingsregime voor foutieve risicoanalyse

### 2.3.3 China explosie 2015 Tianjin

**Beschrijving** De explosie zorgde voor de vernietiging van 12000 voertuigen, schade aan 17000 huizen binnen een straal van 1 km. Er waren 173 doden inclusief brandweermensen. Een van de explosies zorgde voor een beving van 2.3 op de schaal van rigter. Opgeslagen materialen waren: calcium carbide, sodium nitraat, potassium nitraat, ammoniak nitraat en cyanide. Ook is er veel kritiek geweest op de acties van de autoriteiten. Zo was er censuur vanuit de overheid op de journalistiek. Ook was er naar alle waarschijnlijkheid sprake van corruptie. Zo bleek achteraf dat een van de grootste aandeelhouders Dong Shexuang de zoon te zijn van een oud-politiefichef in Tanjin haven, genaamd Dong Pijun

**Datum en Plaats** De ramp vltrok zich pp 12 augustus 2015 bij de Rulthai logistiek. Deze faciliteit zorgde voor de opslag van gevaarlijke stoffen.

**Oorzaak** De volgende factoren zouden een rol hebben gespeeld:

- Een onjuiste afbakening van het opslagmateriaal
- Er was weinig kennis bij de autoriteiten over opslagmaterialen. Zo bleek er 7000 ton aan materiaal opgeslagen, dat is ruim 70 keer te maximaal toegestane hoeveelheid.
- Onverenigbaar grondgebruik in de nabije omgeving. Veel woonwijken met naar schatting 6000000 bewoners en 500 lokale bedrijven in de buurt van de opslag gevaarlijke stoffen.

### 2.3.4 schipholbrand

**Beschrijving** Bij de schipholbrand op 27/10/2005 vielen zeker 11 doden onder migranten in de cellencomplexen van schiphol-oost. Doodsoorzaak van de slachtoffers is verstikking. Het gebouw voldeed niet aan de eisen voor brandveiligheid, personeel was niet goed getraind voor dergelijke situaties en de hulpverlening kwam door verschillende factoren te laat op gang.

**Datum en Plaats** De ramp voltrok zich bij de vleugels j en k van het cellencomplexen van schiphol-oost op 27 oktober 2005 .

**Oorzaak** Doordat de deur van cel 11 niet werd gesloten kreeg de brand zuurstof. De brand kon zich uitbreiden in de aanwezigheid van grote hoeveelheid brandbare materialen. Deze fungeerde als brandstof waardoor de brand zich verder kon ontwikkelen zowel binnen als buiten de cel. De dakluiken werkten met als gevolg dat de rook en warmte niet werden afgevoerd. Dit heeft het mede onmogelijk gemaakt alle celdeuren te openen en daardoor de reddingsoperatie van de bewaarders belemmerd. Door de schilconstructie van het cellencomplex was het mogelijk dat de brand zich kon uitbreiden naar de andere cellen en naar de gang.

Dit is onopgemerkt gebleven doordat er geen specifieke eisen werden gesteld bij de bouw van vleugels j en k. Was er te weinig aandacht voor bouwregels en onderzoek naar eerdere branden in het cellencomplex. Bovendien waren bHV'ers niet goed opgeleid en was er geen goede afstemming met de brandweer. Er was geen evacuatieplan dat dat rekening hield met de overplaatsing van gedetineerden naar ander penitentiaire inrichtingen. En had men te weinig aandacht voor de traceerbaarheid van celbewoners en bovendien voor ademhalingsproblemen van gedetineerden.

### 2.3.5 1951

**Beschrijving** Turkish Airlines-vlucht 1951 (ook bekend als TK 1951) was een passagiersvlucht die op woensdag 25 februari 2009 neerstortte in de buurt van het Nederlandse vliegveld Schiphol. Hierbij kwamen 9 van de 135 inzittenden om het leven.

Het vliegtuig, een Boeing 737-800 van de Turkse maatschappij Turkish Airlines, was op weg van het vliegveld Istanbul Atatürk naar Schiphol, maar stortte om 10:26 uur, kort voor de landing op de Polderbaan, neer op een akker en brak daarbij in drie stukken.

**Datum en Plaats** Op woensdag 25 februari 2009 voltrok zich de ramp met een toestel van turkisch airlines tijdens vlucht 1951.



**Oorzaak** De automatische reactie van het toestel werd getriggerd door een fout gevoelige radio altimeter waardoor de automatische gashendel de energiemotor op actief stelde. Inadequaats handelen van de piloten ondanks een defecte hoogtemeter en onvolledige instructies van de luchtverkeersleiding.

Het officiële onderzoek van de Onderzoeksraad voor Veiligheid wees inadequaats handelen van de piloten als hoofdoorzaak aan voor het ongeluk. Ondanks een defecte hoogtemeter en onvolledige instructies van de luchtverkeersleiding hadden de piloten het ongeluk kunnen voorkomen, aldus de Onderzoeksraad.

### 2.3.6 slmramp

**Beschrijving** Toen het toestel op 07/06/1989 de Anthony Nesty Zanderij naderde, was het daar, anders dan het weerbericht had voorspeld, mistig. Het zicht was evenwel niet zo slecht dat er niet op zicht kon worden geland. Gezagvoerder Will Rogers besloot echter via het Instrument Landing System (ILS) te landen, hoewel dit niet betrouwbaar was en hij voor zo'n landing ook geen toestemming had. De gezagvoerder brak drie landingspogingen af. Bij de vierde poging negeerde de bemanning de automatische waarschuwing (GPWS) dat het toestel te laag vloog. Het toestel raakte op 25 meter hoogte twee bomen. Het rolde om de lengte-as en stortte om 04.27 uur plaatselijke tijd ondersteboven neer.

**Datum en Plaats** De ramp voltrok zich op 07 juni 1989 toen het vliegtuig de Anthony Nesty Zanderij naderde

#### Oorzaak

Uit onderzoek bleek dat de papieren van de bemanning niet in orde waren door nalatigheid in de crew-member screening bij de SLM. Geconcludeerd werd dat de gezagvoerder roekeloos had gehandeld door voor een ILS-landing te kiezen terwijl hij daar geen toestemming voor had, en door onvoldoende op de vlieghoogte te hebben gelet. Het vliegen onder de minimum hoogte leidde tot collision met een boom.

### 2.3.7 Tsjernobyl

**Beschrijving** De mislukte veiligheidscontrole die 26 april 1986 01.24 uur in de Sovjet-Unie leidde tot explosies in een van de reactoren in de kerncentrale. De reactoren hadden geen veiligheidssomhulling en de reactor bevat grote hoeveelheden brandbaar grafiet. Door de explosie en de brand kwamen er radioactieve stoffen vrij. Het gaat helemaal mis in de kernreactor 4. De warmteproductie nam toe met een explosie tot gevolg. 31 mensen kwamen om, waaronder veel mensen dagen later door stralingsziekte.

**Datum en Plaats** De ramp van Tsjernobyl voltrok zich op 26 april 1986 [?].

#### Oorzaak

Technici bij kerncentrale 4 voerden een slecht opgezet/ ontworpen experiment uit. De kracht regulering werd uitgeschakeld evenals veiligheidssystemen. Door een Bedieningsfout in een testprocedure werd het vermogen van de koelinstallaties negatief beïnvloed. Mede door een ontwerpfout in de noodstopprocedure kon in het systeem niet snel genoeg schakelen om remmende invloed uit te oefenen op het toenemende vermogen van de reactorkernen. Met brand en explosies tot gevolg. [?] [?] [?] [?] [?]

Modellen

## 2.4 De Kripke structuur

De kripke structuur is een set van locaties, transities, guards, klokken en data-variabelen. Temporal logic formele taal voor het specificeren en redeneren over hoe de Het gedrag van een systeem verandert in de loop van de tijd breidt de propositiologica uit met modale/temporele operatoren een belangrijk gebruik: representatie van toekomstige systeemeigenschappen gecontroleerd door een modelchecker. Informeel betekenen A en E 'langs alle paden' en 'langs ten minste één pad', respectievelijk; terwijl X, F, G en U verwijst naar 'volgende staat', 'een toekomstige staat', 'hele toekomst' staten," en "Tot." We gebruiken structurele inductie op CTL-formules om te definiëren een tevredenheidsrelatie:  $M, s$  impliceert  $\phi$  dat hangt af van een transitiesysteem  $M = (S, \rightarrow, L)$ , a toestand  $s$  van  $M$ , en een CTL-formule  $\phi$ . Syntaxis van CTL is opgesplitst in state- en padformules specificeer respectievelijk eigenschappen van toestanden/paden een CTL-formule is een toestandsformule State formulae: waarbij een E AP en een padformule is voor een atomaire propositie Path formulae waarbij een toestandsformule (stateformula) is Elke padkwantificator moet worden gevolgd door een temporele kwantificator in de syntactische boom van elke formule

### 2.4.1 CTL

CTL, introduced by Emerson and Clarke [18], is a branching-time temporal logic used as a specification language for finite-state systems. The executions of the system are modeled as linear sequences of system events. Those event sequences are called computation paths in the underlying computation tree modeling the structure of time. CTL formulas are composed of logical operators, path quantifiers, and temporal operators. The path quantifiers are used to state if a property should hold on some computation path (E) or on all computation paths (A) starting from the current state. The temporal operators are used to describe properties of a path through the computation tree. Computatieboomlogica (CTL) is een vertakkingstijdlogica dat omvat zowel de propositionale connectieven als temporele verbindingen AX, EX, AU, EU, AG, EG, AF, en EF. We moeten aantonen dat een real-time programma voldoet aan de eisen opgesteld en gespecificeerd.

Een formele verificatie van de gespecificeerde requirements ofwel modeleigenschappen wordt gerealiseerd door deze te vertalen naar de query-language van de symbolic model-checker Uppaal. [?],[?].

Temporele logica is een al lang gebruikt en goed begrepen concept om softwarespecificaties en computerprogramma's te modelleren door middel van toestandsovergangsemantiek. Een pad is een reeks toestanden. Bij elke toestand kan een toestandsformule worden geëvalueerd. Een vertakkende tijdpadformule strekt zich uit over mogelijke reeksen van opeenvolgende toestanden. States hebben labels. Als de huidige toestand een label  $p$  heeft, dan is de atomaire toestandsformule  $p$  waar in die toestand. Toestandsformules worden opgebouwd uit deze atomaire formules met behulp van de Booleaanse connectieven en de padlogische kwantificatoren, met de beperking dat we een padlogische kwantificator in een toestandsformule niet kunnen ontkennen. (Negatie kan effectief worden toegepast op conjuncties en disjuncties door de negatief-normale vorm te gebruiken).

Het bewijs van de modeleigenschappen kan wiskundig worden opgesteld met behulp van kwantoren, zoals:

## 2.5 Guards en invarianten

Invarianten en guards zijn condities. Een invariant is een conditie die geldt in een state en die altijd waar is wanneer de automaat zich in die state bevindt. Een guard is een conditie die geldt in een transitie.

M.a.w. De transitie kan alleen genomen worden wanneer de guard geldig is.

In feite zijn veiligheidseigenschappen een soort invarianten. Invarianten zijn eigenschappen die worden gegeven door een voorwaarde voor de toestanden en vereisen dat geldt voor alle bereikbare staten. Het begrip 'invariant' kan dus als volgt worden uitgelegd: er moet aan de voorwaarde zijn voldaan door alle begintoestanden en de bevrediging van is onveranderlijk onder alle overgangen in het bereikbare pad van het gegeven transitiesysteem.

Invarianten kunnen worden gezien als toestandseigenschappen en kan worden gecontroleerd door te kijken naar de bereikbare staten.

## 2.6 Deadlock

Deadlock is een bereikbare staat waarin het systeem helemaal geen acties kan uitvoeren – Een Deadlock hangt af van de reeks acties die een bereikbare staat niet kan uitvoeren. Om de impasse te behouden moet A niet alleen overschatten wat P kan doen, maar ook wat P weigert

## 2.7 Zeno gedrag

Zeno gedrag houdt in dat de mogelijkheid dat in een eindige hoeveelheid tijd een oneindig aantal handelingen kan worden verricht.

# 3 Logica

## 3.1 Propositielogica

In de propositielogica onderzoekt men het waarheidsgehalte van samengestelde uitspraken aan de hand van elementaire proposities en logische voegwoorden. Wij noemen de woorden 'en', 'of', 'als . . . dan . . .', 'impliceert' logische voegwoorden. Hoewel het taalkundig geen voegwoorden zijn, noemen wij de ontkenningen 'niet' en 'geen' toch logische voegwoorden.

In de propositielogica worden proposities gemaakt van elementaire proposities en logische voegwoorden. In de symbolische propositielogica, de propositierekening, wordt een propositieformule gemaakt van variabelen (letters), logische operatoren en haakjes. Net zoals een formule in de rekenkunde, heeft het deel van een propositieformule binnen de haakjes een hogere prioriteit dan het deel buiten de haakjes. Bovendien wordt alles wat tussen haakjes staat, beschouwd als een eenheid. Om een propositieformule correct samen te stellen, moeten wij verplicht gebruik maken van de volgende grammaticale regels:

1. Een variabele is een formule;
2. Als  $p$  een formule is, dan is  $\neg p$  een formule;
3. Als  $p$  en  $q$  formules zijn, dan zijn  $(p \wedge q)$ ,  $(p \vee q)$ ,  $(p \rightarrow q)$  en  $(p \leftrightarrow q)$  formules.

In een propositieformule mogen haakjes worden weggelaten mits er geen verwarring ontstaat

Tijdens het logisch beschrijven van logica of bij het maken van formules over formules, is het mogelijk dat wij een paradoxale uitspraak doen. Zo'n uitspraak is vergelijkbaar met de volgende zin: Deze zin is onwaar. De bovenstaande zin beschrijft zichzelf. Is deze zin nu waar of onwaar? Dit is een paradox. Wij moeten de uitspraken over de logica scheiden van de logica zelf. Daarom introduceren wij

metasymbolen. Deze metasymbolen zijn geen onderdeel van de beschreven logica, maar een toevoeging aan de omgangstaal in dit dictaat.

Voor het bepalen van de waarheidswaarde van een formule  $f(p_1; p_2; \dots; p_n)$  moeten alle mogelijke combinaties van waardetoekenningen worden bepaald. Deze combinatie van waardetoekenningen vormen samen de waarheidstabel van deze formule.

### 3.2 Predicatenlogica

Hoewel de volgende redenering in de propositielogica ongeldig is, wordt zij intuïtief toch als geldig beschouwd:

1.  $f_1$  "alle republieken plegen geen overspel"
2.  $f_2$  "sommige overspeligen zijn president"
3.  $y$  "sommige presidenten zijn geen republiek"

De geldigheid van deze redenering is gebaseerd op informatie, waarmee de propositielogica geen rekening mee houdt. Om deze extra informatie bij het redeneren te betrekken, moeten wij de propositielogica uitbreiden met de begrippen eigenschappen, variabelen en kwantoren. De zin "alle mensen zijn sterfelijk" geeft aan dat objecten, die de eigenschap 'menselijk zijn' hebben, blijkbaar ook de eigenschap 'sterfelijk zijn' hebben. uitspraak symbolisch:

1. sterfelijk objecten  $S(x)$
2. menselijke objecten  $M(x)$
3.  $x$  is een priemgetal  $P(x)$

In de uitspraken geven wij de eigenschappen sterfelijk en priemgetal aan met de hoofdletters  $S$  en  $P$ . De variabele  $x$  stelt objecten voor. Een variabele, waarvan de waarde onbekend is, noemen wij vrije variabele. Definitie 4.1 (Predikaat) Een predikaat is een uitspraak met vrije variabelen, die een propositie wordt zodra alle vrije variabelen in dat predikaat gebonden zijn aan een waarde.

### 3.3 Kwantoren

Kwantificator wordt gebruikt om de variabele van predikaten te kwantificeren. Het bevat een formule, een soort verklaring waarvan de waarheidswaarde kan afhangen van de waarden van sommige variabelen. Wanneer we een vaste waarde aan een predikaat toekennen, wordt het een propositie. Op een andere manier kunnen we zeggen dat als we het predikaat kwantificeren, het predikaat een propositie wordt. Kwantificeren is dus een woordsoort dat verwijst naar kwantificeringen als 'alles' of 'sommige'.

Er zijn hoofdzakelijk twee soorten kwantoren: universele kwantoren en existentiële kwantoren. Daarnaast hebben we ook andere soorten kwantoren, zoals geneste kwantoren en kwantoren in standaard Engels gebruik. Kwantificator wordt voornamelijk gebruikt om aan te tonen dat voor hoeveel elementen een beschreven predikaat waar is. Het laat ook zien dat voor alle mogelijke waarden of voor sommige waarde(n) in het universum van het discours het predikaat waar is of niet.

Tot dusverre hebben wij in de voorbeelduitspraken "voor alle  $x$ " en "er is een  $x$ " gekoppeld aan alle voorkomens van  $x$  in een universum. De uitspraak "er is een  $x$  in het universum van  $x$  met de eigenschap  $P$ " wordt weergegeven als:

Soms willen wij zo'n universum beperken. Zo'n beperkt deel van een universum, een domein, is het gebied waaruit de gebonden variabele moet putten. De beschrijving van het domein  $D(x)$  wordt onder de kwantor geplaatst.

Met de universele kwantor  $\forall x$  bedoelen wij alle waarden van  $x$  binnen het opgegeven domein. Als het domein niet wordt opgegeven, dan bedoelen wij een of meer waarden  $x$  uit een eindig universum. Bijvoorbeeld:

Met de existentiële kwantor  $\exists x$  bedoelen wij 'één of meer waarden van  $x$  uit het opgegeven domein. Als het domein niet wordt opgegeven, dan bedoelen wij een of meer waarden  $x$  uit een eindig universum. Bijvoorbeeld:

Een overzicht van meerde temporal operators:

1. F sometime in the Future, The future time operator (F) is used to specify that some property eventually holds at some state in our path
2. G Globally in the future, The global operator (G) is used to specify that some property holds for all states of a particular path.
3. X neXtime, The next time operator (X) is used to specify that some property holds in the second state of a path.
4. U until, The until operator (U) is a binary operator, and is used to specify that the first property holds in all states preceding the one where the second property is satisfied
5. R the release operator, The release operator (R) is also a binary operator, and is used to specify that the second property holds in all states along a path up to and including the first state that satisfies the first property. The first property is not required to be eventually satisfied.

Path quantifiers:

1. E there Exists a path
2. A in All paths

### 3.4 Dualiteiten

Het dual van elke uitspraak in een Booleaanse algebra  $B$  is de uitspraak die wordt verkregen door de bewerkingen uit te wisselen  $+$  en  $,$  en hun identiteitselementen  $0$  en  $1$  in de oorspronkelijke verklaring verwisselen. De dubbele van bijvoorbeeld  $(1 + a) (b + 0) = b$  is  $(0 a) + (b 1) = b$ . Observeer de symmetrie in de axioma's van een Booleaanse algebra  $B$ . Dat wil zeggen, de duale van de reeks axioma's van  $B$  is de hetzelfde als de originele set axioma's. Dienovereenkomstig geldt het belangrijke principe van dualiteit in  $B$ . Namelijk: Stelling 15.1 (Dualiteitsprincipe): De dualiteit van elke stelling in een Booleaanse algebra is ook een stelling. Met andere woorden: als een uitspraak een gevolg is van de axioma's van een Booleaanse algebra, dan is de duale ook een gevolg van deze axioma's, aangezien de dubbele verklaring kan worden bewezen door de duale van elke stap van het bewijs te gebruiken van de oorspronkelijke verklaring.

### 3.5 Operator: AG

AG P: in alle mogelijke paden, is P altijd waar

### 3.6 Operator: EG

EGp - in tenminste een pad geldt p is overal geldig. Een pad is een reeks toestanden. Bij elke toestand kan een toestandsformule worden geëvalueerd. Een vertakkende tijdpadformule strekt zich uit over een enkele reeks van opeenvolgende toestanden. In de huidige reeks heeft elke toestand een label p, en is de atomaire toestandsformule p waar in die toestand behorend tot deze reeks.

### 3.7 Operator: AF

AF p - voor alle paden geldt, p is ergens in de toekomst/ ooit geldig.

Een pad is een reeks toestanden. Bij elke toestand kan een toestandsformule worden geëvalueerd. Deze vertakkende tijdpadformule strekt zich uit over meerdere paden met in de toekomst een geldige opvolgende toestand. Als voor alle paden geldt dat deze vanaf de huidige toestand er in de toekomst een toestand een label p heeft, dan is de atomaire toestandsformule p waar in die toestand.

### 3.8 Operator: EF

EF, er is een pad waar uiteindelijk, vanaf de huidige staat met enkele andere staten p geldig is EF P: in tenminste 1 pas, zal P vroeg of laat geldig zijn

We hebben hier een pad met een reeks toestanden. Waarbij ooit op dit pad een toestand kan toestandsformule worden geëvalueerd welke een label p heeft, en de atomaire toestandsformule p waar is in die toestand.

### 3.9 Operator: AX

AXp - voor alle paden is p in de volgende state geldig

We spreken hier van alle mogelijke paden waarvoor geldt dat in de volgende toestand kan een toestandsformule worden als "waar" wordt geëvalueerd. States hebben labels. Als de volgende toestand een label p heeft, dan is de atomaire toestandsformule p waar in die toestand.

### 3.10 Operator: EX

EX P: in ten minste 1 execution path, is p geldig in de volgende state Een pad is een reeks toestanden. Bij elke toestand kan een toestandsformule worden geëvalueerd. Een vertakkende tijdpadformule strekt zich uit over een mogelijke reeks van opeenvolgende toestanden. Waarbij geldt dat in de huidige reeks er een toestand is met een opvolgende state met een label p heeft, en is de atomaire toestandsformule p waar in die toestand.

### 3.11 Operator: p U q

A[P U Q]: in alle uitvoerpaden is P geldig totdat Q geldig is. Of E[P U Q]: in tenminste 1 pad is P geldig totdat Q geldig is. Een pad is een reeks toestanden. Bij elke toestand kan een toestandsformule worden geëvalueerd. Een vertakkende tijdpadformule strekt zich in geval van A[P U Q]: uit over alle reeksen van opeenvolgende toestanden. Waarvoor geldt dat als de toestand een label p heeft, dan is de atomaire toestandsformule p waar in die toestand tot aan de toestand waarin een toestand het label q heeft.

### 3.12 Operator: $p \ R \ q$

$R$  ("release") is de logische duaal van de  $U$ -operator. De operator vereist dat het tweede argument stand moet houden tot en met de eerste staat waar het eerste argument geldt. Het eerste argument is dat niet nodig om uiteindelijk waar te worden.

Een pad is een reeks toestanden. Bij elke toestand kan een toestandsformule worden geëvalueerd. Een vertakkende tijdpadformule strekt zich uit over een mogelijke reeks van opeenvolgende toestanden. States hebben labels. Als de huidige toestand een label  $q$  heeft, dan is de atomaire toestandsformule  $q$  waar in die toestand tot aan de toestand waarin  $p$  geldig is. Met de voorwaarde dat de eigenschap  $p$  niet gelabeld hoeft te worden aan een opvolgende state. De release-operator ( $R$ ) is ook een binaire operator en betekent dat de tweede argument geldig moet zijn in alle staten tot aan  $q$  inclusief de eerste staat die voldoet aan de eerste eigenschap.

### 3.13 Fairness

Deze eigenschappen beschrijven de vereiste dat een proces vooruitgang boekt in de richting van een specifiek doel, waarvan de verwezenlijking afhangt van de fairness van het systeem. Als een proces nooit wordt uitgevoerd, kan het zijn doel meestal niet bereiken. Daarom worden deze eigenschappen vaak alleen langs eerlijke paden geëvalueerd, d.w.z. paden die voldoen aan een van de volgende drie vereisten voor fairnessconstraints: Absolute Fairness, Impartiality: every process should be executed infinitely often: Strong Fairness: every process that is infinitely often enabled should be executed infinitely often in a state where it is enabled: Weak Fairness: every process that is almost always enabled should be executed infinitely often:

### 3.14 Safety

Safety properties are often characterized as "nothing bad should happen". The mutual exclusion property—always at most one process is in its critical section—is a typical safety property. It states that the bad thing (having two or more processes in their critical section simultaneously) never occurs. Another typical safety property is deadlock freedom. For the dining philosophers (see Example 3.2, page 90), for example, such deadlock could be characterized as the situation in which all philosophers are waiting to pick up the second chopstick. This bad (i.e., unwanted) situation should never occur.

### 3.15 liveness properties

Veiligheidseigenschappen geven aan wat er wel of niet mag voorkomen, maar eis niet dat er ooit iets gebeurt. liveness geeft aan wat er moet gebeuren. De eenvoudigste vorm van een liveness-eigenschap garandeert dat er uiteindelijk iets goeds gebeurt. Met andere woorden: er bestaat een tijdstip waarop het systeem zich in de