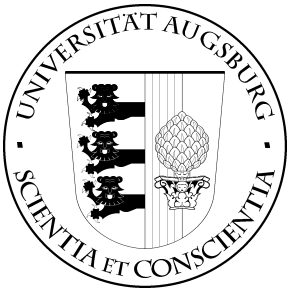
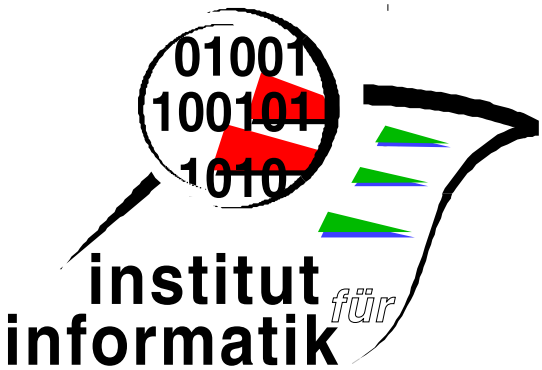
Universität Augsburg



Formale Methoden und Sicherheitsanalyse

A. Thums, F. Ortmeier

Report 2002-15 Dezember 2002



Institut für Informatik
D-86135 Augsburg

Formale Methoden und Sicherheitsanalyse

A. Thums and F. Ortmeier*
Lst. Softwaretechnik und Programmiersprachen,
Universität Augsburg, 86135 Augsburg
email: {thums, ortmeier}@informatik.uni-augsburg.de

3. Dezember 2002

Zusammenfassung

Sicherheitsanalysetechniken wurden in den Ingenieurwissenschaften schon in den 60iger Jahren für technische Systeme entwickelt. Inzwischen sind aber nicht nur Hardware-Komponenten für die Sicherheit technischer Systeme verantwortlich, sondern in zunehmenden Maße wird die steuernde Software zu einem sicherheitskritischen Faktor. Formale Methoden wurden für den Nachweis der Korrektheit von Softwarekomponenten entwickelt.

Es wird eine durchgängige Methode für die sicherheitstechnische Analyse softwarebasierter Systeme gezeigt, die durch Integration von Sicherheitsanalysetechniken und formalen Methoden entsteht. Die resultierende formale Sicherheitsanalyse erlaubt zum Einen die Korrektheits- und Vollständigkeitsprüfung der Sicherheitsanalyse und zum Anderen die Beurteilung der Systemsicherheit bei Komponentenausfällen. Damit erhalten wir Systeme, für die sowohl korrektes Funktionieren als auch Sicherheit bei Komponentenausfällen garantiert ist.

1 Überblick

Die formale Sicherheitsanalyse ist eine Methode zur Entwicklung sicherer Spezifikationen für softwarebasierte Ingenieuranwendungen. Der Begriff 'sicher' bezieht sich hierbei sowohl auf das korrekte Funktionieren eines Systems als auch auf die Robustheit und Fehlertoleranz bezüglich Komponentenausfälle. Dieses Ziel soll durch eine Integration formaler Spezifikation und Verifikation mit Techniken der Sicherheitsanalyse erreicht werden. Die formalen Methoden werden zur präzisen Beschreibung des Systemmodells sowie zur Formulierung und zum Nachweis von Sicherheitseigenschaften verwendet. Mit den Techniken der Sicherheitsanalyse (FTA und FMEA) werden Schwachstellen, Designfehler sowie Sicherheitsanforderungen ermittelt. Dadurch ergibt sich der folgende, wechselseitige Zugewinn:

1. Formale Methoden profitieren von der Sicherheitsanalyse durch die Systematisierung der Suche nach Sicherheitsanforderungen, durch das Herunterbrechen von Anforderungen auf (Software-) Komponenten und durch die Möglichkeit einer quantitativen Beurteilung der Sicherheit.

*Diese Arbeit wurde teilweise im Rahmen des DFG Schwerpunktes „Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen“ erstellt

2. Die Sicherheitsanalyse profitiert von den formalen Methoden durch die exakte Festlegung des Systemverhaltens und die Möglichkeit, Korrektheit und Vollständigkeit der Analyseergebnisse zu garantieren.

3. Durch die Integration entstehen Systeme, die den beiden komplementären Sicherheitsbegriffen standhalten: Korrektes Funktionieren und möglichst ausfalltolerante Gestaltung.

In (Schellhorn *et al.*, 2002) wurde dazu die Sicherheitsanalysetechnik FTA (*fault tree analysis*) formal fundiert. Es wurde eine Semantik in *Intervaltemporallogik (ITL)* (Moszkowski, 1985) entwickelt, die Ideen bisheriger Arbeiten aufgreift und deren Mängel beseitigt. Ein *Minimal-Cut-Set-Theorem* zeigt, dass die Bedeutung der Cut Sets herkömmlicher FTA auch bei unserer Verallgemeinerung auf dynamische Systeme erhalten bleibt. Dadurch wurde es erstmals möglich, Korrektheit und Vollständigkeit von Fehlerbäumen formal zu beweisen. Basierend auf dieser formalen Semantik wurde eine Fallstudie aus dem Bereich *Verkehrsleittechnik* (Jansen & Schnieder, 1999) formal modelliert (Klose & Thums, 2002) und sicherheitstechnisch analysiert (Reif *et al.*, 2000a; Reif *et al.*, 2000b).

Beweisunterstützung für den Nachweis der FTA-Bedingungen wurde in das Spezifikations- und Verifikationswerkzeug KIV (Balser *et al.*, 2000) integriert. Die ITL-Bedingungen aus der formalen FTA können dort direkt mit Hilfe eines ITL Kalkül (Balser *et al.*, 2002) verifiziert werden. Für die Beschreibung der (technischen) Systeme unterstützen wir die Spezifikation dynamischer mit Statecharts (Harel & Naamad, 1996; Pnueli & Shalev, 1991), die in den Ingenieurwissenschaften häufig eingesetzt werden.

Die Integration von Statecharts in den ITL Kalkül und Regeln für Nachweis temporaler Eigenschaften für Statechart Spezifikationen beschreiben wir in Abschnitt 2 und verweisen für Details auf (Balser & Thums, 2002). In Abschnitt 3 zeigen wir, wie Fehlerbäume zustandsendlicher Systeme mittels Modelchecking automatisch validiert werden können. Eine Beschreibung der Methodik in Abschnitt 4 rundet die formale FTA ab. Im folgenden Abschnitt 5 erläutern wir die Formalisierung der FMEA, die analog zur FTA Semantik in ITL durchgeführt wurde. Die Beweisunterstützung kann dann von der FTA übernommen werden. An drei Fallstudien in Abschnitt 6, 7, bzw. 8 werden die Techniken der formalen FTA und FMEA erprobt und die Methodiken evaluiert werden. Ein Fazit in Abschnitt 9 schließt den Bericht.

2 Semantische Integration und Beweiskalkül

Für die formale Sicherheitsanalyse sind verschiedene Formalismen notwendig, um Systeme, deren Eigenschaften und die formale FTA/FMEA zu beschreiben. Diese müssen auf einer einheitlichen Basis integriert werden, um formale Sicherheitsanalyse durchführen zu können. Wir verwenden *Zustandsübergangssyteme* zu Modellierung der zu untersuchende Systeme. Um Eigenschaften des Modells nachzuweisen, verwenden wir *Intervalltemporallogik* (ITL). Schließlich werden die Sicherheitsanalysetechniken FTA und FMEA als eine Menge von Formeln definiert (siehe (Schellhorn *et al.*, 2002) bzw. Absch. 5), die wiederum in ITL formuliert sind. Das Projekt konzentriert sich auf zeitdiskrete Systeme. Allerdings lassen sich viele Ergebnisse auch auf kontinuierliche Systeme übertragen. So wurde in (Schellhorn *et al.*, 2002) die Formalisierung der FTA für kontinuierliche Systeme im Duration Calculus angegeben und Modelchecking-Versuche damit durchgeführt (siehe Absch. 3.1).

Im Projekt verwenden wir *Statecharts* zur Systemspezifikation. Zum Nachweis von Systemeigenschaften und der FTA- und FMEA-Bedingungen ITL. Dazu ist eine semantische

Integration von Statecharts und ITL notwendig. Zusätzlich wird für den Nachweis von ITL-Eigenschaften ein Beweiskalkül benötigt, der durch ein Werkzeug unterstützt wird.

Wir haben diese Voraussetzungen für die formale Sicherheitsanforderungen geschaffen und sie prototypisch im Spezifikations- und Verifikationswerkzeug KIV (Balser *et al.*, 2000; Reif, 1999) implementiert. Im Folgenden stellen wir die semantische Integration und den Beweiskalkül vor.

2.1 Statecharts in ITL

Als Grundlage für unseren Ansatz dient eine Intervalltemporallogik (Moszkowski, 1985) erster Ordnung mit endlicher Sortenmenge. Funktionen und Prädikate werden mit algebraischen Spezifikationen beschrieben und – wie üblich – über Algebren interpretiert (siehe z. B. (Sperschneider & Antoniou, 1991a)). Die ITL-Semantik basiert auf Intervallen I (auch Traces genannt), die eine endliche oder unendliche Sequenz von Belegungen $I = (\sigma_0, \ldots)$ beschreiben. Jede Belegung σ_i bildet Variablen auf Werte ihres Domains ab. Alle Variablen sind flexibel, wohingegen Funktions- und Prädikatsymbole rigide sind. Als Temporaloperatoren benutzen wir u. a. $\varphi \frown \psi$ (chop), $\Box\ \varphi$ (always), $\Diamond\ \varphi$ (eventually), $\circ\ \varphi$ (strong next) und $\bullet\ \varphi$ (weak next, im Gegensatz zu $\circ$ muss keine nächste Belegung existieren) mit der üblichen ITL Semantik. Mit diesen Operatoren können die benötigten FTA- und FMEA-Bedingungen beschrieben werden.

Syntax. Die Syntax von Statecharts ist bei uns ein strukturierter Text mit Schlüsselworten. Die Statechart-Spezifikation in Abbildung 1 links entspricht der graphischen Repräsentation rechts und beschreibt eine Parallelschaltung von vier Charts. Der ausführlich beschriebene Zeitschalter `TIMER` zählt im geschlossenen Zustand (`TiClose`) einen Zähler hoch. Wenn dieser größer 6 ist, öffnet der Zeitschalter (`TiOpen`). Geschlossen wird er durch ein `reset` Ereignis, der den Zähler (x) wieder zurücksetzt.

chart specification
  BasicChart TiClose;
    import events Tick;
    variables x : nat;
    static reactions
      Tick | begin x := x +1 end;
  OrChart Timer = TiOpen **subcharts** TiClose;
    import variables reset : bool;
    import events Tick;
    initial state TiClose;
    transitions
      TiOpen → reset | begin x := 0 end → TiClose;
      TiClose → x > 6 | → TiOpen;
  AndChart Circuit = Relais1 | Relais2 | Switch | Timer;
end chart specification

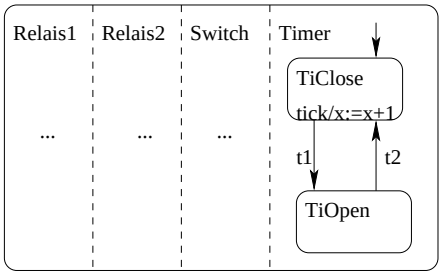


Abbildung 1: Statechart Spezifikation

Die Syntax erlaubt die drei Chart-Typen *BasicChart*, *OrChart* und *AndChart*, wobei

Or- und And-Charts andere Charts importieren können. Dies ist nützlich, um Komponenten wiederverwenden zu können. Die durch *import* spezifizierten Variablen und Ereignisse können in den Charts verwendet werden, sind dort aber nicht definiert. Definiert werden Variablen bzw. Ereignisse im *variables* bzw. *events* Slot. Neben dem *initial state* beschreibt ein Or-Chart die Zustandsübergangsrelation mit *transitions* der Form $s_1 \rightarrow c \mid a \rightarrow s_2$, wobei s_1/s_2 der Ausgangs- bzw. Endzustand, c die Aktivierungsbedingung und a die auszuführende Aktion ist. Die Aktivierungsbedingungen sind, wie bei *static reactions* der Form $c \mid a$, eine boolscher Ausdruck und die auszuführende Aktionen sind sequentielle Programme.

Semantik. Wir haben uns für die STATEMATE-Semantik für Statecharts nach (Harel & Naamad, 1996; Harel *et al.*, 1990) entschieden. Für die Einbettung in ITL werden Statecharts in Traces übersetzt. Dazu haben wir die Formalisierung von (Damm *et al.*, 1998) übernommen, in der ein Statechart SC Traces als Abfolge von Belegungen beschreibt, d. h. $(\sigma_0, \sigma_1, \sigma_2, \ldots) \in traces(SC)$. Die Relation (σ_i, σ_{i+1}) entspricht der Übergangsrelation des Statecharts SC. Damit werden Statecharts wie folgt interpretiert:

$$(\sigma_0, \sigma_1, \ldots) \models SC :\Leftrightarrow (\sigma_0, \sigma_1, \ldots) \in traces(SC)$$

2.2 Neuer Kalkül für ITL und Statecharts

Die Grundidee des Kalküls für ITL und Statecharts ist *symbolische Ausführung* und *Induktion*. Diese Vorgehensweise ist aus der Programmverifikation bekannt und hat sich dort sehr gut bewährt, da ein Beweis der Programmstruktur folgt und damit für den Beweisführenden leicht verständlich ist. Dieses Paradigma wurde auf ITL und Statecharts übertragen und in einen Sequenzenkalkül integriert. Dieser Kalkül eignet sich deshalb sehr gut als Grundlage für das symbolische Ausführen, da der Sequenzenkalkül (wenn man die Regeln von unten nach oben liest) auf der Vereinfachung von Beweisverpflichtungen durch Abarbeiten der logischen Operatoren beruht. Dies kommt dem Abwickeln oder Ausführen von Temporaloperatoren bzw. Statecharts nahe. Im Folgenden werden wir den Sequenzenkalkül für Prädikatenlogik als bekannt voraussetzen (Sperschneider & Antoniou, 1991b).

ITL. Für jeden ITL-Operator sind zwei Sequenzenkalkülregeln notwendig die beschreiben, wie der Operator im Antezedenten bzw. im Sukzedenten abgewickelt wird. Exemplarisch geben wir die Regeln für den $\Box$-Operator an.

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \bullet \Box \varphi, \Delta}{\Gamma \vdash \Box \varphi, \Delta} \; \textit{always right} \qquad \frac{\varphi, \bullet \Box \varphi, \Gamma \vdash \Delta}{\Box \varphi, \Gamma \vdash \Delta} \; \textit{always left}$$

Z. B. zerlegt die *always right* Regeln $\Box \varphi$ im Sukzedenten einer Sequenz in zwei Prämissen. In der linken Prämisse wird nachgewiesen, dass φ im aktuellen Zustand gilt, und in der rechten Prämisse wird die Eigenschaft für den restlichen Trace postuliert. Das Abwickeln von $\Diamond \varphi$, $\circ \varphi$ und $\bullet \varphi$ ist offensichtlich. Für Details zum Abwickeln von $\varphi \frown \psi$ und weiteren Temporaloperatoren verweisen wir auf (Balser *et al.*, 2002).

Die Temporaloperatoren werden nun solange abgewickelt, bis jede Temporalformel in Γ und Δ mit einem *Next*-Operator beginnt, und alle anderen Formeln γ und δ prädikatenlogisch sind. Dann können wir einen Schritt im Trace fortschreiten indem die Regel *step* (ähnlich zur *next*-Regel in (Biundo *et al.*, 1992)) angewandt wird.

$$\frac{\gamma_0, \Gamma \vdash \delta_0, \Delta}{\gamma, \circ \Gamma \vdash \delta, \bullet \Delta} \; \textit{step}$$

Die führenden *next*-Operatoren werden entfernt und die entsprechenden Werte der Variablen nach Durchführung des Schritts in γ_0 bzw. δ_0 gespeichert. So wird durch Abwickeln von Operatoren und Ausführen von Schritten der Trace nach und nach „abgearbeitet“.

Statecharts. Statecharts werden zur Beschreibung von Systemen benutzt und können als Voraussetzung für die nachzuweisende Eigenschaft gesehen werden (wenn sich das System so verhält, wie im Statechart beschrieben, dann gilt...). Deshalb stehen Statecharts im Antezedenten und es ist nur eine Regel für das Ausführen von Statecharts notwendig, die mit folgendem Regelschema beschrieben wird:

$$\frac{\Big(\ cond(stp_i), exec(stp_i), SC, \Gamma \vdash \Delta\ \Big)_{i=1\ldots n}}{SC, \Gamma \vdash \Delta}\ sc\ unwind$$

Im Gegensatz zu sequentiellen oder parallelen Programmen (Harel, 1984; Balser *et al.*, 2002), die sich beim Ausführen ändern (abgearbeitet werden), beschreibt ein Statechart SC eine statische Übergangsrelation, die von Schritt zu Schritt unverändert bleibt. Deshalb steht auch SC (unverändert) in den Prämissen der Regel. Die aktuelle Belegung (Variablenbelegung, aktive Zustände und Ereignisse) werden durch die Formeln Γ und Δ beschrieben und durch Ausführen eines Schritts $exec(stp_i)$ modifiziert. Da Statecharts indeterministisch sein können, müssen bei einem Schritt n Prämissen, eine für jeden möglichen, indeterministischen Schritt stp_i, betrachtet werden. Die Aktivierungsbedingung $cond(stp_i)$ führt Einschränkungen für das Ausführen eines Schritts ein, die sich aus den Aktivierungsbedingungen der einzelnen Transitionen dieses Schritts zusammensetzen.

Für die Berechnung der möglichen Übergänge wurde der *step*-Algorithmus von (Damm *et al.*, 1998) verwendet und erweitert. Die Erweiterung war notwendig, da durch die zugrundeliegenden algebraischen Spezifikationen die Aktivierungsbedingungen der Übergänge durch symbolische Werte beschrieben werden können und damit interaktive Beweisschritte notwendig sind, um diese Bedingungen auszuwerten.

Neben dem Ausführen der Aktionen eines Zustandübergangs, setzt $exec(stp_i)$ die Ereignisse zurück (Ereignisse sind nur einen Schritt aktiv). Für die Beschreibung der Aktionen in Statemate benutzen wir sequentielle Programme, die durch die Dynamische Logik (Harel, 1984) in KIV unterstützt werden.

Ein vollständiges Regelsystem für die DL findet man z. B. in (Heisel *et al.*, 1988), eine detaillierte Beschreibung der Integration algebraischen Spezifikationen, ITL und Statecharts mit der dazugehörigen, uniformen Beweistechnik in (Balser & Thums, 2002).

Fazit. Der entwickelte Kalkül hat drei Vorteile. Der ITL-Framework eignet sich gut, um Systembeschreibungen explizit in die Logik mit aufzunehmen. Statecharts sind Formeln der Logik. Sie werden nicht in reine Temporallogik kodiert, wie z. B. die Systembeschreibungen in (Manna & the STeP group, 1994) und (Lamport, 1994). Zweitens erhält man durch das symbolische Ausführen lokale Invarianten. In (Balser *et al.*, 2002) wurde festgestellt, dass die lokalen Invarianten im Vergleich zu den globalen in (Manna & Pnueli, 1995) wesentlich kleiner werden. Schließlich ist das Beweisprinzip des symbolischen Ausführens sehr gut automatisierbar. Die konkrete Durchführung der Automatisierung mit entsprechenden Heuristiken für Statecharts steht aber noch aus.

3 Modelchecking von Fehlerbäumen

Ein hoher Automatisierungsgrad erleichtert die Anwendung formale Sicherheitsanalyse. Deshalb haben wir untersucht, inwieweit der automatische Nachweis der Fehlerbaumbedingungen mit Modelcheckern möglich ist.

In (Schellhorn *et al.*, 2002) wurden Bedingungen für die verschiedenen Gatter eines Fehlerbaums vorgestellt. Diese Bedingungen garantieren, dass bei einem erfolgreichen Nachweis über einem formalen Modell der Fehlerbaum vollständig und korrekt ist. Das bedeutet, dass es keine weiteren Fehlerursachen für das Top-Ereignis im Fehlerbaum gibt, bzw. dass die Ursachen tatsächlich zum Top-Ereignis führen. Für zeitdiskrete Systeme sind die FTA-Bedingungen in Intervalltemporallogik (ITL), für kontinuierliche Systeme im Duration Calculus (DC) formuliert. Zusätzlich zu den üblichen Gattern wurden sogenannte Ursache/Wirkungs Gatter eingeführt, die in dynamischen Systemen zum Einsatz kommen. Die Vollständigkeitsbedingung für ein Ursache/Wirkungs-Und Gatter lautet z. B. $\Box\neg\ (\neg\ \Diamond(\psi_1 \wedge \psi_2); \varphi)$ und bedeutet umgangssprachlich, dass es nicht sein kann, dass es einen Ablauf in dem System zur Wirkung φ gibt, auf dem zuvor nicht irgendwann die Ursachen ψ_1 und ψ_2 aufgetreten sind. Solche Bedingungen sollen nun automatisch geprüft werden.

3.1 DC Modelchecking von Fehlerbaumbedingungen

Um zu untersuchen, ob sich die Fehlerbaumbedingungen für kontinuierliche Systeme direkt in einem Modelchecker nachweisen lassen, haben wir gemeinsam mit dem Projekt *PDZ* (Prof. Olderog) Versuche mit dem dort entwickelten DC-Modelchecker Moby/DC (Tapken, 2001) durchgeführt.

Dieser Modelchecker weist Eigenschaften für eine Untermenge von DC nach, die mit sog. Phasenautomaten beschrieben werden können. Die Phasenautomaten werden sowohl für die Beschreibung des Modells als auch für die nachzuweisenden Aussagen benutzt und beschreiben Abläufe in einem System. Verifiziert wird dann die Negation der nachzuweisenden Aussage, d.h. solch ein Ablauf kann niemals auftreten. Dies entspricht der Formel $\neg\ F$ mit F aus folgender Grammatik (P ist ein Zustandsausdruck und $expr(l)$ ein Intervallausdruck über l, der Länge des Intervalls).

$$\begin{aligned} F \quad ::= \quad & (F1 \vee F2) \mid (\lceil P \rceil \wedge expr(l)); F \mid ((\lceil P \rceil \wedge expr(l)) \vee l = 0); F \mid \\ & (\lceil P \rceil \wedge expr(l)) \mid ((\lceil P \rceil \wedge expr(l)) \vee l = 0) \end{aligned}$$

Die Ursache/Wirkungs-Beziehung für die Fehlerbaumbedingung kann aber nicht mit solchen Abläufen beschrieben werden, da Negationen in der Formel auftreten. Man müsste negierte Abläufe beschreiben können. Dies ist mit Phasenautomaten nicht möglich.

Die FTA-Beweispflichten können deshalb nicht direkt in Moby/DC ausgedrückt werden. Am Lehrstuhl von Prof. Olderog ist daraufhin eine Diplomarbeit entstanden (Schäfer, 2001), in der eine Kodierung unserer FTA-Beweispflichten in Phasenautomaten entwickelt wurde. Wir geben im nächsten Abschnitt eine Alternativlösung an, bei der die Vollständigkeit von Fehlerbäumen mit konventionellen CTL-Modellprüfern nachgewiesen werden kann.

3.2 CTL-Modelchecking für eingeschränkte Fehlerbäume

Eigenschaften für zeitdiskrete Systeme lassen sich mit CTL-Modellprüfern nachweisen. Allerdings lassen sich allgemeine Fehlerbäume nicht in CTL beschreiben. Sobald Ursachen eine

zeitliche Ausdehnung besitzen, kann die entsprechende Ursache/Wirkungs-Bedingungen im Fehlerbaum nicht korrekt ausgedrückt werden. Ist z. B. die Ursache für eine Explosion (φ) der Austritt von Gas (ψ_1) und dazu irgendwann ein Funke (ψ_2), kann das im DC wie folgt beschrieben werden: $(\psi_1 \wedge \Diamond\ \psi_2); \varphi$.
Insbesondere beschreibt diese Formel, dass die Ursache $\psi_1 \wedge \Diamond\ \psi_2$ abgeschlossen sein muss (sowohl muss Gas ausgetreten sein, als auch der Funke aufgetreten), bevor die Wirkung eintritt. Dieser Sachverhalt ist nicht in CTL und nicht einmal in der ausdrucksstärkeren Logik CTL* ausdrückbar. Ist die Dauer der Ursache bekannt (z. B. bevor eine bestimmte Konzentration überschritten wird, muss zuvor 10sek. Gas austreten), kann der Sachverhalt in CTL* beschrieben werden, nicht aber in CTL, da diese Untermenge von CTL* verlangt, dass vor jedem Temporaloperator ein Pfadquantor (alle Pfade ..., oder es gibt einen Pfad ...) steht. Dieser verhindert, dass über genau den Pfad gesprochen wird, auf dem die Wirkung folgt.

Schränkt man die Fehlerbäume ein, so dass nur Ereignisse, die keine zeitliche Ausdehnung besitzen (z. B. eine Explosion φ erfolgt, nachdem ein Funke ψ sprüht), im Fehlerbaum auftreten, kann die Ursache/Wirkungs-Beziehung mit dem CTL-Operator *weak before (wB)* beschrieben werden. Umgangssprachlich bedeutet er, dass wenn φ gilt, zuvor ψ gegolten haben muss. Damit entspricht er der Ursache/Wirkungs-Beziehung der FTA.

Nimmt man Änderungen im Systemmodell in Kauf, können auch Ereignisse mit einer zeitlichen Ausdehnung behandelt werden, indem diese Ereignisse auf ihr Ende reduziert werden. D. h., dass z. B. das Ereignis *10sek Gasaustritt* beschrieben wird, indem das Ende dieses Ereignisses betrachtet wird, nämlich *es ist 10sek lang Gas ausgetreten*, die 10sek sind also vorbei. Dazu muss das Systemmodell um einem Automaten (U/W-Chart) ergänzt werden, der diese Beziehung beschreibt (siehe Abb. 2) und parallel zum vorhandenen Systemmodell geschaltet wird. Für jedes Ursache/Wirkungs-Gatter im Fehlerbaum, das Ereignissen mit zeitlicher Ausdehnung verknüpft, ist ein solcher U/W-Chart erforderlich. Der initiale Zustand eines U/W-Charts ist `IDLE`. Beobachtet er n Zeitschritte lang die Ursache ψ (im Automaten durch $\psi, [n]$ abgekürzt), geht er in den `OK` Zustand über, falls jedoch zuvor die Wirkung φ auftritt, in den `KO` Zustand. φ und ψ sind dabei Formeln (ohne Temporaloperatoren) über den Zuständen des Systemmodells, z. B. für zwei Ursachen ψ_1 und ψ_2, die zusammen auftreten müssen, ist $\psi = \psi_1 \wedge \psi_2$.

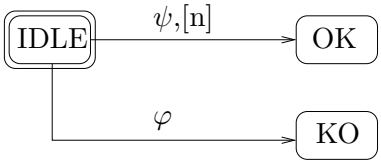


Abbildung 2: U/W-Chart

Damit beschreibt der Zustand `OK` das Ende des zeitlich ausgedehnten Ereignisses $\psi, [n]$ und mit `OK` wB φ wird die Ursache/Wirkungs-Beziehung nachgewiesen. Alternativ kann auch die Bedingung $AG \neg$ `KO` (auf allen Pfaden gilt immer $\neg$ `KO`) als Vollständigkeitsbedingung nachgewiesen werden, da der Zustand `KO` nur betreten wird, wenn zuvor keine Ursache auftrat. Kann dies gezeigt werden, tritt also immer die Ursache vor der Wirkung auf und der Fehlerbaum ist vollständig.

Fazit. Ursache/Wirkung-Beziehungen von Fehlerbäumen können mit CTL-Modelcheckern nachgewiesen werden. Falls im Fehlerbaum Ursachen mit zeitlicher Ausdehnung auftreten, können sog. U/W-Charts parallel zum Systemmodell geschaltet werden, die diese Ereignisse auf deren Ende reduzieren. In der Fallstudie „Pressure Tank“ in Abschnitt 7 wird die oben beschriebene Technik angewendet, um eine formale Fehlerbaumanalyse durchzuführen.

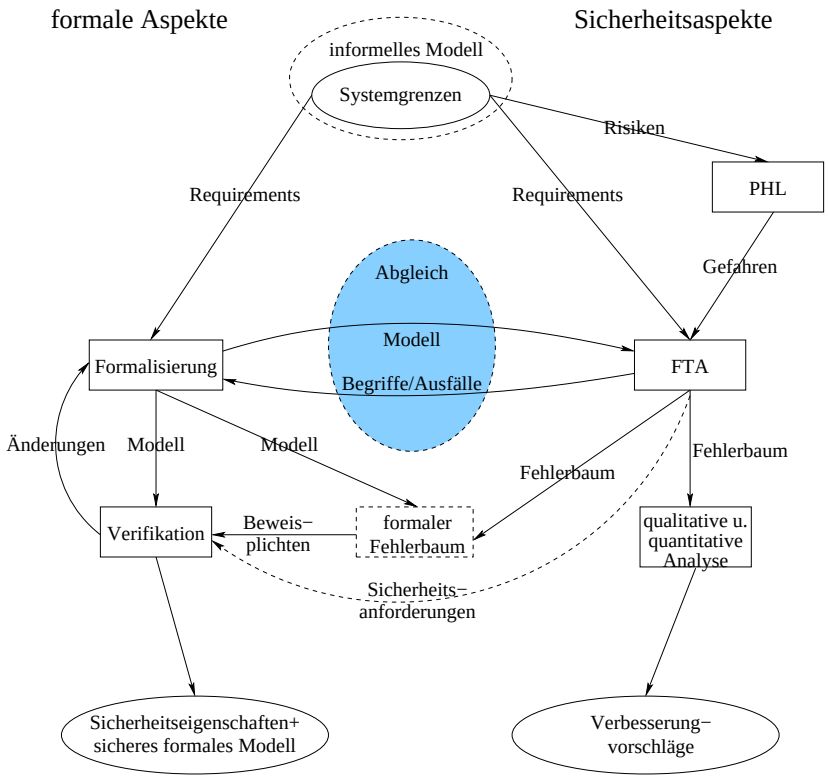


Abbildung 3: Methodik

4 Methodik formaler FTA

Ein wesentlicher Punkt bei der Entwicklung sicherheitsrelevanter Software ist, dass die Sicherheitsaspekte „von Anfang an“ in das System integriert und nicht nachträglich hinzugefügt werden („Safety must be designed into a System“ (Leveson, 1995)). Nachträgliches Hinzufügen von (Sicherheits-) Funktionalität macht ein System komplizierter und damit anfälliger für Fehlverhalten.

Deshalb schlagen wir vor, die Sicherheitsanalyse während des Designs durchzuführen. Die Abbildung 3 soll unsere Vorgehensweise verdeutlichen. Als erstes wird aus dem *informellen Modell* eine *Gefährdungsliste* erstellt (preliminary hazard list, PHL), die alle möglichen Gefahren, die vom zu untersuchenden System ausgehen, enthält. Sie dient als Grundlage für die *Fehlerbaumanalyse* (fault tree analysis, FTA), die parallel zur *Formalisierung* bzw. formalen Modellbildung des in Frage stehenden Systems durchgeführt wird. Beide Techniken haben verschiedene Sichten auf das System. Während bei der Formalisierung die Funktionalität des betrachteten System bei regulärem Verhalten der Umgebung/Komponenten im Mittelpunkt steht, fokussiert die Fehlerbaumanalyse auf Komponentenausfälle und unerwartete Umgebungsereignisse und liefert damit wichtige Hinweise auf Schwachstellen.

Wichtig ist bei der parallelen Durchführung der Formalisierung und der FTA, dass die beiden Techniken die gleichen *Systemgrenzen* berücksichtigen, da für die spätere Formalisierung der FTA alle auftretenden Ereignisse des Fehlerbaums im formalen Modell ausgedrückt werden müssen. Dies geschieht durch *Abgleich* der beiden Modelle. Dabei liefert das formale

Modell Hinweise für die FTA, z. B. welche Systemaspekte nicht bedacht wurden oder vernachlässigbar sind. Die FTA enthält andererseits *Begriffe* und Beschreibungen von *Ausfällen*, die in der Formalisierung berücksichtigt werden müssen. Der Sicherheitsexperte entscheidet dann, welche Ausfälle für das System relevant sind und im formalen Modell berücksichtigt werden müssen, und welche so unwahrscheinlich sind, dass sie nicht berücksichtigt werden müssen. Der Abgleich der Formalisierung mit der Fehlerbaumanalyse ist im allgemeinen ein iterativer Prozess, der dann abgeschlossen ist, wenn jedes Fehlerbaumereignis und alle relevanten Ausfälle formal beschrieben werden können.

Hardware- vs. Softwareausfälle. Die FTA eines softwarebasierten Systems deckt sowohl relevante Hardware- als auch Softwareausfälle auf. Hardwarefehler kann man nicht ausschließen, da sie nicht konstruktionsbedingt sind, sondern durch betriebsbedingte Defekte hervorgerufen werden. Deshalb kann man solche Ursachen nicht ausschließen, sondern nur die Ausfallwahrscheinlichkeit durch Wahl zuverlässiger Komponenten bzw. Einbau redundanter Komponenten verringern. An dieser Stelle hilft die FTA verschiedene Systemmodelle miteinander zu vergleichen und Vorschläge für Verbesserungen des Modells zu liefern.

Für eine formale Sicherheitsanalyse müssen hardwarebedingte Komponentenausfälle aus der FTA im formalen Modell berücksichtigt werden (siehe Abgleich zwischen Formalisierung und FTA). Ansonsten ist es unmöglich, die Wirkung möglicher Komponentenausfälle im Modell zu betrachten. Soll eine Wirkung nach einem Komponentenausfall gezeigt werden (Komponentenausfall *impliziert* Wirkung) ist dies in einem Modell ohne Komponentenausfall immer trivial möglich, da die Vorbedingung der Implikation nie erfüllt wird.

Für softwarebedingte Fehler sehen wir eine andere Vorgehensweise vor. Hier folgen wir Leveson: „... if design errors are found in the tree through this process, they should be fixed rather than left in the code and assigned a probability“ (Leveson, 1995). Für Softwarekomponenten fordern wir deshalb die formale Verifikation.

Schwach vs. stark gekoppelte formale Sicherheitsanalyse. Für das weitere Vorgehen sehen wir zwei Alternativen vor, die sich in verschiedenen Fallstudien bewährt haben. Sie unterscheiden sich dadurch, wie sehr die FTA an das formale Modell gekoppelt ist. Bei der *schwachen Kopplung* leiten wir aus der FTA *Sicherheitsanforderungen* für das formale Modell ab, die dort formalisiert und nachgewiesen werden. Bei der *starken Kopplung* führen wir eine vollständige formale Fehlerbaumanalyse durch. Dazu werden die Ereignisse im Fehlerbaum formalisiert und man erhält einen *formalen Fehlerbaum*. Dieser beschreibt *Beweispflichten* für das System. Dies sind Vollständigkeitsbedingungen für den Fehlerbaum deren Nachweis garantiert, dass alle Ursachen für das Top-Ereignis des Fehlerbaums berücksichtigt worden sind (Schellhorn *et al.*, 2002).

Modelchecking vs. interaktive Verifikation. Für die *Verifikation* der Beweispflichten verwenden wir, wenn möglich, die automatische Modellprüfung. Neben der Voraussetzung, dass das zugrundeliegende Systemmodell zustandsendlich sein muss, ergeben sich bei der Modellprüfungen noch weitere Einschränkungen bzgl. der Ausdrucksmächtigkeit der Vollständigkeitsbedingungen (siehe dazu Absch. 3). Ist die Modellprüfung nicht möglich, verwenden wir zum Nachweis interaktive Verifikation. Die Grundlagen dazu wurden in Absch. 2 gelegt.

Beim Modellprüfungsansatz haben wir bei verschiedenen Beispielen ein interessante Beobachtung gemacht. Die Modellprüfer benötigen meist weniger Zeit für den Nachweis, dass

bei Ausschluss des minimal cut sets das Top-Ereignis des Fehlerbaums nicht auftritt, als für den Nachweis der einzelnen Vollständigkeitsbedingungen. Wir schließen daraus, dass es den Modellprüfern nicht gelingt, die dadurch gegebene Beweisstrukturierung auszunutzen. Für die interaktive Verifikation scheinen die Vollständigkeitsbedingungen jedoch eine günstige Beweisstrukturierung zu ergeben.

Ergebnisse. Als Ergebnis der formalen Sicherheitsanalyse erhalten wir auf der einen Seite *Sicherheitseigenschaften* für Softwarekomponenten und ein *sicheres, formales Modell*. Auf der anderen Seite liefert die Fehlerbaumanalyse *Verbesserungvorschläge*, um Sicherheitslücken des Systems zu schließen. Die Verbesserungsvorschläge resultieren aus der *qualitativen* und *quantitativen Analyse* des Fehlerbaums, die aus der klassischen FTA bekannt sind (Vesely *et al.*, 1981). Dieser gegenseitige Zugewinn hat sich eindrucksvoll bei den Fallstudien „Elbtunnel“ (siehe Absch. 6) und „Pressure Tank“ (siehe Absch. 7) gezeigt. Für die erste Fallstudie wurde eine schwach gekoppelte, für die zweite eine stark gekoppelte formale Sicherheitsanalyse durchgeführt. Es gibt auch Systeme, für die sich FTA nicht eignet. Ein solches Beispiel geben wir in Abschnitt 8.

5 Formale FMEA

Bei der Fehlermöglichkeits- und Einflußanalyse (failure mode and effect analysis, FMEA) handelt es sich um eine vorwärtsgerichtete Analyse. Ausgehend von einer Komponente werden all ihre möglichen Fehlverhalten (Fehlermodi), deren Ursachen und die daraus resultierenden Wirkungen auf die nächst höhere Komponentenebene bzw. das Gesamtsystem untersucht. Neben der Auftrittswahrscheinlichkeit des Fehlermodus und der Relevanz der resultierenden Wirkung (Gefahr) werden mögliche Kontroll- und Verbesserungsvorschläge erarbeitet. Die Ergebnisse dieser Analyse werden in einer Tabelle wie in Abbildung 4 zusammengefasst.

FMEA						
Name	Fehler-modus	Ursache	Auswirkung	Wahrschein-lichkeit	Relevanz	Kontroll- u. Verbesserungs-maßnahmen
Schalter	fails open	„verklebt“	Überlastung Motor	$1*10^{-6}$	kritisch	regelmäßige Wartung
...	...	...	...	...	...	...

Abbildung 4: FMEA Tabelle

Für eine formale Behandlung der FMEA sind die Beziehungen zwischen *Fehlermodus* und *Ursache* bzw. *Fehlermodus* und *Auswirkung* interessant. Im ersten Fall müssen für einen Fehlermodus alle möglichen Ursachen in der Tabelle aufgelistet sein. Damit entspricht diese Beziehung der Vollständigkeitsbeziehung in einem Fehlerbaum (siehe (Schellhorn *et al.*, 2002)). Jedoch wird eine FMEA meist auf der Komponentenebene durchgeführt, die im formalen Modell nicht mehr detaillierter beschrieben ist. Z. B. macht es wenig Sinn die Ursache „Verkleben“ eines defekten Schalters (siehe FMEA Tabelle, Abb. 4) in einem formalen Modell

zu beschreiben. Deshalb haben wir uns bei der Formalisierung der FMEA auf die Beziehung zwischen Fehlermodus und dessen Auswirkung beschränkt.

Semantik Wir formalisieren die Fehlermodus/Auswirkungs-Beziehung der FMEA wie die Korrektheits- und Vollständigkeitsbeziehungen der FTA im Duration Calculus (DC) für kontinuierliche bzw. in Intervalltemporallogik (ITL) für zeitdiskrete Systeme. Die Beziehung, die wir beschreiben möchten, lautet informell: „Die Ursache φ führt zur Wirkung ψ“. Im DC lässt sich diese Beziehung folgendermaßen beschreiben:

$$\diamondsuit\ \psi \rightarrow \diamondsuit\ \varphi. \tag{1}$$

$\diamondsuit\ \varphi$ bedeutet informell: es gibt einen Anfangsstück (im Trace), in dem φ gilt. Die Formel (1) bedeutet „Falls eine Ursache ψ auftritt, dann gibt es (mindestens) eine zeitlich Entwicklung des Systems, so dass auch die Wirkung φ eintritt.“. Im allgemeinen kann die stärkere Forderung, dass nach jeder Ursache eine Wirkung folgt, nicht erfüllt werden. Z. B. muss nach einer Zugkollision (Wirkung) ein Bremsausfall (erneute Ursache) nicht zu einer zweiten Kollision führen.

Methodik Zur Durchführung einer formalen FMEA geht man folgendermassen vor: Nachdem das System modelliert wurde, müssen die unerwünschten Gefährdungen (Wirkungen) definiert und formalisiert werden. Dies kann wie bei der FTA durch Erstellen einer Gefährdungsliste (preliminary hazard list, PHL) geschehen (siehe auch Abschnitt 4) oder aus dem jeweiligen Expertenwissen abgeleitet werden. Anschließend muss der zu untersuchende Fehlermodus definiert und formalisiert werden. Um ihn untersuchen zu können, muss er im Ausgangsmodell berücksichtigt werden. Das Modell M wird zu einem Modell M' modifiziert, in dem dieser Fehlermodus auftreten kann. Das Modell M' ist dann die Grundlage für den Nachweis der obigen Fehlermodus/Auswirkungs-Beziehung.

Im bisher gesagten haben wir nur *single point of failures* betrachtet. Das sind Fehler die nur von einem einzigen Komponentenausfall herrühren. Das ist auch der Weg, der ausnahmslos bei informeller FMEA beschritten wird. Mittels formaler FMEA ist es nun auch möglich zusammengesetzte Fehlerursachen zu untersuchen. In DC Formeln bedeutet das: $(\diamondsuit\ \varphi_1 \wedge \varphi_2) \rightarrow \diamondsuit\ \psi$. Die Anzahl der auftretenden elementaren Fehlursachen spielt bei der formalen FMEA prinzipiell keine Rolle. Dadurch werden die Analysemöglichkeiten gegenüber der herkömmlichen FMEA erweitert.

In der Literatur gibt es nur wenige Ansätze zur formalen Behandlung von FMEA. In (Cichicki & Górski, 2001; Cichicki & Górski, 2000) wird ein Ansatz, der ähnlich zu unserem ist, beschrieben. Auch dort wird ein formales Modell erstellt, in das für die FMEA Fehlerfunktionalität hinzugefügt wird. Als Referenz wird ein abstraktes (CSP) Modell und eine korrektes, verfeinertes Modell erstellt. In das verfeinerte Modell werden dann Fehler eingebaut. Kann dann die Verfeinerungsbeziehung nicht mehr nachgewiesen werden, hat dieser Fehler eine Auswirkung auf die Korrektheit.

Damit kann allerdings nur gezeigt werden, dass ein Fehlermodus eine Auswirkung auf die Systemsicherheit hat. In unserem Ansatz kann dagegen detailliert nachgewiesen werden, zu welchen Gefährdungen ein Fehlermodus führt und ob dies mit der informellen FMEA übereinstimmt.

Toolsupport Da die formale FMEA stark mit der formalen FTA korreliert ist - sie verwenden dieselbe Logik (DC bzw. ITL) und Spezifikationssprache (Statecharts) -, ergaben sich auch für den Toolsupport positive Synergieeffekte. Die Werkzeuge zum Nachweis von FTA-Bedingungen können auch zum Nachweis der FMEA-Bedingungen benutzt werden. Beim automatischen Nachweis mit Modelcheckern müssen im Gegensatz zur FTA (siehe Absch. 3) keine weiteren Anpassungen des Modells vorgenommen werden. Für nicht zustandsendliche Systeme wird der in Abschnitt 2 beschriebene Beweiskalkül zum Nachweis über Statechart Modellen benutzt.

Ergebnis. Die Formalisierung der FMEA erlaubt, die Fehlermodus/Auswirkungs-Analyse formal nachzuweisen. Die dazu notwendigen Bedingungen sind in der gleichen Logik wie die Vollständigkeits- und Korrektheitsbedingungen der FTA formuliert, können aber auch in weniger ausdrucksstarken Logiken (z. B. in CTL) beschrieben werden. Die intensive Beschäftigung mit der Formalisierung der FTA hat uns sehr geholfen, die FMEA adäquat zu formalisieren.

6 Schwach gekoppelte formale FTA: Fallstudie „Elbtunnel“

Im Rahmen einer Kooperation mit der Siemens AG - Abteilung “Industrial Solutions and Services” - wurde eine Sicherheitsanalyse des Steuersystems der Höhenkontrolle für den neuen Elbtunnel in Hamburg durchgeführt (Ortmeier *et al.*, 2002). Es handelt sich um ein System, das 2003 in den realen Einsatz gehen wird. Es wurde eine schwach gekoppelte formale Sicherheitsanalyse, wie in Abschnitt 4 beschrieben, durchgeführt. Für die Höhenkontrolle wurde ein formales Modell erstellt und eine informelle Fehlerbaumanalyse durchgeführt. Aus der FTA wurden Sicherheitsanforderungen abgeleitet, die im formalen Modell nachgewiesen wurden. Außerdem hat die FTA Schwächen der Höhenkontrolle aufgedeckt und entsprechende Verbesserungvorschläge geliefert. Die Resultate dieser Fallstudie sind außerordentlich positiv und unterstreichen den praktischen Nutzen der von uns entwickelten Techniken. Das Vorgehen entspricht einer „schwach gekoppelten Analyse“, da die FTA selbst nicht formalisiert wird.

Im Folgenden wird kurz die Problematik des Systems Höhenkontrolle - das Kernelement unserer Fallstudie - erläutert, bevor wir auf die verwendeten Analysetechniken und die Ergebnisse eingehen. Abbildung 5 zeigt eine Skizze des Hamburger Elbtunnels. Die Röhren eins bis drei existieren seit längerem, die vierte Röhre wurde neu gebaut und im Herbst Ende des Jahres 2002 in Betrieb genommen werden.

In den alten Röhren ist das Passieren für überhohe LKWs (Fahrzeuge mit über 4m Höhe) unmöglich. Die vierte Röhre wird höher, und damit auch von überhohen LKWs (üLKWs) befahrbar sein. Mit dem derzeitigen Höhenkontrollsystem für die Röhren 1-3 werden ca. 700 Fehlalarme pro Jahr (üLKWs signalisiert, obwohl keiner einfährt) ausgelöst. Es ist mit einem Anstieg dieser Zahl zu rechnen, da mit Eröffnung der 4.ten Röhre auch tatsächlich überhohe Fahrzeuge den Tunnel (genauer die 4. Röhre) befahren dürfen.

Darum ist ein neues Kontrollsystem notwendig. Diese Höhenkontrolle hat zwei Ziele. Erstens soll durch einen Alarm und anschließende Sperrung der Spuren sichergestellt werden, dass keine Kollisionen überhoher Fahrzeuge mit den Röhren eins, zwei und drei stattfinden kann, und zweitens soll die Zahl der Fehlalarme deutlich gesenkt werden.

Im Folgenden betrachten wir nur den nördliche Tunneleingang, von dem aus die neue, 4. Röhre befahren werden kann. Das Kontrollsystem besteht im wesentlichen aus drei Kontrollzonen: Vorkontrolle, Nachkontrolle und Endkontrolle. Die Vorkontrolle besteht aus einem Licht-

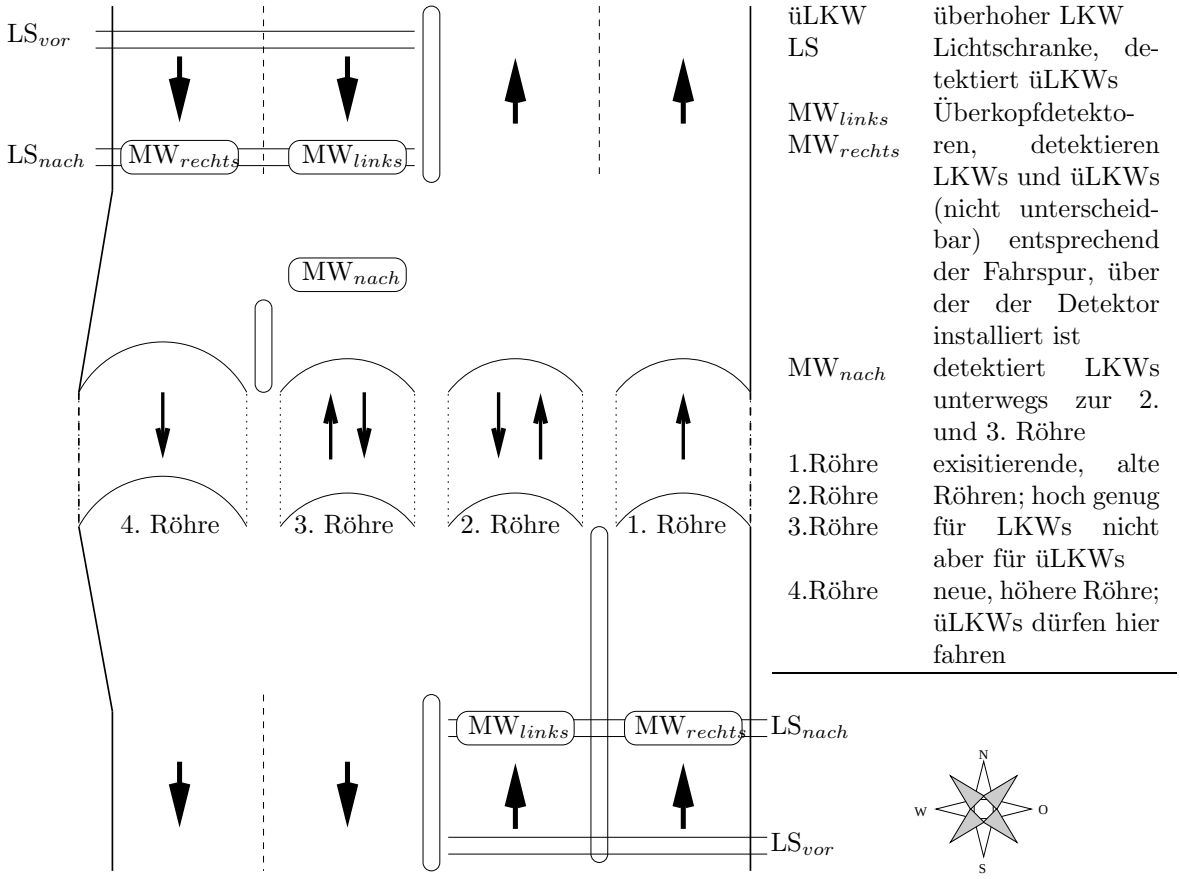


Abbildung 5: Skizze der Tunneleingänge

schrankenpaar (LS$_{vor}$), die Nachkontrolle besteht ebenfalls aus einem Lichtschrankenpaar und zusätzlichen Überkopfdetektoren (LS$_{nach}$ und MW$_{links,rechts}$) für die einzelnen Fahrbahnen. Die Endkontrollsensoren (MW$_{nach}$) sind Überkopfdetektoren knapp 300m vor der Tunneleinfahrt. Die Vorkontrolle ist immer aktiviert. Bei Auslösung der Vorkontrolle wird ein Timer - Timer 1 - gestartet. Nur während dieser läuft sind die Sensoren der Nachkontrolle aktiviert. Eine Auslösung der Nachkontrolle aktiviert einen weiteren Timer - Timer 2 - der die Endkontrolle scharf schaltet. Nach Ablauf von Timer 2 wird die Endkontrolle wieder deaktiviert. Folgende zusätzliche Logik ist implementiert:

- Die Zahl der üLKWs zwischen Vor- und Nachkontrolle wird in einem Zähler ZV gespeichert
- Jede Auslösung von LS$_{vor}$ erhöht den Zähler ZV um eins und startet Timer 1 neu
- Jede Auslösung von LS$_{nach}$ - falls die Nachkontrolle aktiviert ist - dekrementiert ZV um eins und startet Timer 2 neu
- Bei aktivierter Nachkontrolle wird ein Alarm ausgelöst, wenn ein hohes Fahrzeug auf einer anderen als der in Fahrtrichtung rechten Spur detektiert wird

- Bei aktivierter Endkontrolle löst jedes dort detektierte hohe Fahrzeug einen Alarm aus

Diese Steuerung wurde einer formalen Sicherheitsanalyse unterzogen. Es wurden die Techniken Simulation, Modellprüfung und Fehlerbaumanalyse kombiniert. Das wichtigste inhaltliche Resultat ist, dass das System eine Sicherheitslücke aufweist. Dieses Risiko tritt bei gleichzeitigem Durchfahren von zwei überhohen Fahrzeugen durch die Vorkontrolle auf. Ein Lichtschrankensensor kann aber bauartbedingt nicht unterscheiden, wieviele unterschiedliche überhohe LKWs sein Signal unterbrechen. Dies hat zur Folge, dass der interne Zähler ZV nur um eins inkrementiert wird - obwohl zwei üLKWs in den Gefahrenbereich einfahren. Da das System nun ein potentiell gefährliches Fahrzeug „vergessen“ hat, kann der Fall eintreten, dass sowohl Nach- als auch Endkontrolle bereits wieder deaktiviert sind, bevor das Fahrzeug die entsprechenden Messpunkte erreicht. Somit ist eine Kollision möglich, ohne dass zuvor ein Alarm ausgelöst wird.

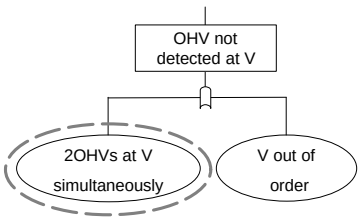


Abbildung 6: Sicherheitslücke

Dieses Ergebnis kann theoretisch mit allen drei verwendeten Analysetechniken gewonnen werden. Es ist jedoch äußerst unwahrscheinlich diese Sicherheitslücke mittels Simulation zu finden, da nicht nur gleichzeitiges Durchfahren von LS_{vor}, sondern auch noch stark unterschiedliche Geschwindigkeiten notwendig sind. Auch bei der informellen Fehlerbaumanalyse wurde das Problem nicht entdeckt, da es unterhalb des betrachteten Detaillierungsgrads lag.

Mittels formaler Modellprüfung konnte dieses Problem aber entdeckt werden und der Fehlerbaum wurde, wie in Abbildung 6 dargestellt, erweitert. Die gestrichelt umrandete Ursache beschreibt das gleichzeitige Durchfahren der Vorkontrolle durch zwei üLKWs, die im Fehlerbaum zusätzlich berücksichtigt werden muss.

Auf der anderen Seite konnte durch die Fehlerbaumanalyse gezeigt werden, dass die Vorkontrolle die Wahrscheinlichkeit für das Auftreten einer Kollision erhöht. Diese Ergebnis kann weder mittels Modellprüfung noch durch Simulation erhalten werden. Insbesondere konnte durch die Fehlerbaumanalyse eine Beurteilung des Gesamtsystem vorgenommen werden und es ergaben sich mehrere Verbesserungs- und Optimierungsvorschläge für das Design. So verringert z. B. eine zusätzliche Lichtschranke am Eingang der 4. Röhre erheblich die Anzahl der Fehlalarme bei einer sehr geringen Erhöhung des Kollisionsrisikos. Dieses Ergebnis schlägt sich auch in einem Arbeitspaket („Vergleichende Sicherheitsanalyse und Sicherheitsoptimierung“) für die nächste Projektphase nieder.

7 Stark gekoppelte formale FTA: Fallstudie „Pressure Tank“

Um die Tragfähigkeit formaler Sicherheitsanalyse und der zugehörigen Methodik zu Evaluieren, haben wir aus dem *Fault Tree Handbook* der *U.S. Nuclear Regulatory Commision* (Vesely *et al.*, 1981) das Beispiel eines *Pressure Tank* entnommen. An diesem Beispiel führen wir eine *stark gekoppelte* formale Sicherheitsanalyse (siehe Absch. 4) durch und untersuchen mit FMEA (siehe Absch. 5) die Auswirkungen eines Komponentenausfalls. Im Gegensatz zum vorigen Abschnitt, bei dem der informelle Fehlerbaum Sicherheitsanforderungen lieferte, werden hier die Ereignisse des Fehlerbaums formalisiert und die FTA-Vollständigkeitsbedingungen nachgewiesen. Dieses Vorgehen ist aufwendiger, liefert als Ergebnis aber einen

Fehlerbaum, der nachweisbar vollständig ist. Die Vollständigkeitsprüfung kann mit den Techniken aus Abschnitt 3 durch Modellprüfung geschehen, da das Modell des Pressure Tank zustandsendlich ist.

Als Modelchecker wurde RAVEN (Ruf, 2000; Ruf & Kropf, 1997) benutzt und den Pressure Tank als ein Zustandsübergangssystem modelliert. Da RAVEN *Clocked*-CTL (CCTL) (Ruf & Kropf, 1999) Formeln nachweisen kann, können auch in den Zustandsgraphen Zeitintervalle mit angegeben werden $(\psi, [n, m])$. Damit bedeutet ein Übergang, dass er nur stattfindet, wenn ψ i-Zeitschritte lange gilt, wobei $n \leq i \leq m$. Da Zeit im *Pressure Tank* Beispiel eine wichtige Rolle spielt, eignet sich diese Modellierungsmöglichkeit sehr gut, um das Beispiel zu beschreiben.

7.1 Beispiel: „Pressure Tank“

Im Pressure Tank Beispiel wird ein Versorgungstank beschrieben. Einmal gefüllt, liefert er solange Wasser, bis er leer ist und befüllt sich dann selbständig wieder. Die Tanksteuerung ist in Abbildung 7 zu sehen. Durch schließen des (Tipp-)Schalter S_1 wird der Versorgungstank

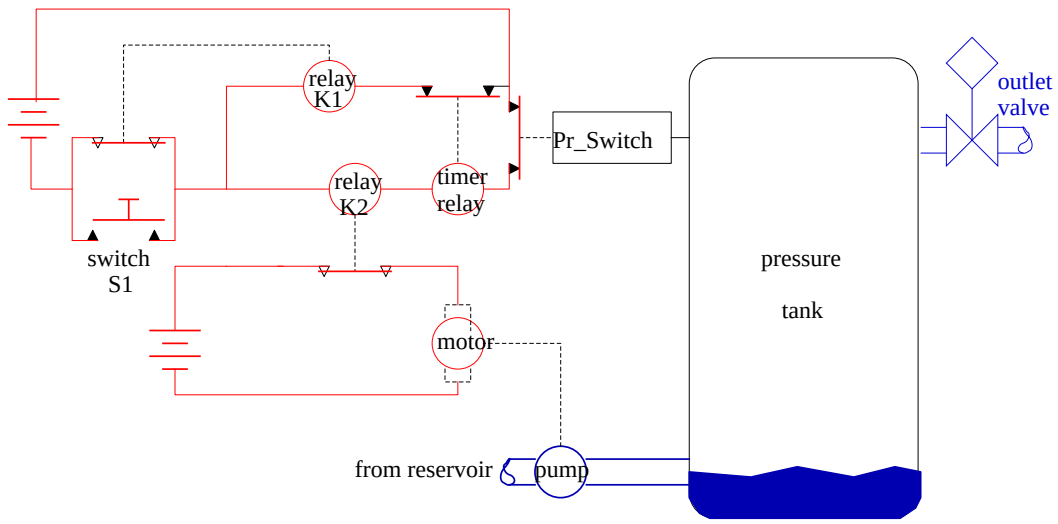


Abbildung 7: Pressure Tank

in Betrieb genommen. Über das Relais K_1 wird die Steuerung mit Strom versorgt, auch wenn S_1 nicht mehr gedrückt ist. Der Pumpenmotor ist über das Relais K_2 von der Steuerung galvanisch getrennten. Dieses Relais wird vom Steuerstromkreis mit Strom versorgt, bis der Sensor `pressure switch` signalisiert, dass der Tank voll ist, oder der `Timer` erkennt, dass der Tank „lange genug“ (60s) gefüllt wurde. Der Timer ist ein Schutzmechanismus, der bei Ausfall des Sensors verhindert, dass der Tank durch Überfüllen zum Platzen gebracht wird. Der Timer unterbricht den Steuerstromkreis komplett, indem er das Relais K_1 öffnet.

Damit bei korrektem Funktionieren des Sensors `pressure switch` mehrmaliges Befüllen des Tanks erreicht werden kann, wird der Timer rückgesetzt, wenn der `pressure switch` den Stromkreis unterbricht. Der Timer unterbricht damit den Steuerstromkreis nur, wenn der `pressure switch` defekt ist.

7.2 Vollständigkeitsprüfung der FTA

Nach der Modellierung und der Formalisierung des Fehlerbaums, der im *Fault Tree Handbook* schon vorgegeben ist, wurden die einzelnen Vollständigkeitsbedingungen nachgewiesen. Dies ist auch nach Beseitigung einiger Formalisierungs- und Modellierungsfehler gelungen. Danach wurde das Modell soweit erweitert, dass einzelne Komponenten auch ausfallen können.

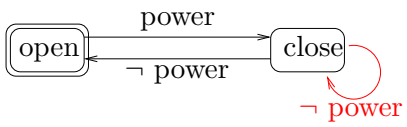


Abbildung 8: *Pressure Switch* mit Fehler

Die Vorgehensweise war, dass entsprechend dem Fehlerbaum die Ausfallmodi der einzelnen Komponenten betrachtet wurden und dafür zusätzliche Fehlerübergänge hinzugefügt wurden (siehe Abb. 8). Der betrachtete Fehlermodus ist, dass das Relais nicht öffnet, obwohl keine Strom mehr anliegt. Dies kann durch „Verkleben“ der Kontakte passieren. Dazu wurde im Zustand `close` der zusätzliche Übergang ¬ `power` hinzugefügt. Dadurch kann sich das Modell indeterministisch entscheiden, ob das Relais öffnet, wenn kein Strom anliegt, oder nicht (und damit fehlerhaft arbeitet).

Für alle Komponenten des *Pressure Tank* wurden die Fehlermodi, die im Fehlerbaum auftreten, im Modell berücksichtigt und zusätzlich wurden die in Abschnitt 3.2 beschriebenen Automaten in das Modell integriert, um die Ursache/Wirkungs-Beziehungen nachweisen zu können.

Um die Wirksamkeit unserer Methode zu testen, wurde eine Ursache – das Relais K_2 öffnet fälschlicherweise nicht – für das Platzen des Pressure Tank im Fehlerbaum entfernt. Versucht man nun, die entsprechende (nun unvollständige) Beweisverpflichtung nachzuweisen, schlägt der Beweis fehl. Der Modelchecker generiert ein Gegenbeispiel das beschreibt, welcher Ablauf diese Bedingung nicht erfüllt.

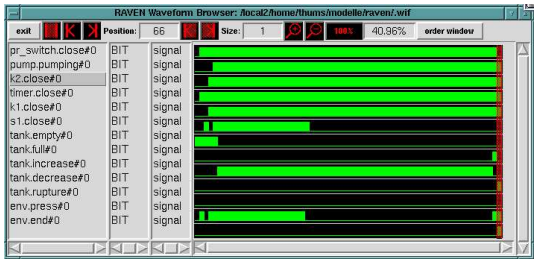


Abbildung 9: Gegenbeispiel

Abbildung 9 zeigt das entsprechende Gegenbeispiel, wie es von RAVEN dargestellt wird. Im Bild erkennt man links die verschiedenen Systemzustände. Ein Balken auf der rechten Seite bedeutet, dass der Zustand aktiv ist. Dieses Gegenbeispiel liefert Hinweise auf den Fehler: das Relais K_2 ist geschlossen ($K2.close$), obwohl kein Strom an den Kontakten anliegt. Dies ist eine zusätzliche Fehlerursache. Fügt man diese Ursache im Fehlerbaum hinzu, kann man die entsprechende, nun geänderte, Fehlerbaumbedingung nachweisen. So erweitert man nach und nach den Fehlerbaum, bis er vollständig ist. Schließlich wurde der Fehlerbaum als vollständig nachgewiesen.

7.3 Formale FMEA

Für die formale FMEA haben wir untersucht, wie sich das fehlerhafte Verhalten des *Pr_Switch* (Sensor am Tank, der den Füllzustand erkennt) auf das System auswirkt. Es wurde der Fehlermodus *fails open* betrachtet und versucht die folgende Aussage nachzuweisen: „wenn `full` eintritt und danach nicht `Pr_Switch_Open`, dann explodiert der Tank (`rupture`)“.

Die Verifikation der entsprechenden Bedingung (siehe Absch. 5) war nicht erfolgreich. Sie zeigt, dass dieser Fehler nur dann zu einem Platzen des Tanks führt, wenn gleichzeitig

der Eingangs-Schalter S_1 geschlossen ist. Falls S_1 nicht geschlossen, ist greift der Backup-Mechanismus des Timers. Es handelt sich also um ein fehlertolerantes System. `Pr_Switch` ist kein *single point of failure*.

Im Gegensatz dazu kann die FMEA-Bedingung für der Fehlermodus fails open des Relais K_2 nachgewiesen werden und zeigt damit, dass der alleinige Ausfall von K_2 zum Platzen des Tanks führen kann.

7.4 Ergebnisse

Die stark gekoppelte formale FTA ermöglicht es, Sicherheitsaussagen (das nicht-Eintreten des Top-Ereignisses im Fehlerbaum) für Systemmodelle zu machen, bei denen Komponentenausfälle mitmodelliert wurden. Der Vollständigkeitsbeweis garantiert, dass alle Ausfälle/Ursachen im Fehlerbaum berücksichtigt wurden.

Die FMEA unterstützt den Analytiker beim Feststellen, ob Schutzmechanismen ihre Aufgabe erfüllen, welche Ausfälle ungewünschte Wirkungen zeigen und welche Komponenten single points of failure sind. Beide Techniken helfen, ein tieferes Verständnis für die Funktionsweise des Systems zu bekommen. Die formale Sicherheitsanalyse eignet sich damit hervorragend für die sicherheitstechnische Untersuchung von Systemmodellen. Ein Papier zum Thema CTL-Modelchecking von FTA und FMEA am Beispiel „Pressure Tank“ befindet sich in Vorbereitung.

8 Formale FTA: Fallstudie „prellfreier Umschalter“

Im Rahmen einer Zusammenarbeit mit der DaimlerChrysler AG Esslingen wurde eine Sicherheitsanalyse eines „prellfreien Umschalters“ durchgeführt. Ausgangspunkt dieser Fallstudie war eine Fehlerbaumanalyse der DaimlerChrysler AG dieser Komponente. Bei dieser Analyse waren den Autoren einige Schwierigkeiten aufgefallen (siehe unten). Ziel der Fallstudie war die Beseitigung der bestehenden Probleme und die Erstellung einer vollständigen Fehlerbaumanalyse unter Anwendung der von uns entwickelten Techniken. Bei der Durchführung dieser Studie stellte sich jedoch heraus, dass die Fehlerbaumanalyse eine ungeeignete Technik für die hier auftretende Problemstellung ist. Das führte zu der interessanten Frage, wann welche Technik für welche Fragestellungen geeignet ist. Zwar sieht man, dass der prellfreie Umschalter ein Beispiel für eine ganze Klasse von Systemen ist, bei denen die FTA an ihre Grenzen stößt, eine systematische Klassifizierung, in welchen Anwendungen ähnliche Phänomene auftreten, existiert aber noch nicht. Im Folgenden wollen wir unsere Beobachtung kurz beschreiben.

Der prellfreier Umschalter ist eine Komponente, die möglichst kontinuierlich – prellfrei –, innerhalb eines gewissen Zeitintervalls T von einem Eingangssignal U1 auf ein anderes Eingangssignal U2 umschaltet. Zusätzlich hat der Umschalter noch weitere Eingänge wie etwa die Richtung der Umschaltung, die kleinst möglichen Umschaltintervalle oder Reset- und Aktivierungssignal. Obwohl der Umschalter von seinem Funktionsumfang her eher einfach erscheint, stellt er eine sehr sicherheitskritische Komponente dar. Das liegt daran, dass solche Umschalter in vielen komplexen Systemen (z.B. im Automobil im ABS) vorkommen und das fehlerfreie Funktionieren dieser Systeme entscheidend von diesen Komponenten abhängt.

Eine Beschreibung des prellfreien Umschalter lag in Mathlab-Simulink Notation vor. Dieses Modell bildete die Grundlage für die Fehlerbaumanalyse. Als top event war die Frage nach einem Sprung in der Ausgabe gegeben. Hier stellten sich schon die ersten Probleme. Die informelle Beschreibung des top events ist bei weitem nicht ausreichend für eine formale Analyse.

Insbesondere die Definition von „Sprung“ ist ungenügend in Hinblick auf die Abhängigkeit von den Eingabewerten. Hier muss spezifiziert werden, wie die Definition des top events von den zeitlichen Schwankungen der Eingabewerte abhängig ist. Das heißt dann aber auch, dass der Fehlerbaum von der zeitlichen Entwicklung der Umgebungsparameter abhängig ist. Dieses Problem kann noch gelöst werden, indem man postuliert, dass die Definition des top events nur von den Umgebungsparametern zu einem beliebigen – aber festen – Zeitpunkt abhängt.

Bei genauerer Analyse stellte sich aber noch ein ganz anderes Problem. Der prellfreie Umschalter realisiert eine mathematische Formel. Insbesondere besteht er aus einer Reihe von Additions- und Multiplikationsoperatoren. Die Definition des top events stellt eine Bedingung an den Wert der Ausgabe – hier das Ergebnis einer Reihe von Rechenoperationen – dar. Wenn die Ausgabe einer Komponente fehlerhaft ist, so kann a priori keine Aussage darüber gemacht werden ob dies zu einem Fehler in der Ausgabe der nächsten Komponente führt oder nicht. So kann zum Beispiel eine Abweichung in einer Eingabe einer Additionskomponente zu einem Fehler in der Ausgabe führen oder auch nicht – abhängig von der zweiten Eingabe. Umgekehrt kann das Ergebnis einen Fehler darstellen (z. B. Abweichung größer x), ohne dass die Eingabewerte über die Toleranzgrenze hinaus fehlerbehaftet sind (fehlerbehaftet bedeutet hier Abweichung größer y). Für eine Fehlerbaumanalyse ist allerdings eine der zentralen Forderungen, dass alle auftretenden Knoten Fehler darstellen. Noch schwieriger wird eine Vorhersage, wenn Multiplikation betrachtet wird. Hier hängt die Größe des Fehlers in der Ausgabe nicht nur von den Fehlern in der Eingabe, sondern auch noch von den korrekten Werten der Eingaben ab.

Aufgrund dieser Zusammenhänge ist eine Fehlerbaumanalyse für den prellfreien Umschalter und das top event „Sprung in der Ausgabe“ ungeeignet und nutzlos. Generell kann man dies für alle Systeme erwarten, die eine Berechnungsfunktion mit verschiedenen Gattern (Integrator-, Multiplikator-, ...) beschreiben. Dieses Ergebnis überraschte uns etwas, wirft aber die interessante Fragestellungen auf: Wie, wann und unter welchen Umstände ist welche Analysetechnik einsetzbar und geeignet. Kriterien, die im Voraus Auskunft über die Anwendbarkeit von Sicherheitsanalysetechniken geben können, sind äußerst wichtig und hilfreich.

9 Fazit

In diesem Bericht wurden die Grundlagen formaler Sicherheitsanalyse und entsprechende Werkzeugunterstützung beschrieben. Neben der Beweisunterstützung für Statecharts in KIV, um dynamische, nicht zustandsendliche Systeme zu analysieren, wurde eine FTA Formalisierung in CTL für die automatische Modellprüfung von FTA-Bedingungen angegeben und eine weitere Sicherheitsanalysetechnik (FMEA) formalisiert. An verschiedenen Fallstudien wurde die formale Sicherheitsanalyse durchgeführt und die entwickelte Anwendungsmethodik evaluiert.

Es hat sich gezeigt, dass sich durch die formale Sicherheitsanalyse ein Zugewinn an Sicherheit gegenüber den Einzelmethoden Sicherheitsanalyse bzw. formale Methoden ergibt. Die zuletzt beschriebene Fallstudie zeigt aber auch die Grenzen der formalen FTA auf, da sie für dieses Problem nicht die geeignete Analysemethode darstellt. Deshalb möchten wir in Zukunft untersuchen, ob es a priori Eigenschaften von Problemen gibt, an denen sich die Eignung entsprechender Analysemethoden festmachen lassen. Dann kann ein Katalog erstellt werden, in dem vor der Sicherheitsanalyse die am besten geeignete Methode ausgewählt wird. Mit der Integration eines FTA Werkzeugs in KIV erhalten wir dann durchgängige, forma-

le Sicherheitsanalysemethoden für sicherheitskritische Systeme, die mit der entsprechenden Werkzeugunterstützung durchgeführt werden kann.

References

Balser, M., & Thums, A. 2002. Interactive Verification of Statecharts. *In: Integration of Software Specification Techniques (INT'02).*

Balser, M., Reif, W., Schellhorn, G., Stenzel, K., & Thums, A. 2000. Formal System Development with KIV. *In:* Maibaum, T. (ed), *Fundamental Approaches to Software Engineering.* LNCS, no. 1783. Springer.

Balser, M., Duelli, C., Reif, W., & Schellhorn, G. 2002. Verifying Concurrent Systems with Symbolic Execution. *Journal of Logic and Computation*, **12**(4).

Biundo, S., Dengler, D., & Koehler, J. 1992. Deductive planning and plan reuse in a command language environment. *Pages 628–632 of: Proceedings of the 10th European Conference on Artificial Intelligence.*

Cichicki, T., & Górski, J. 2000. Failure Mode and Effect Analysis for Safety-Critical Systems with Software Components. *In:* Koomneff, F., & van der Meulen, M. (eds), *SAFECOMP 2000.* LNCS, vol. 1943. Springer-Verlag Berlin Heidelberg.

Cichicki, T., & Górski, J. 2001. Formal Support for Fault Modelling and Analysis. *In:* Voges, U. (ed), *SAFECOMP 2001.* LNCS, vol. 2187. Springer-Verlag Berlin Heidelberg 2001.

Damm, W., Josko, B., Hungar, H., & Pnueli, A. 1998. A Compositional Real-time Semantics of STATEMATE Designs. *Pages 186–238 of:* de Roever, W.-P., Langmaack, H., & Pnueli, A. (eds), *COMPOS' 97.* LNCS, vol. 1536. Springer-Verlag Berlin Heidelberg.

Harel, D. 1984. Dynamic Logic. *Pages 496–604 of:* Gabbay, D., & Guenther, F. (eds), *Handbook of Philosophical Logic*, vol. 2. Reidel.

Harel, D., & Naamad, A. 1996. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, **5**(4), 293–333.

Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., & Trakhtenbrot, M. 1990. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, **16**(4).

Heisel, M., Reif, W., & Stephan, W. 1988. Program Verification Using Dynamic Logic. *In:* Börger, E., Büning, H. Kleine, & Richter, M. (eds), *1st Workshop on Computer Science Logic. Proceedings.* Springer LNCS 329.

Jansen, L., & Schnieder, E. 1999. *Referenzfallstudie Bahnübergang – Referenzfallstudie im Bereich Verkehrsleittechnik des DFG-SPP Softwarespezifikation.* Tech. rept. Institut für Regelungs- und Automatisierungstechnik, http://www.ifra.ing.tu-bs.de/m33/spezi/. in German.

Klose, J., & Thums, A. 2002. *The STATEMATE Reference Model of the Reference Case Study 'Verkehrsleittechnik'.* Tech. rept. 2002-01. Universität Augsburg.

Lamport, L. 1994. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, **16**(3).

Leveson, N. 1995. *Safeware: System Safety and Computers.* Addison Wesley.

Manna, Z., & Pnueli, A. 1995. *Temporal Verification of Reactive Systems—Safety.* Springer-Verlag.

Manna, Z., & the STeP group. 1994 (July). *STeP: The Stanford Temporal Prover.* Tech. rept. Computer Science Department, Stanford University.

Moszkowski, B. 1985. A Temporal Logic for Multilevel Reasoning about Hardware. *IEEE Computer*, **18**(2), 10–19.

Ortmeier, F., Reif, W., Schellhorn, G., Thums, A., Hering, B., & Trappschuh, H. 2002. Safety Analysis of the Height Control System for the Elbtunnel. *Pages 296 – 308 of: Proceedings SAFECOMP 2002.* Catania, Italy: Springer LNCS 2434.

Pnueli, A., & Shalev, M. 1991. What is in a Step: On the Semantics of Statecharts. *Pages 244–264 of: Symposium on Theoretical Aspects of Computer Software.*

Reif, W. 1999. Formale Methoden für sicherheitskritische Software – Der KIV-Ansatz. *Informatik – Forschung und Entwicklung*, **14**(3).

Reif, W., Schellhorn, G., & Thums, A. 2000a. Formale Sicherheitsanalyse einer funkbasierten Bahnübergangsteuerung. *In:* Schnieder, E. (ed), *Forms2000 – Formale Techniken für die Eisenbahnsicherung.* Fortschritt-Bericht VDI, vol. Reihe 12, Nr. 441.

Reif, W., Schellhorn, G., & Thums, A. 2000b (June). Safety Analysis of a Radio-Based Crossing Control System Using Formal Methods. *In:* Schnieder, E., & Becker, U. (eds), *9th IFAC Symposium Control in Transportation Systems 2000.*

Ruf, J. 2000 (January). *RAVEN: Real-Time Analyzing and Verification Environment.* Technical Report WSI 2000-3, University of Tübingen, Wilhelm-Schickard-Institute.

Ruf, J., & Kropf, T. 1999. Modeling Real-Time Systems with I/O-Interval Structures. *Pages 91–100 of: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen.* Shaker Verlag.

Ruf, Jürgen, & Kropf, Thomas. 1997. Symbolic Model Checking for a Discrete Clocked Temporal Logic with Intervals. *Pages 146–166 of:* Cerny, E., & Probst, D.K. (eds), *Conference on Correct Hardware Design and Verification Methods (CHARME).* Montreal: Chapman and Hall, for IFIP WG 10.5.

Schäfer, Andreas. 2001. *Fehlerbaumanalyse und Model-Checking.* M.Phil. thesis, Universität Oldenburg. in German.

Schellhorn, G., Thums, A., & Reif, W. 2002. Formal Fault Tree Semantics. *In: Proceedings of The Sixth World Conference on Integrated Design & Process Technology.*

Sperschneider, V., & Antoniou, G. 1991a. *Logic: A Foundation for Computer Science*. Addison Wesley.

Sperschneider, V., & Antoniou, G. 1991b. *Logic: A Foundation for Computer Science*. Addison Wesley.

Tapken, J. 2001 (June). *Model-Checking of Duration Calculus Specifications*. M.Phil. thesis, University of Oldenburg. http://semantik.informatik.uni-oldenburg.de/projects/.

Vesely, W. E., Goldberg, F. F., Roberts, N. H., & Haasl, D. F. 1981. *Fault Tree Handbook*. Washington, D.C. NUREG-0492.