Slimme Semantiek voor Onderhoud in Foutenbomen

Bart van den Pol
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
B.vandenpol@student.utwente.nl

ABSTRACT

Er zijn systemen actief die niet mogen falen. Het falen van een systeem zoals de beveiliging van een kernreactor zou desastreuze gevolgen kunnen hebben. Om te controleren hoe veilig een systeem is kan een Dynamische FoutenBoom (DFB) geconstrueerd worden die het systeem weergeeft. Een methode om een DFB te analyseren is door de DFB te converteren naar een Input/Output-Interactive Markov Chain (I/O-IMC).

De conversie van een DFB naar een I/O-IMC is reeds gemaakt, maar de I/O-IMC bevat meer toestanden dan nodig om het systeem weer te geven en de analyse uit te voeren. Deze toestanden worden overbodige toestanden genoemd. Door de overbodige toestanden krijgt de I/O-IMC een state-space explosie wat leidt tot meer berekenstappen om het systeem te controleren. Om het controleren van systemen te versnellen is een betere conversie van DFB naar I/O-IMC gewenst. In [7] is voor een gedeelte van de DFB dit geïmplementeerd, maar nog niet voor de inspectie- en reparatiemodule.

Voorbeeld 1: *Wanneer een onderdeel van een DFB altijd actief is, dan kan de inactieve toestand in de I/O-IMC verwijderd worden.*

Door de inactieve toestanden te verwijderen ontstaat al een grote tijdwinst. Wanneer dit ook nog toegepast kan worden op het onderhoud van DFB elementen, kunnen systemen sneller geanalyseerd worden.

Keywords

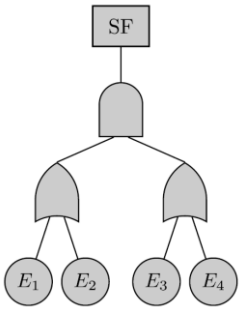
Maintainable Fault Trees, DFTCalc, DFT, smart semantics, I/O IMC, risk analysis, context dependent.

1. INTRODUCTIE

Het is bij sommige systemen essentieel om te weten hoe groot de kans is dat het systeem faalt. Voorbeelden van dergelijke systemen zijn de back-up stroomvoorziening van een ziekenhuis, een wissel bij het spoor en het stoppen van kernsplitsing in een kerncentrale.

Een manier om de risico's van een systeem te controleren is door van het systeem een Foutenboom (FB) te maken. Een FB geeft alle onderdelen van het systeem weer, gemodelleerd in kansen dat het betreffende onderdeel kapot gaat en dat weer verbonden met logische poorten. In een FB staat overzichtelijk welke onderdelen moeten falen voordat het systeem faalt.

24thTwente Student Conference on IT, January 22st, 2016, Enschede, The Netherlands.



Figuur 1: Een FB

Voorbeeld 2: *figuur 1 geeft een FB weer. De ronde elementen E1 tot en met E4 geven de onderdelen in de FB weer. Wanneer onderdeel E1 of E2 faalt, dan faalt ook de bovenliggende OR poort. Hetzelfde voor E3 en E4. Wanneer beide OR poorten falen, faalt de AND poort en daarmee het systeem.*

Een FB kan niet op alle systemen goed worden toegepast, omdat de elementen die in een FB gebruikt worden beperkt zijn. Daarom is de FB uitgebreid met dynamische poorten. De FB wordt dan een Dynamische FoutenBoom genoemd (DFB). In een DFB kan bijvoorbeeld de volgorde van de falende elementen invloed hebben. Dit is van belang wanneer de switch naar het back-up systeem faalt, als het back-up systeem al ingeschakeld is. Wanneer de switch faalt als het back-up systeem niet aan is ontstaat er een kritieke situatie, is het na het activeren dan is het (voorlopig) niet van belang.

DFB's kunnen onder andere geanalyseerd worden door de onderdelen van een DFB te converteren naar een I/O-IMC. Vervolgens kunnen de I/O-IMC's samengevoegd worden tot één I/O-IMC die geanalyseerd kan worden. Een programma dat DFB converteert naar I/O-IMC's is DFTCalc. DFTCalc converteert alle componenten van een DFB naar verschillende I/O-IMC's met behulp van module templates. De conversie van DFB naar I/O-IMC geeft nu een state-space explosie. Om de conversie te verkleinen kan er gekeken worden naar de context waarin de elementen van de DFB zich bevinden. Wanneer een element van een DFB niet gedeactiveerd kan worden, kan deze toestand uit de I/O-IMC.

In [7] is al een begin gemaakt met het verbeteren van de conversie van DFB naar I/O-IMC's. Daarin is op verschillende onderdelen van de DFB slimme semantiek toegepast. Dit houdt in dat onderscheid tussen altijd-actieve en soms-actieve onderdelen gemaakt is. Wanneer een onderdeel een SPARE poort boven zich heeft dan hoeft het onderdeel niet altijd-actief te zijn.

Slimme semantiek is echter nog niet toegepast op het onderhoud gedeelte van de DFB. Dat wil zeggen de reparaties, vernieuwingen en reserveonderdelen. Om de conversie daarop toe te passen moet er gekeken worden naar de verschillende toestanden waarin de elementen van de DFB zich bevinden. Het doel van dit onderzoek is om met behulp van slimme semantiek overbodige toestanden uit de I/O-IMC te halen door bij de

conversie van het onderhoudt gedeelte van DFB naar I/O-IMC de kleinst mogelijke templates te gebruiken die het element correct weergeven.

2. DOEL EN ONDERZOEKSVRAGEN

Het doel van dit onderzoek is om het converteren van het onderhoudt gedeelte van een DFB naar I/O-IMC's te verbeteren. Het converteren is al geïmplementeerd, maar er wordt nog niet gekeken in welke context de elementen van de DFB zich bevinden. In dit onderzoek wordt er ook voor deze conversie slimme semantiek toegevoegd. Wanneer de overbodige toestanden uit de I/O-IMC zijn gehaald kan de I/O-IMC sneller geanalyseerd worden en kunnen dus grotere DFB geanalyseerd worden.

De hoofdvraag is als volgt:

'Hoe kan de state-space tijdens de analyse van een onderhoud DFB verkleind worden?'

Om het doel te bereiken en de hoofdvraag te beantwoorden zijn de volgende vragen opgesteld.

1. Welke toestanden van de I/O-IMC kunnen weggelaten worden, terwijl de weergave van het systeem dezelfde functionaliteit behoudt?
2. Welke module templates moeten in welke situatie worden toegepast zodat je altijd dezelfde functionaliteit van het systeem behoudt?
3. Zijn de verbeterde module templates correct?
4. Levert de verbeterde module template een snellere analyse van een DFB op?

3. ACHTERGROND

3.1 Foutenboom

Zoals beschreven staat in [5] is een FB een gerichte acyclische graaf bestaande uit twee elementen, namelijk onderdelen (leaves) en poorten (nodes). Onderdelen worden ook wel basis elementen (BE) genoemd en geven de onderdelen van het daadwerkelijke systeem weer. Onderdelen kunnen falen, wat via de poorten doorgegeven kan worden. Afhankelijk van de input kan een poort dan falen. Een FB heeft de volgende poorten:

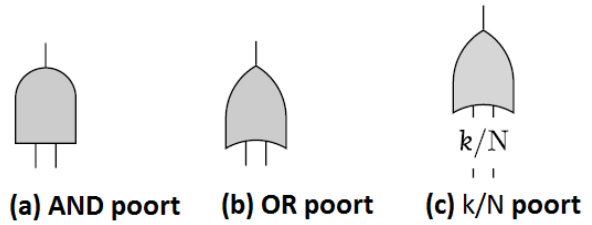
AND faalt als alle inkomende signalen input geven.

OR faalt als één van de inkomende signalen input geeft.

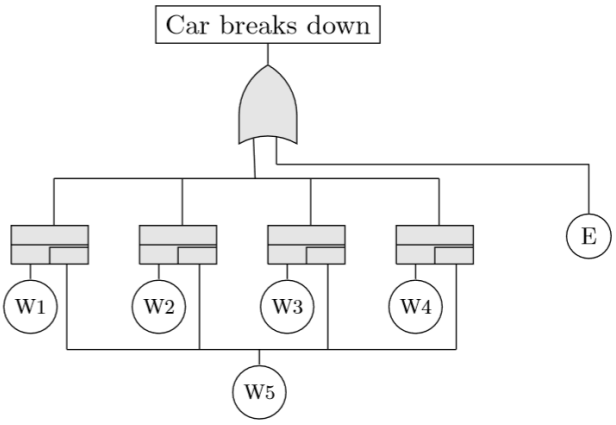
k/N faalt als minstens k van de N signalen input geven.

INHIBIT faalt als al zijn inkomende signalen input geven en daarnaast ook de conditie gebeurtenis. Aangezien dit hetzelfde is als de AND gate wordt deze poort niet verder behandeld in deze paper.

Alle onderdelen die in een FB zitten zijn ofwel actief of gefaald.



Figuur 2: FB poorten



Figuur 3: DFB van een auto

3.2 Dynamische FoutenBoom

DFB is een uitbreiding van FB met dynamische poorten. Deze poorten zorgen o.a. ervoor dat de volgorde van de falende input van belang kan zijn en voegen daarnaast reserve onderdelen toe aan de FB.

Voorbeeld 3: *Een DFB is gegeven in figuur 3. Wanneer één van de wielen (W1-W4) faalt, kan daar het reservewiel W5 gebruikt worden. Als er daarna weer een wiel faalt, of de motor (E) faalt dan is de auto kapot.*

Een DFB voegt de volgende poorten toe:

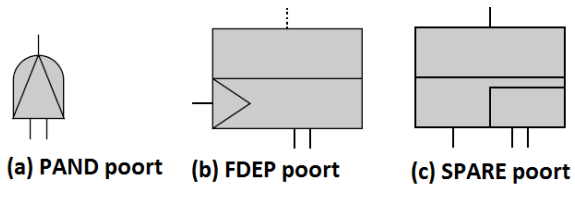
PAND Faalt als de inkomende signalen van links naar rechts falen. Wanneer de inkomende signalen niet in volgorde falen, faalt de PAND niet.

SPARE Wanneer een BE die aan de SPARE poort verbonden is faalt, dan claimt de SPARE het reserve onderdeel als dat beschikbaar is. Als het reserve onderdeel niet beschikbaar is en een BE faalt, dan faalt de SPARE.

FDEP Wanneer deze poort een signaal krijgt van zijn conditie dan falen al zijn uitgangen.

Naast de toevoeging van deze poorten, kunnen bij DFB onderdelen inactief zijn. Een voorbeeld hiervan zijn reserveonderdelen, zoals de vijfde band. Inactief staat dus voor niet in gebruik. Wanneer onderdelen inactief zijn kunnen deze onderdelen een verschillend faaltempo hebben dan wanneer ze actief zijn. Dit kan verschillen van niet kunnen falen tot dezelfde kans om te falen als de actieve onderdelen.

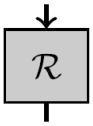
Alle onderdelen zijn in een DFB actief, tenzij de onderdelen onder een SPARE zitten. Als de SPARE repareerbaar is dan zijn alle onderdelen onder de SPARE niet repareerbaar. Als de SPARE niet repareerbaar is, dan is het eerste element dat aan de SPARE zit actief en de andere element inactief.



Figuur 4: DFB poorten

3.3 Onderhoud van Dynamische FoutenBoom

De DFB is nog verder uitgebreid in [1] met een onderhoudt gedeelte. DFB kan onderhouden worden door middel van inspecties, reparaties en vernieuwingen. De poorten worden daarom uitgebreid met een reparatie module welke kijkt of een BE aan reparatie toe is. Wanneer een onderdeel dan faalt wordt het signaal doorgegeven, als het signaal bij de reparatie module komt, dan gaat de reparatie module repareren. De reparatie module houdt de volgorde bij waarin de reparaties worden uitgevoerd. Mocht een BE gerepareerd zijn dan komt het voor dat de bovenliggende poorten niet meer falen.



Figuur 5: REPAIR module

Een BE is repareerbaar als er een reparatie module aan zit, voor de poorten geldt dat ze repareerbaar zijn wanneer er onderliggende BE's zijn zie repareerbaar zijn. Niet alle poorten zijn repareerbaar, van de DFB poorten is het gedrag bij reparatie nog niet vastgelegd.

3.4 Input/output-Interactive Markov Chains

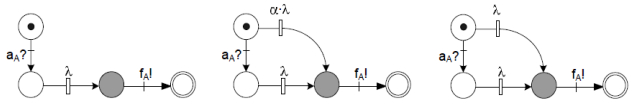
DFB analyseren met behulp van I/O-IMC's is geïntroduceerd in Boudali et al. [2] I/O-IMC's bestaan uit verschillende toestanden, in- en output acties, interne acties en Markovian acties [3]. Een input actie wordt omschreven met een "?" en moet gesynchroniseerd worden met een output actie, aangeven met een "!". Interne acties gebeuren onmiddellijk en hebben geen synchronisatie met andere I/O-IMC's. Markovian acties die aangegeven worden met μ en λ geven een systeem vertraging weer.

Een van de eigenschappen van een I/O-IMC is dat I/O-IMC's kunnen worden samengevoegd met andere I/O-IMC's. Wanneer dezelfde acties erin voorkomen worden deze met elkaar gesynchroniseerd, wanneer die er niet zijn kunnen de onderdelen direct worden samengevoegd.

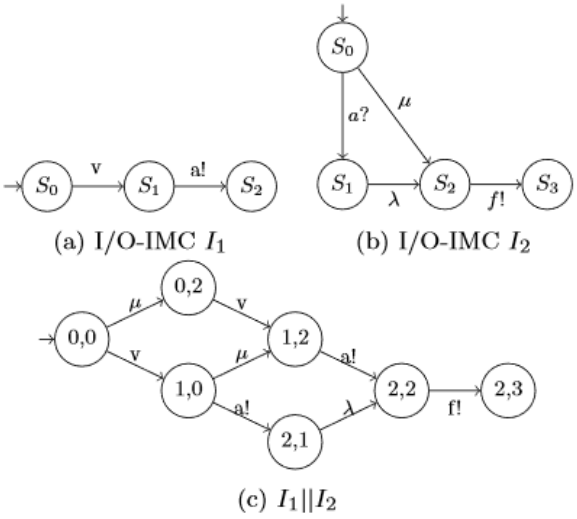
Voorbeeld 4: *Wanneer de I/O-IMC's I_1 en I_2 van figuur 6 worden samengevoegd heeft dit voornamelijk invloed bij de actie a. De actie a kan alleen plaatsvinden wanneer ervoor een v heeft plaatsgevonden. Los daarvan kunnen de acties μ en v altijd plaatsvinden voor a. Alleen als er nog geen μ heeft plaatsgevonden en wel a kan λ plaatsvinden. Dit blijft echter allemaal gewoon hetzelfde als in de normale I/O-IMC.*

Een andere eigenschap van I/O-IMC input enabled. Wanneer er een actie a! uitgevoerd wordt terwijl dit niet mogelijk is, dan wordt a! geabsorbeerd. Alle toestanden van de I/O-IMC zijn absorberend.

Elke I/O-IMC heeft een begin toestand, tussenliggende toestanden en een eindtoestand. Wanneer een toestand grijs is, dan geeft het aan dat de I/O-IMC gefaald heeft. De toestand met de stip erin geeft de begintoestand weer en de dubbele cirkel de eindtoestand.



Figuur 7: I/O-IMC's van een koud, warm en actief onderdeel



Figuur 6: I/O-IMC samenvoegen

3.5 Van DFB naar I/O-IMC

DFTCalc bouwt I/O-IMC's om een DFB te analyseren. De I/O-IMC's worden op de volgende manier gebouwd:

1. Elk onderdeel van de DFB wordt getransformeerd in een I/O-IMC met behulp van module templates.
2. Twee verschillende I/O-IMC's worden samengevoegd tot één I/O-IMC.
3. De overbodige signalen worden verborgen.
4. De gecombineerde I/O-IMC wordt geminimaliseerd.
5. Wanneer er meer dan één I/O-IMC is, ga terug naar stap 2.

Bij stap 1 worden alle onderdelen in de DFB vertaald naar een I/O-IMC.

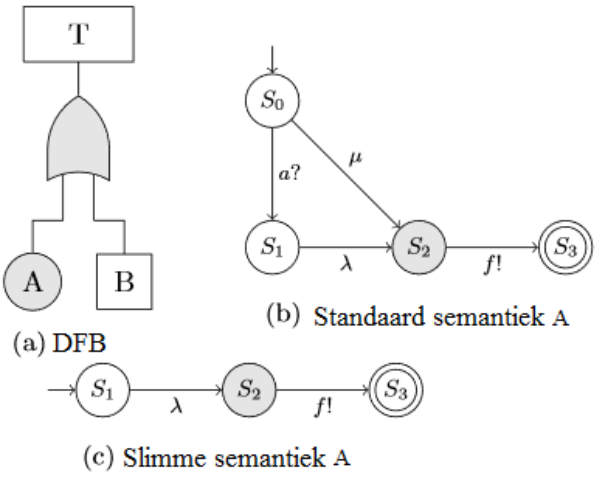
3.6 Overbodige toestanden in I/O-IMC's

Definitie 1. *Een I/O-IMC bestaat uit < s_x, →^{I}, →^{o} >, waarvoor geldt:*

- *s_x is een toestand in de I/O-IMC*
- *→^{I} is een inkomende actie bij een toestand*
- *→^{o} is een uitgaande actie van een toestand.*

Overbodige toestanden in I/O-IMCs zijn toestanden waarvan de inkomende actie niet uitgevoerd wordt en daardoor de toestand niet bereikt wordt. Daarnaast kunnen alle toestanden die uit de I/O-IMC worden gehaald zonder de resultaten van de analyse van de I/O-IMC aan te passen als overbodige toestanden gekenmerkt worden.

Voorbeeld 5: *In figuur 8.a staat een DFB met een subsysteem weergegeven door de vierkant met een B erin. BE a is in dit systeem altijd actief. Dat betekent dat wanneer het systeem geanalyseerd wordt, bij 8.b de stap van S0 →a? S1 direct gemaakt wordt. Wanneer slimme semantiek toegepast wordt, dan krijg je 8.c waarin deze stap al gedaan is en daarom niet meer geanalyseerd hoeft te worden.*



Figuur 8: Slimme semantiek

Aan de hand van de context waarin de element van een DFB zich bevindt, kan bepaalt worden welke module template gekozen moet worden, om het element correct en minimaal weer te geven. Wanneer de conversie van een DFB naar I/O-IMC plaatsvindt heb je vier verschillende situaties die van toepassing op een BE kunnen zijn.

1. De BE kan niet worden gerepareerd en is inactief.
2. De BE kan niet worden gerepareerd en altijd actief.
3. De BE kan worden gerepareerd en is inactief.
4. De BE kan worden gerepareerd en is altijd actief.

Voor elke BE moet gekeken worden welke situatie van toepassing is en daarna de module template bepaald worden. De verschillende situaties voor BE's zijn weergegeven in figuur 9. De inspecties kunnen nog extra toestanden toevoegen, maar geen andere module templates. Aan de hand van de inspecties worden automatisch toestanden toegevoegd of weggehaald.

Daarnaast heb je nog verschillende soorten poorten. Voor de poorten heb je dezelfde vier situaties. Om te weten welke situatie van toepassing is op de poort moet gekeken naar alle kinderen van de poort. Wanneer er minstens één kind is dat gerepareerd kan worden dan is situatie 3 of 4 van toepassing. Mochten er geen BE's onder zitten die kunnen worden gerepareerd dan is situatie 1 of 2 van toepassing. Actief of inactief is afhankelijk van bovenliggende SPARE poorten.

Voorbeeld 6: *Wanneer in figuur 8.a BE a faalt, dan faalt de OF poort ook. Wanneer a echter gerepareerd wordt (wat in deze situatie niet mogelijk is) zou dit tot gevolg hebben dat de OF poort gerepareerd wordt en niet meer faalt.*

4. GERELATEERD WERK

Een handboek over FB en DFB kan gevonden worden bij [8], geschreven in 2002. In 2015 is er door [5] nog uitgebreid gekeken naar papers over FB en daar een samenvatting van gemaakt.

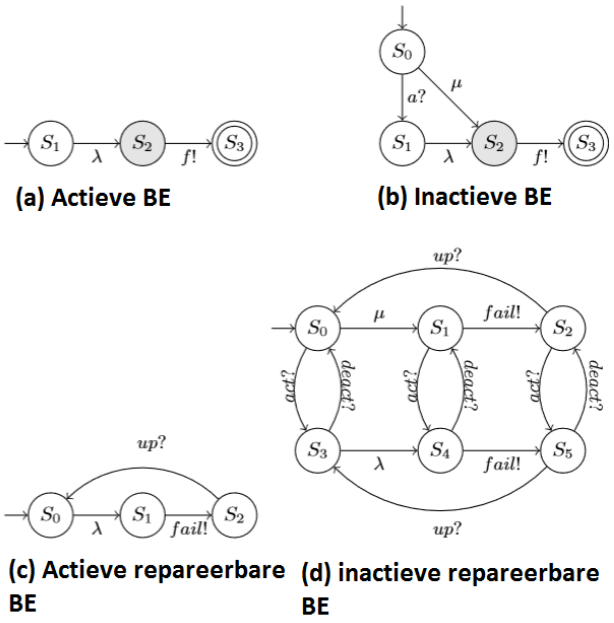
In [7] is met behulp van slimme semantiek verbeterde module templates gemaakt voor de basis onderdelen van een DFB. Er zijn voor alle poorten van een normale DFB templates gemaakt die de poorten actief weergeven. Echter, het onderhoud gedeelte is niet meegenomen bij deze verbetering.

Een andere techniek die gebruikt wordt om DFB sneller te kunnen analyseren is het herschrijven van de DFB. In [6] zijn een aantal regels opgesteld waarmee de DFB herschreven kan worden tot een kleinere DFB met dezelfde functionaliteit.

5. AANPAK

Allereerst moeten er beschikbare module templates van alle poorten en BE's voor elke context. Wanneer de module templates niet beschikbaar zijn, worden deze templates aangemaakt.

Nadat de BE en poorten zijn geïmplementeerd kan er gekeken worden naar verbeteringen, zodat aan de hand van de context waarin de elementen van de DFB zich bevinden kleinere module templates gebruikt worden.



Figuur 9: Mogelijke BE's

Om te kunnen bewijzen dat de verbeterde module templates daadwerkelijk werken zou de uitkomst van het oude geconverteerde I/O-IMC's vergeleken moeten worden met de "verbeterde" I/O-IMC's. Wanneer de uitkomst hetzelfde is voor een aantal kenmerkende geconverteerde DFB dan zijn de verbeterde module templates correct.

Vervolgens kan gekeken worden hoeveel toestanden vermeden worden door deze verbetering en de gevolgen daarvan op de hele analyse.

5.1 Casussen

Bij dit onderzoek wordt gebruik gemaakt van vier verschillende casussen, waarvan de casus fuses met inspectie al eerder gebruikt is in [4]. CAS_rep is een aangepaste versie van CAS dat in [7] ook al gebruikt is. Aan deze casus zijn nu repareerbare poorten toegevoegd om te testen. Daarnaast zijn twee simpele casussen toegevoegd met reparatie- en inspectiemodule.

5.2 Implementatie

Voor de implementatie van het onderhoudt gedeelte is DFTCalc aangepast. Er is een nieuwe module template gemaakt voor actieve repareerbare VOTING poort en daarnaast verdere aanpassing om het smart semantics uit te breiden met een onderhoudt gedeelte. Daarnaast is er een aanpassing gemaakt zodat nu het eerste element van een SPARE actief wordt als de SPARE niet repareerbaar is.

Tabel 1: Resultatentabel

Casus	Tool	Toestanden	Transities	Toestanden max	Transities max	Bereken tijd (s)	P(faal)
CAS reparatie T=1	Origineel	51	247	156	658	6.172167	5.049678306e-09
	Slimme semantiek	50	196	84	258	5.935644	5.049678306e-09
Reparatie T = 1	Origineel	15	45	28	66	15.069369	1.695908231e-18
	Slimme semantiek	14	30	20	38	13.551093	1.695908231e-18
Fuses inspectie T=1	Origineel	2622	17632	2992	19586	8.367207	8.585808039e-07
	Slimme semantiek	2621	15010	3705	17946	7.916114	8.585808039e-07
inspectie t=50000	Origineel	11	24	27	63	27.421411	0.5512081919
	Slimme semantiek	10	13	24	34	20.938373	0.5512081919

6. ANALYSE VAN RESULTATEN EN DISCUSSIE

Bij elke casus geeft de nieuwe implementatie een reductie van de berekentijd. De uitkomst van de faalkans blijft constant wat aangeeft dat de implementatie van het onderhoud gedeelte correct gedaan is.

In de resultaten staat een grote reductie van het aantal fases en transities wanneer met de reparatie module gewerkt wordt. Dit is echter niet terug te zien in de berekentijd die maar lichtelijk afneemt. Waarom dit verband ontbreekt is iets voor verder onderzoek.

Bij de inspectie resultaten geeft de nieuwe implementatie een vreemde uitkomst bij fuses. Het aantal toestanden is toegenomen bij smart semantics ten opzichte van de originele implementatie en de transities zijn afgenomen. Daarnaast neemt zowel bij de simpele variant als de uitgebreide fuses variant de berekentijd af. Bij de simpele variant is dit voornamelijk een groot verschil, wat waarschijnlijk ligt aan de grotere hoeveelheid iteratiestappen.

7. CONCLUSIE

Door de uitbreiding van het onderhoudt gedeelte is het nu ook mogelijk om een uitgebreide DFB sneller te analyseren. Het algoritme geeft dezelfde resultaten terug als de originele analyse en bewijst daarmee dat het correct werkt. Ook de nieuwe module template VOTING actief repareerbaar blijkt goed te werken, zoals te zien is bij CAS.

7.1 Samenvatting

In dit onderzoek is de analyse van een DFB met behulp van DFTCalc verder uitgebreid. De analyse van een uitgebreide DFB met onderhoudt gedeelte was al mogelijk, alleen met overbodige toestanden wat voor een state-space explosie zorgde. In dit onderzoek is een eerder onderzoek naar smart semantics voor DFB verder uitgebreid voor de repareerbare en inspectie poorten. Daardoor kan nu een DFB met onderhoudt gedeelte sneller geanalyseerd worden.

7.2 Reflectie

Door dit onderzoek is het nu ook mogelijke om uitgebreide DFB's met een onderhoudt gedeelte sneller te analyseren met DFTCalc. Daardoor kunnen systemen sneller geanalyseerd worden, wat tot betere regulering kan leiden en veiligere systemen. De methode is vergeleken met de oude implementatie en de uitkomsten daarvan en daarmee is bewezen dat de nieuwe implementatie correct is. Verdere uitbreiding van DFTCalc met andere systemen behoort ook nog tot de mogelijkheden.

7.3 Verder onderzoek

Er kan nog verder onderzocht worden naar de mogelijkheden van de poorten. Wanneer gedrag van poorten beter vast ligt kan gekeken wat er gebeurt als poorten zoals een FDEP, SPARE en PAND gerepareerd worden.

Daarnaast kan het herschrijven van een DFB zoals beschreven in [6] toegepast worden om een DFB nog kleiner te maken zodat snellere analyse mogelijk is. Verder onderzoek kan ook nog gedaan worden naar het multi threaded maken van DFTCalc wat de analyse sneller zou maken.

8. REFERENTIES

[1] Bobbio et al 'Parametric fault trees with dynamic gates and repair boxes'(RAMS 2004)

[2] H. Boudali, P. Crouzen, and M. Stoelinga. A compositional semantics for dynamic fault trees in terms of interactive markov chains. In K. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, Automated Technology for Verification and Analysis, volume 4762 of Lecture Notes in Computer Science, pages 441-456. Springer Berlin Heidelberg, 2007.

[3] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on, pages 708-717, June 2007.

[4] Dennis Guck, Jip Spel and Mariëlle Stoelinga. DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper)

[5] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15U16(0):29 -- 62, 2015.

[6] Sebastian Junges, Dennis Guck, Joost-Pieter Katoen, Arend Rensink, Mariëlle Stoelinga: Fault Trees on a Diet -- Automated Reduction by Graph Rewriting -. SETTA 2015: 3-18

[7] Spel, J. 2015. Smart Semantics for Fault Trees. http://fmt.ewi.utwente.nl/files/sprojects/280.pdf

[8] W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, JosephMinarick III, and J. Railsback. Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance NASA Headquarters Washington, DC 20546, August 2002.

CAS_REP

toplevel "SYSTEM";
"SYSTEM" or "FDEP" "CPU" "MOTOR" "PUMPS";
"FDEP" fdep "TRIGGER" "P" "B";
"TRIGGER" or "CS" "SS";
"CPU" wsp "P" "B";
"MOTOR" or "SWITCH" "MOTORS";
"SWITCH" pand "MS" "MA";
"MOTORS" csp "MA" "MB";
"PUMPS" and "PUMP1" "PUMP2";
"PUMP1" and "PA" "PS";
"PUMP2" and "PB" "PS";
"RU" ru "P" "B";
"RU1" ru "PA" "PB" "PS";
"PA" lambda=5.0e-5 repair=1 dorm=0;
"PB" lambda=5.0e-5 repair=1 dorm=0;
"PS" lambda=5.0e-5 phases=2 repair=1 dorm=0;
"CS" lambda=2.0e-5 dorm=0;
"SS" lambda=2.0e-5 dorm=0;
"MS" lambda=1.0e-6 dorm=0;
"MA" lambda=1.0e-4 dorm=0;
"MB" lambda=1.0e-4 dorm=0;
"P" lambda=5.0e-5 phases=2 repair=3 dorm=0.0000000000;
"B" lambda=5.0e-5 repair=3 dorm=0.0000000000;

Reparatie

toplevel "PUMPS";
"PUMPS" and "PUMP1" "PUMP2";
"PUMP1" and "PA" "PC";
"PUMP2" and "PB" "PD";
"RU" ru "PA" "PC";
"PA" lambda=5.0e-5 repair=3 dorm=0.0000000000;
"PC" lambda=5.0e-5 repair=3 dorm=0.0000000000;
"PB" lambda=5.0e-5 dorm=0.0000000000;
"PD" lambda=5.0e-5 dorm=0.0000000000;
"PE" lambda=5.0e-5 dorm=0.0000000000;

FUSES_insp_4

toplevel "FA-RK-BB-OR";

"FA-RK-BB-OR" or "RK-BB-DUBBEL" "HS-RK-BB-DUBBEL" "FA-RK-BB";

"FA-RK-BB" and "RK-BB" "HS-RK-BB";

"RK-BB" or "RK-BB1" "RK-BB2" "RK-BB3" "RK-BB4";

"HS-RK-BB" or "HS-RK-BB1" "HS-RK-BB2" "HS-RK-BB3" "HS-RK-BB4";

"RK-BB-DUBBEL" 2of4 "RK-BB-DUBBEL1" "RK-BB-DUBBEL2" "RK-BB-DUBBEL3" "RK-BB-DUBBEL4";

"HS-RK-BB-DUBBEL" 2of4 "HS-RK-BB-DUBBEL1" "HS-RK-BB-DUBBEL2" "HS-RK-BB-DUBBEL3" "HS-RK-BB-DUBBEL4";

"Insp1" 2insp4 "RK-BB1" "RK-BB2" "RK-BB3" "RK-BB4";

"Insp2" 2insp4 "HS-RK-BB1" "HS-RK-BB2" "HS-RK-BB3" "HS-RK-BB4";

"Insp4" 2insp4 "RK-BB-DUBBEL1" "RK-BB-DUBBEL2" "RK-BB-DUBBEL3" "RK-BB-DUBBEL4";

"Insp5" 2insp4 "HS-RK-BB-DUBBEL1" "HS-RK-BB-DUBBEL2" "HS-RK-BB-DUBBEL3" "HS-RK-BB-DUBBEL4";

"RK-BB1" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"RK-BB2" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"RK-BB3" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"RK-BB4" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB1" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB2" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB3" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB4" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

"RK-BB-DUBBEL1" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"RK-BB-DUBBEL2" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"RK-BB-DUBBEL3" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"RK-BB-DUBBEL4" lambda=0.010000000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB-DUBBEL1" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB-DUBBEL2" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB-DUBBEL3" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

"HS-RK-BB-DUBBEL4" lambda=0.025300000 phases=2 interval=1 dorm=0.0000000000;

INSPECTIE

toplevel "PUMPS";

"PUMPS" and "PUMP1" "PUMP2";

"PUMP1" and "PA" "PC";

"PUMP2" and "PB" "PD";

"RU" 2insp4 "PA" "PC";

"PA" lambda=5.0e-5 phases=2 interval=3 dorm=0.0000000000;

"PC" lambda=5.0e-5 interval=3 dorm=0.0000000000;

"PB" lambda=5.0e-5 dorm=0.0000000000;

"PD" lambda=5.0e-5 dorm=0.0000000000;

"PE" lambda=5.0e-5 dorm=0.0000000000;