Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Eine Stochastische Testmethodik für den Robustheitstest Automobiler Steuergeräte

Hans-Werner Wiesbrock[1]

Abstract: Heute laufen vielzählige Applikationen auf einem Automobilen Steuergerät, greifen auf gemeinsame Speicher und Sensorik zurück und steuern unterschiedlichste Aktoren an. Sie in ihren Wechselwirkungen und zeitlichen Abhängigkeiten untereinander zu testen ist aufwendig, kostenintensiv und schwierig. Ein neuer Ansatz soll hier helfen.

Im Stochastischen Robustheitstest werden Testdaten zur Eingabe in den Prüfling über parallele Markov Automaten generiert. Der Testlauf selber wird auf die Einhaltung allgemeiner Eigenschaften wie: Blinker müssen stets vorn und hinten synchron leuchten, überprüft, eine Verletzung als Fehler protokolliert. Eine spezielle Abfragesprache, die auch erlaubte zeitliche Verzögerungen in den Erwartungen erfassen kann, unterstützt den Tester bei seiner Formulierung der Regeln. Aus dem Ansatz heraus leitet sich in natürlicher Weise auch eine Klasse von Testende Kriterien ab.

Explorativ entwickelt und gemeinsam erprobt wurde dieser Ansatz parallel zur Entwicklung eines Zentralen Bordnetz Steuergerätes bei einem großen Automobil Hersteller.

Keywords: Stochastische Testdaten Generierung, Markov Automaten, Regelbasierte Auswertung, Test der Robustheit

[1] ITPower Solutions GmbH, Kolonnenstraße 26, 10829 Berlin, hans-werner.wiesbrock@itpower.de

1 Einleitung

Zahlreiche Anwendungen laufen heute auf einem Automobilen Steuergerät und viele von ihnen interagieren asynchron über Bus Kommunikationen mit anderen Steuergeräten. So sind bspw. auf einem Zentralen Bordnetz Steuergerät Wischer, Außenlicht, Zentralverriegelung und viele andere Funktionalitäten implementiert und über mehrere CAN und LIN Busse mit anderen ECUs verbunden. Ungünstige Zeitabläufe und Nutzung gemeinsamer Ressourcen können dazu führen, dass bei gleichzeitiger Betätigung des Heckrollos und Startens des Motors das Abblendlicht kurz blinkt, der Scheibenwischer zuckt oder andere unerwünschte Seiteneffekte auftreten. Wie können wir im Test absichern, dass die vielfältigen Applikationen sich nicht gegenseitig beeinträchtigen, dass es nicht zu sporadischen Ausfällen und Fehlfunktionen in ihrer Interaktion kommt? Dass zeitliche Jitter, die zu einer merklichen Verzögerung oder gar zum Ausfall führen, nicht erst vom Kunden, sondern im Vorfeld entdeckt werden können?

1.1 Test einzelner Applikationen

Die Hersteller/Zulieferer von Steuergeräten und ihren Anwendungen haben zumeist die Hardware für sich und auch die einzelnen Funktionen umfangreich getestet. Wie die Integration sich im Verbund mit anderen Steuergeräten oder auch das komplexe Zusammenspiel vieler gleichzeitiger Anwendungen verhält kann z.T. von Ihnen nicht oder nur unzureichend getestet werden. Das liegt nicht zuletzt an dem hohen Aufwand, geeignete Testszenarien zu entwickeln und an der schwierigen Auswertung solcher Testläufe. Was ist ein Fehlverhalten, wenn vieles zugleich angesteuert wird?

1.2 Qualitätssicherung nicht-funktionaler Eigenschaften

Auch die verschiedenen Sicherheitsnormen und Qualitätsstandards fordern von heutigen Steuergeräten im Automobil, der Eisenbahn oder anderen kritischen Bereichen u.a. ihre Robustheit. In der Standard Literatur wird Robustheit bestimmt als Eigenschaft, auch in ungewöhnlichen Situationen definiert zu arbeiten und sinnvolle Reaktionen durchzuführen, [Li02]. Meist versucht man, konstruktiv dieser Forderung zu begegnen, d.h., durch Verwendung erprobter Architekturen und Designprinzipien ein robustes System zu entwickeln. Eine Teilfrage ist hier: Funktioniert mein System definiert auch bei allen möglichen Eingaben mit zeitlichen Ungewissheiten? Model Checking und andere Verifikationstechnologien versuchen, durch erschöpfende Analyse des Testeingaberaumes diese Frage zu beantworten. Doch ist dieser hoch dimensionale Raum bei realen Steuergeräten und ihren Implementierungen nicht mehr zu explorieren und die Frage nach möglicher Testunterstützung taucht auf. Auf diese Fragen versucht eine neue Testmethodik, der Stochastische Robustheitstest, eine Antwort zu geben.

1.3 Lösungsansatz

Aus den obigen Anmerkungen leiten sich zwei vordringliche Probleme ab:

- Wie erhält man mit möglichst geringem Aufwand die nötigen Testeingaben?
- Wie beurteilt man sinnvoll den Testausgang?

Auf diese beiden Fragen versucht der Stochastische Robustheitstest (SRT) Antworten zu geben. Ausgehend von Benutzer Profilen werden parallele Markov Automaten definiert, die die Testeingaben für den Prüfling erzeugen, s. auch [AL15],[se15]. Da die einzelnen Funktionen im Allgemeinen bereits gut getestet werden, siehe 1.1, ist das mögliche Fehlverhalten weniger in komplexen Funktionsabläufen zu suchen, als in einfachen Konflikten oder Verletzungen allgemein gewünschter Erwartungen. Dazu wird eine speziell für diesen Anwendungsfall entwickelte Abfragesprache vorgestellt die auch erlaubte zeitliche Verzögerungen in den Erwartungen erfassen kann. Dies ist nötig, da asynchrone Kommunikation definierte zeitliche Latenzen gestattet. Grundlegende Ideen hierzu findet man bereits in [Im07] . Im folgenden Kapitel 2 wird zunächst gezeigt, wie man beliebig viele sinnvolle Testeingaben einfach erhält. Dazu kann auf Standard Techniken (Markov Automaten) zurück gegriffen werden, die kurz eingeführt und anschließend leicht erweitert werden. Im Kapitel 3 wird dann eine Abfragesprache zur Beschreibung der Allgemeinen Erwartungen an die Testläufe und deren Beurteilung definiert, ihre theoretischen Grundlagen skizziert, sowie ihre Anwendung in Beispielen gezeigt. Ziel dieser Arbeit ist es, eine auf stochastischen Methoden beruhende Testmethode vorzustellen, die wechselseitig-bedingte Störungen effektiv aufspüren kann. Zugleich wird eine Werkzeugunterstützung gezeigt, mit der diese Methode vollautomatisch umgesetzt wurde.

Anmerkung Die mit dem SRT möglichen Prüfungen liefern keinen Ersatz für die weiterhin notwendigen umfassenden funktionalen Tests, sondern sind komplementär als Ergänzung zu den bereits bestehenden Testfällen und Testvorgehen konzipiert.

2 Markov Automaten

Nach einer kurzen Motivation wird in diesem Kapitel die Theorie der Markov Automaten skizziert und ihre spezielle Anwendung im **SRT** beschrieben.

2.1 Problemstellung

Eine zentrale Aufgabe des Tests einzelner Applikationen ist die Erstellung geeigneter Testszenarien und Eingabewerte, für Gutfälle als auch für Negativtests. Ausgehend von den Anforderungen werden zumeist typische Anwendungsfälle geprüft, Varianten von ihnen sowie einige Extreme. Im systematischen Test werden eventuell noch Code Überdeckungskriterien gemessen und ergänzende Tests abgeleitet.

Soll hingegen die gleichzeitige Ansteuerung verschiedenster Applikationen mit ihrer Kombinatorik systematisch getestet werden, so ist diese Aufgabe ungleich schwerer. Es gibt meist keine Anforderungen dazu oder nur sehr allgemein gehaltene. Lasten- und Pflichtenhefte sind häufig nach einzelnen Funktionen oder Featuren aufgeteilt und ihr Zusammenspiel nur in 'mit geltenden Unterlagen' festgehalten. Welche Kombinationen sollte man testen? Welche parallelen Szenarien? Ein rein zufälliger Test, naiv stochastische Eingaben, führen zu überwiegend inkonsistenten Werten, die bei geringster Plausibilisierung im Prüfling durch ihn bereits verworfen werden. Um zu sinnvollen, interessanten Eingaben zu kommen, sollte man sich an Benutzerprofilen orientieren. Dazu gibt es eine etablierte Technik, die Markov Automaten.

2.2 Definition von Markov Automaten

Automobile Steuergeräte sind spezielle Eingebettete Systeme und so gilt auch für sie, dass Eingabewerte in der Regel über die Zeit sequenzierte Werte sind. So ist die stochastische Erzeugung von Testeingaben im engeren Sinn Resultat eines stochastischen Prozesses. Wir wählen hierzu eine spezielle Form : Die Markov-Kette. Ohne weiter auf ihre genaue Definition [WS13, Oc05] einzugehen soll hier eine einfache Konstruktion beschrieben werden, nämlich über Markov Automaten. Dies sind endliche Automaten, deren Zustandsübergänge über Wahrscheinlichkeiten gesteuert werden.

Beispiel: Stochastische Generierung von Zündschlüsselstellungen

In diesem Beispiel werden vier verschiedene Zündschlüsselstellungen betrachtet und definiert, mit welcher Wahrscheinlichkeit die Übergänge stattfinden. Angenommen, der

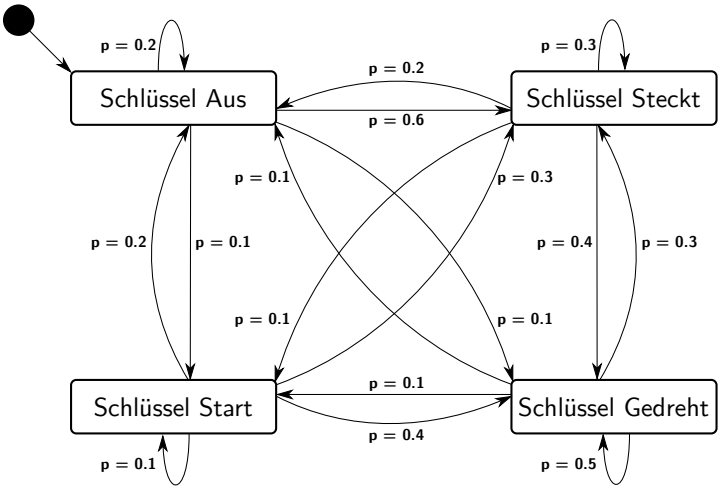

Abb. 1: Testdatenautomat: Zündung

Schlüssel befinde sich in der 'Aus'-Stellung. Dann soll bei jedem 10. mal der Schlüssel anschließend in die 'Gedreht' Stellung übergehen: *Schlüssel Aus* $\overset{[p=0.1]}{\rightarrow}$ *Schlüssel Gedreht*.

Einfacher zu beschreiben ist ein solcher Automat durch eine sogenannte Übergangs- oder Transitions-Matrix.

State	Aus	Steckt	Gedreht	Start
Aus	0,2	0,2	0,1	0,2
Steckt	0,6	0,3	0,3	0,3
Gedreht	0,1	0,4	0,5	0,4
Start	0,1	0,1	0,1	0,1

Tab. 1: Übergangswahrscheinlichkeiten

Gegeben solch ein Markov Automat 'würfelt' man jetzt bspw. alle 100 [ms] (Periode) und wechselt entsprechend der 'Würfelzahl' und der angegeben Wahrscheinlichkeit der Transition von einem Zustand in den nächsten. Als Ergebnis erhält man einen stochastischen Prozess, der nach und nach verschiedene Zustände des Automaten durchläuft. Technisch realisiert wird ein solcher Automat durch Pseudozufallsvariablen unter Verwendung eines Seed Wertes, um auf diese Weise reproduzierbare Läufe zu erhalten.

2.3 Test Semantik der Wahrscheinlichkeiten

Ohne zu sehr auf die mathematischen Nachweise hier einzugehen, soll kurz erläutert werden, wie die Übergangswahrscheinlichkeiten aus Testersicht zu lesen sind und wie geeignete Werte ermittelt werden können. Durch eine Rückführung auf ein Bernoulli Experiment lässt sich leicht ermitteln, wie groß die durchschnittliche Verweildauer in einem Zustand ist, in diesem Beispiel:

State	Aus	Steckt	Gedreht	Start
Mittlere Verweildauer	1,25	1,43	2	1,1

Tab. 2: Mittlere Verweildauer in [Periode] Einheiten

Die Verhältnisse der Übergangswahrscheinlichkeiten zu anderen Zuständen hingegen steuern, wie häufig ein spezieller Folgezustand gewählt wird:

Schlüssel Aus $\overset{[p=0.2]}{\rightarrow}$ *Schlüssel Steckt* und *Schlüssel Aus* $\overset{[p=0.1]}{\rightarrow}$ *Schlüssel Gedreht*

besagt, dass doppelt so häufig von 'Aus' nach 'Steckt' gegangen wird als nach 'Gedreht'.

Umgekehrt gilt aber auch folgendes. Gegeben seien alle relativen Übergangszahlen zu anderen Zuständen, im Beispiel:

Schlüssel Aus$\rightarrow$ *Schlüssel Steckt* doppelt so häufig wie *Schlüssel Aus*$\rightarrow$ *Schlüssel Gedreht*

und ebenso für die anderen Zustandsübergänge von *Schlüssel Gedreht*$\rightarrow$ *Schlüssel Start* et cetera. Außerdem seien die gewünschten mittleren Verweildauern in Einheiten der 'Würfel' Periode bekannt. Dann ist dadurch eindeutig die gesamte Übergangsmatrix bestimmt.

Wir können also aus Benutzerprofilen für die einzelnen Applikationen leicht die entsprechenden Transistionsmatrizen ableiten!

Die Matrixelemente (Übergangswahrscheinlichkeiten) sind alle positiv, der Eigenvektor zum maximalen Eigenwert ist also eindeutig (Perron Frobenius) und lässt sich einfach berechnen. Normiert ergibt dieser die mittleren Aktivierungen der einzelnen Zustände, also wie häufig in Relation zu den anderen, ein Zustand im durchschnittlichen Testlauf aktiv war.

2.4 Stochastische Generierung bei kontinuierlichen Testdaten

Das in 2.2 beschriebene Vorgehen funktioniert sehr gut, wenn die möglichen Eingabedaten aufzählwertig sind wie z.B. Schlüssel, Wischerhebel und Lichdrehschalter Stellungen. Im Falle kontinuierlicher Eingabedaten wie der Geschwindigkeit muss dieser Ansatz erweitert werden. Dazu kann man nach der Klassifikationsbaummethode vorgehen, [GG93]. Nach dieser werden die möglichen Eingabewerte in Äquivalenzklassen eingeteilt. Die zugrundeliegende Relation wird bestimmt durch die Uniformitätshypothese, d.h., es werden Eingabedaten als annähernd gleich klassifiziert, wenn sich die Applikation bei ihrer Eingabe voraussichtlich annähernd gleich verhält. Der kontinuierliche Wertebereich wird auf diese Weise in eine endliche Zahl von Klassen ($\rightarrow$ Zuständen im Markov Automat) partitioniert. Damit ist man in einer ähnlichen Situation wie im Falle enumerationswertiger Eingaben. Doch sind hier die Transitionen etwas allgemeiner zu definieren.

Beispiel: Stochastische Generierung von Geschwindigkeitseingaben

Sei die Geschwindigkeit eines Fahrzeuges ein Eingabedatum einer Applikation. Gemäß der Klassifikationsbaum Methode identifiziert man verschiedene Geschwindigkeitsbereiche [0-5] km/h, [5-50] km/h und [50-150] km/h. Nach Kapitel 2.3 wird eine Übergangsmatrix für die Bereiche definiert, siehe linke Seite in Abb. 2.

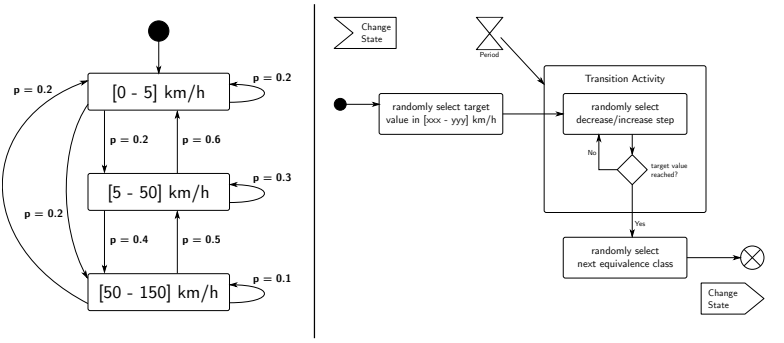

Abb. 2: Vorgehen bei kontinuierlichen Daten

In Erweiterung zum Vorgehen in Kapitel 2.2, wählt man zunächst zufällig einen Zielwert in der nächsten Äquivalenzklasse, z.B. 37 km/h $\in$ [5,50] km/h. Gegeben sei nun für jede Äquivalenzklassen Transition eine Schrittweite und eine Wahrscheinlichkeit, z.B. 0.5

km/h, und p = 0.1. Dann würfelt man mit Wahrscheinlichkeit p erneut, ob zur aktuellen Größe die Schrittweite hinzu addiert, oder subtrahiert wird. Man erhält so einen Random Walk mit definierter Schrittweite, der abhängig von der Wahrscheinlichkeitsangabe schneller oder langsamer den Zielwert erreicht. Ist er erreicht wird erneut ein Zielwert ausgewürfelt und das Verfahren beginnt von vorne.

Allgemein kann man so zu kontinuierlichen Eingabegrößen mit Hilfe der Klassifikationsbaummethode geeignete Markov Automaten assoziieren und entsprechende Übergänge modellieren.

Auf diese Weise lassen sich dezidiert die Testdaten für beliebig viele Applikationen über parallel laufende Markov Automaten stochastisch definieren und generieren.

2.5 Überdeckungskriterium

Bei dieser Testmethode werden die Testdaten wie in 2.2 und 2.4 beschrieben über spezielle Zustandsautomaten (Markov Automaten) generiert. Ziel des **SRT** ist es, gegenseitige Störeinflüsse auf dem Steuergerät zu entdecken. Insbesondere sollten deshalb in den Tests vielfältige Ansteuerungskombinationen durchlaufen werden. Als Kriterium für die Testtiefe bietet sich an, die Anzahl der unterschiedlichen Kombinationen in den Ansteuerungen zu verwenden. Dazu werden folgende Überdeckungsmaße vom Grad $k \in \mathbb{N}$ vorgeschlagen:

$$CovM(k) \stackrel{def}{=} \frac{k!(n-k)!}{n!} \sum_{n-tuple(S_1,S_2,..,S_k)} \frac{\sharp\text{activated Substates in } (S_1 \times \cdots \times S_k)}{\sharp\text{Substates in } (S_1) \cdots \sharp\text{Substates in } (S_k)}$$

mit S_i - Markov Automat, $\sharp(S_i)$ - Anzahl der Zustände in S_i, $S_1 \times S_2$ - Produkt Automat, n-Anzahl aller Markov Automaten.

Beispiel: CovM(2)

Zu jedem Paar von Testdatengeneratoren wird festgehalten, welche ihrer Zustandskombinationen in den Testläufen angenommen wurden. Die Anzahl aller Möglichkeiten für ein Paar ist offensichtlich gleich dem Produkt aus der Anzahl ihrer Unterzustände. Es wird nun die Summe aller durchlaufenen Zustandskombinationen, gemittelt durch die Anzahl aller möglichen, gebildet. $\frac{n!}{2!(n-2)!}$ *ist die Anzahl aller Paare. Das Überdeckungsmaß CovM(2) wird also durch den Divisor auf 1 normiert. Ist CovM(2) = 1, so ist zu zwei beliebigen Eingabeautomaten jede der wechselseitigen Eingabekombinationen mindestens einmal durchlaufen worden.*

Abhängig von der gewünschten Testtiefe im **SRT** lassen sich dann über die Wahl des k in CovM(k) geeignete zu erreichende Kennzahlen ableiten.

3 Überprüfung allgemeiner Eigenschaften

Wenden wir uns der zweiten Frage zu: Wie beurteilt man sinnvoll den Testausgang?

Bei allgemeinen, stochastischen Eingaben ist nicht offensichtlich, wie der Testausgang beurteilt werden soll. Die Erfahrung zeigt jedoch, dass viele der im Feldversuch aufgetretenen Fehler nicht komplexe Reaktionen auf spezifische Eingaben waren, sondern in der Regel die Verletzung einfach zu überprüfender, allgemeiner Eigenschaften wie z.B. mangelnde Synchronisation der Blinkfunktionen, fehlerhafte, sporadische Ansteuerung verschiedener Leuchten. Der Aufwand, solche Fehler frühzeitig zu entdecken, liegt demnach nicht so sehr in der Spezifikation eines ausgefeilten Testorakels, sondern in der geeigneten komplexen Ansteuerung des Systems mit der parallelen Überwachung auf die Einhaltung allgemeiner Eigenschaften! Es sind Regeln wie:

- Die beiden diskreten Ausgänge zu den Blinkleuchten müssen synchron angesteuert werden! (erlaubte Verzögerung 160 [ms])
- Die Rückfahrlichter dürfen nur bei Zündung an und Rückwärtsgang leuchten! (erlaubte Verzögerung 400 [ms])
- Nebelleuchten dürfen nicht an sein, wenn Lichtdrehschalter nicht auf Nebellicht oder Nullstellung oder AFL steht! (erlaubte Verzögerung 100 [ms])

Zu berücksichtigen ist hierbei, dass bedingt durch zwischengeschaltete Messkaskaden, CAN Bus Kommunikation usw. systematische Zeit- und Wertejitter bei den Messwerten und Ansteuerungen zu erwarten sind. Die Regeln müssen also 'bis auf definierte Zeitverzögerungen und Wertedifferenzen', sogenannte Latenzen, eingehalten werden.

3.1 Berücksichtigung zeitlicher und amplitudenmäßiger Latenzen

Betrachten wir zunächst die Forderung nach synchronem vorderen und hinterem Blinken und ihre Überprüfung im Hardware-in-the-Loop Testschrank oder im Fahrzeug. Um einen sinnvollen Vergleich von Signalen durchzuführen, sind gewisse Toleranzen in Zeit und Wert zu erlauben. (Unterhalb von 30 [ms] z.B. ist eine Abweichung der Synchronizität durch den Menschen nicht wahrnehmbar, siehe [P6] .) Kontrollierte Abweichungen in den Wertemessungen zu bewerten ist Standard und kann über verschiedene Wertvergleichskriterien abgedeckt werden. Problematisch ist jedoch die mögliche Verschiebung in der Zeit zu beurteilen. In [GW10] wird ein Konzept einer temporalen Logik mit Latenzzeiten eingeführt. In ihr sind innerhalb bestimmter Latenzzeiten keine eindeutigen Aussagen über wahr und falsch zu treffen. In dieser Arbeit wird auch gezeigt, wie dieser Ansatz zu bekannten Temporal Logiken in Relation steht, [MP92].

Ohne weiter auf diese Konzepte einzugehen, soll hier nur ein grundlegendes Konstrukt vorgestellt werden, die Implikation mit zeitlichen Latenzen:

Seien $\mathbb{A}(t)$ und $\mathbb{B}(t)$ zwei Flag Signale, d.h. boolwertige Funktionen über der Zeit t. Seien ferner Δ_F und Δ_P Latenzzeiten [ms]. Dann definieren wir für jeden Zeitpunkt t:

$$\begin{aligned} IMPLIES(\mathbb{A},\mathbb{B},\Delta_F,\Delta_P)\ (t) \quad &:\Leftrightarrow \\ &\mathbb{A}(t) = true \Rightarrow \exists t' \in [t-\Delta_F, t+\Delta_P] : \mathbb{B}(t') = true \wedge \\ &\mathbb{B}(t) = false \Rightarrow \exists t'' \in [t-\Delta_F, t+\Delta_P] : \mathbb{A}(t'') = false. \end{aligned}$$

Synchronizität mit Latenzen wird dann als zweifache Implikation definiert:

$$\begin{aligned} SYNCH(\mathbb{A},\mathbb{B},\Delta_F,\Delta_P)\ (t) \quad &:\Leftrightarrow \\ &IMPLIES(\mathbb{A},\mathbb{B},\Delta_F,\Delta_P)(t) \wedge IMPLIES(\mathbb{B},\mathbb{A},\Delta_F,\Delta_P)(t) \end{aligned}$$

Wählt man insbesondere $\Delta_P = 0$ resp. $\Delta_F = 0$ so lassen sich hierüber auch kausale Beziehungen wie: $\mathbb{A}$ verursacht $\mathbb{B}$, bzw. $\mathbb{B}$ muss $\mathbb{A}$ vorausgegangen sein, formulieren. Auch kann eine abgeschwächte Form der Transitivität für die Implikation und Synchronizität gezeigt werden, bei der die Latenzbereiche zu summieren sind.

Beispiel: Synchrones Blinken

Der vordere Blinker soll synchron zum hinteren Blinker leuchten. Kleine zeitliche Verschiebungen unterhalb von 50 ms können toleriert werden, da es die gleichzeitige Wahrnehmung nicht beeinträchtigt.

Bezeichne $U_V(t)$ die Spannungsansteuerung des vorderen Blinkers über die Testzeit, analog $U_H(t)$ für den hinteren Blinker. Die Latenzen sind dann $U(t) := [t-50ms, t+50ms]$ und die Regel lautet, dass zu allen Zeitpunkten

$$SYNCH((U_V(t) > 7{,}5V), (U_H(t) > 7{,}5V), 50, 50)(t) = true$$

gilt. Wir können diese Eigenschaft für jeden Testlauf jetzt algorithmisch überprüfen.

In [GW10] werden weitere Konstrukte aus den obigen Konzepten abgeleitet, über die Eigenschaften wie: Frühestens 30 s nach Ziehen des Schlüssels darf der CAN Bus zur Ruhe gehen, formuliert und überprüft werden können.

Konstrukt	**Beschreibung**
FOR_AT_LEAST($\mathbb{A};\Delta_t$)	Wenn $\mathbb{A}$ aktiv wird (=true), dann mindestens für eine Dauer Δ_t
	Beispiel: Mindestblinkdauer
HOLDS_AT_MOST($\mathbb{A};\Delta_t$)	Wenn $\mathbb{A}$ aktiv wird (=true), dann höchstens für eine Dauer Δ_t
	Beispiel: Maximale Blinkdauer
CHANGE_BEFORE($\mathbb{A}$**,incr;**Δ_t)	Nach einem Flagwechsel ($\mathbb{A} : false \rightarrow true$) für Δ_t =true
	Beispiel: Spätestens nach Zündungswechsel...
&& , || , !	Elementar Boolesche Verknüpfungen von Flagsignalen
	Beispiel: Zündung an und Wischerhebel betätigt...

Tab. 3: Weitere Sprachkonstrukte zur Formulierung von Regeln

3.1.1 Allgemeine Eigenschaften

Um nun Kriterien für die Beurteilung der Testausgänge zu formulieren, wird ein Messergebnis über einfache Vergleichsoperationen ($<, \leq, >, \geq, ==$) zu wahr oder falsch abstrahiert. Man kann sich so bei der Formulierung allgemeiner Regeln ohne Einbußen auf boolwertige Signale beschränken.

Man zeigt leicht, dass die in Tab. 3 angeführten Konstrukte und auch IMPLIES, resp. SYNCH die Kategorie der boolwertigen Signale über der Zeit nicht verlassen. Insbesondere kann man die verschiedenen Konstrukte ineinander verschachteln, komponieren! Auf diese Weise lassen sich beliebig komplizierte neue Boolesche Signale über der Zeit definieren.

Anmerkung: In Experimenten hat man eine definierte Startzeit des Testlaufs. Will man die Konstrukte auf Boolesche Signale über $[0, \infty[$ beschränken, ist noch das Problem der Randbedingungen zu beachten. Es soll aber an dieser Stelle nicht weiter darauf eingegangen werden.

Die Auswertung der stochastisch getriebenen Testläufe geschieht nun wie folgt. Es werden mit Hilfe der Messsignale eines Testlaufes geeignete Boolesche Signale definiert, sogenannte Regeln, die stets wahr sein sollen:

- SYNCHRON (bStBlinkerVREin, bStBlinkerHREin ; 160 [ms])
  Synchronität der beiden diskreten Ausgängen zu den Blinkleuchten. (erlaubte Verzögerung 160 [ms])
- IMPLIES((bKlemme15 && bRueckwaertsGang) , rRueckfahrlichtR ; 400 [ms])
  Die Rückfahrlichter können nur bei Zündung an und Rückwärtsgang leuchten. (erlaubte Verzögerung 400 [ms])
- IMPLIES(!(bStLdsNebellicht || bStLdsNullstellung || bStLdsAssistenzfahrlicht)) , rNebellichtR == 0 ; 400 [ms])
  Nebelleuchten nicht an, wenn Lichtdrehschalter nicht auf Nebellicht oder Nullstellung oder AFL. (erlaubte Verzögerung 400 [ms])

Diese Regeln (Boolschen Signale) werden nun parallel während des Testlaufes überwacht. Ein Fehlverhalten zur Zeit t ist dann gegeben, wenn das Signal zur Zeit t den Wert false annimmt.

Beispiel: Synchrone Ansteuerung der Blinkleuchten *Falls zu einem Zeitpunt t_0 gilt: SYNCHRON (bStBlinkerVREin, bStBlinkerHREin ; 160 [ms]) (t_0)=false, so laufen die Blinkleuchten nicht synchron!*

Aufgrund der Latenzen in der Zeit ist für eine Auswertung zu beachten, dass ggfs. der Zeitpunkt der Fehlerdetektion, im Beispiel: t_0, definiert verzögert zum eigentlichen Fehlverhalten sein kann!

In der explorativen Studie parallel zur Entwicklung eines Zentralen Bordnetz Steuergerätes konnte gezeigt werden, dass die Mächtigkeit der angebotenen Sprachkonstrukte hinreichend war, um die allgemeinen Regeln zur Zentralverriegelung, Außenlicht Ansteuerung etc. zu erfassen.

4 Realisierung

Umgesetzt wurde diese Testmethode mit Hilfe einer Werkzeug Implementierung in *C#*. Ausgehend von einer Übergangsmatrix, vorgegebenen Periode und Seed (Anfangsbedingung für die Erzeugung von Pseudo Random Variables) werden dann zyklisch stochastische Testdaten zu einem Markov Automaten generiert. Die Auswertungsalgorithmen basieren auf fest definierten zyklischen Erfassungszeiten der Messungen. Über eine geeignete GUI können die verschiedenen Aufgaben im Rahmen des **SRT** zentral bearbeitet werden.

Für jede Applikation lässt sich dann ein Markov Automat definieren. Die Zustände erlauben dann die Definition von Entry resp. During Aktionen, in denen beliebig viele Signale sequentiell gesetzt werden können, z.B. Zündung an: $Kl\_S = 1, Kl\_15 = 1, Kl\_X = 1, Kl\_50 = 0$; Die Übergangswahrscheinlichkeiten können dann direkt eingegeben oder durch relative Gewichtungen und Angabe von durchschnittlichen Verweildauern definiert werden.

Die Spezifikation von Regeln ist aufgrund der temporal-logischen Möglichkeiten nicht trivial und bedarf der Übung. Zur Vereinfachung wurden Muster Bibliotheken erstellt, so dass der Benutzer nur noch geeignete Parameter zu ersetzen hat. Auch wird ihm ein Editor angeboten, der die Syntax der Abfragesprache unterstützt. Trotzdem sollten bei der Formulierung **SRT** Experte und Anwendungsspezialist Hand in Hand arbeiten.

Um die Testdaten projektübergreifend wieder verwenden zu können, wurde eine Abstraktionsschicht implementiert, in der Signalbezeichner projektspezifisch angepasst werden können ohne die Definition der Automaten oder Regeln ändern zu müssen. Sie lassen sich dann mit logischen Signalbezeichnern formulieren. Da die Regeln und Ansteuerungen sehr allgemein sind, können so erfahrungsgemäß viele Artefakte projektübergreifend wiederverwendet werden!

Anbindung an eine Testumgebung

Die im **SRT** erstellten Testdatengenerator-Automaten setzen auch der COM Schnittstelle von CANoe auf. Über IO Karten (VT 1040, VT 2040 etc.) wird die Hardware angesteuert.

Für die Überwachung der Regeln wird die .Net Implementierung der Auswertungsalgorithmen als spezieller Simulationsknoten eingebunden. Änderungen in den Messdaten werden über CANoe Trigger an die Algorithmen weitergeleitet und direkt im Testlauf ausgewertet.

Für die Auswertung können die grafischen Möglichkeiten von CANoe genutzt werden, und die eingebaute Auswertungskomponente .

4.1 Erfahrungen und Ausblick

In der Erprobungsstudie wurde eine etwas abweichende Architektur verwendet, in der die verschiedenen Komponenten aus Matlab Simulink® generiert und als CANoe Simulationsknoten eingebunden wurden.

Prüfling war ein Zentrales Bordnetz Steuergerät. Es besitzt mehrere Hundert HW Eingangs- und Ausgangsschnittstellen, ist an verschiedenen CAN und LIN Bussen angeschlossen und zeichnet sich dadurch und einer extrem hohen Anzahl verschiedener auf ihm laufenden Funktionen gegenüber anderen Automobil Steuergeräten aus.

Grundlage für die Erstellung der Benutzerprofile waren Interviews mit den Applikationsexperten sowie die zugehörigen Lastenhefte. Es wurden ca. 25 Markov Automaten entwickelt, die im Testlauf parallel aktiviert wurden. (Funktionen: Zündung, Außenlicht, Wischeransteuerung, Zentralverriegelung,...)

Am schwierigsten gestaltete sich die Erstellung der Regeln. Hierzu wurden Funktionsanforderungsexperten und erfahrene Funktionstester in mehreren Schleifen befragt. Es wurden an die 50 Regeln aufgestellt und parallel im Testlauf überwacht.

Die Testläufe wurden über mehrere Stunden durchgeführt. Zu dem Zeitpunkt waren noch nicht Testende Kriterien implementiert, so dass keine Aussagen über die Abdeckung der Automaten gemacht werden kann.

In der Studie konnten bisher vorrangig zwei Arten von Fehlern entdeckt werden:

- Konkurrierende Zugriffe auf Treiber (Aufblitzen und mangelnde Synchronizität bei Leuchten)
- mangelnde zeitliche Abstimmung von interagierenden Applikationen

die sporadisch auftraten! Über die Anzahl der gefundenen Fehler darf ich keine Aussagen machen.

Weitere positive Effekte waren verbesserte Anforderungsdokumente. In den intensiven Interviews mit den Applikationsexperten für die korrekte Aufstellung der Regeln wurden noch einmal die grundlegenden Prinzipien und Leitlinien der verschiedenen Anwendungen diskutiert. Dies führte im Folgenden auch zu klareren Spezifikationen der Anforderungen.

Bei der Einführung und Entwicklung des **SRT** zeigte sich auch, dass eine neue Testrolle einzuführen ist: Ein **SRT** Experte, der vor allem die Abfragesprache gut beherrscht. Er hat zum einen mit den Anwendungsexperten die Regeln inhaltlich abzuklären, zum anderen mit den Testexperten die konkrete Umsetzung am Prüfstand zu besprechen.

Für die Weiterentwicklung ist vorgesehen:

- Erweiterungen der Testdatengenerierung zur Szenarien Generierung (Beispiel: Normallauf, Ruhe, Extremlauf), bei der Gruppen von Markov Automaten in sogenannten Szenarien zusammengefasst werden.

- Verbesserte Fehleranalyse Unterstützung (Getriggertes Logging, grafische Unterstützung,..)

Dringlichste Aufgabe ist jedoch eine Bewährung dieses explorierten Ansatzes als neue, ergänzende Qualitätssicherungsmaßnahme. Dazu wurde die Neuentwicklung an einen speziellen Teststand angebunden und wird ab Sommer dieses Jahres in einer laufenden Steuergeräte Entwicklung eingesetzt.

Danksagung Diese Arbeit wäre ohne die zahlreichen Diskussionen und die tatkräftige Unterstützung der Herren R. Immig, St. Laufmann und M. Wagener nicht denkbar gewesen. Ihnen sei an dieser Stelle herzlichst gedankt!

Literatur

[AL15] ALL4TEC: MaTeLo Overview. http://www.all4tec.net/MaTeLo/matelo-features.html, 15.04.2015.

[GG93] Grochtmann, Mathias; Grimm, Klaus: Classification-trees for partition testing. Software Testing, Verification and Reliability, 3(3):63–82, 1993.

[GW10] Grossmann, J.; Wiesbrock, H.-W.: An Approach and Extension of Temporal Logic for Testing Real Time Properties of Embedded Systems. unpublished manuscript, 2010.

[Im07] Immig, Ralf: Abdeckungskriterien für den funktionalen Test eines elektronischen Steuergerätes. Diplomarbeit, HU Berlin, Juni 2007.

[Li02] Liggesmeyer, Peter: Software-Qualität. Spektrum Akademischer Verlag Heidelberg-Berlin, 2002.

[MP92] Manna, Z.; Pnueli, Amir: The Temporal Logic of Reactive and Concurrent Systems. Springer, New York, 1992.

[Oc05] Ocone, Daniel: Chapter 5: Markov Chain Models. http://faculty.oxy.edu/angela/prior2005. Vorlesungsskript.

[P6] Pöppel, Ernst: Der Rahmen. Carl Hanser Verlag, München, 2006.

[se15] sepp.med: MBTsuite - The testing framework https://www.seppmed.de/produkte/mbtsuite.html, 15.04.2015.

[WS13] Waldmann, Karl-Heinz; Stocker, U. M.: Stochstische Modelle. Springer-Verlag, Kapitel Markov-Ketten, S. 9 – 57, 2013.