

# Modelchecking von Klassifikationsbaum-Testsequenzen

Alexander Krupp, Wolfgang Mueller  
Universität Paderborn/C-LAB, Paderborn, Germany

## Zusammenfassung

Im Entwurf und Test von elektronischen Systemen stellt sich die Aufgabe, Testmuster mit Anforderungsbeschreibungen abzugleichen. Dieser Artikel stellt eine Methodik zur formalen Verifikation von Testmustern bzgl. funktionalen, zeitannotierten Anforderungsbeschreibungen vor. Während Teile der Anforderungsbeschreibungen in temporallogische Formeln abgebildet werden können, liefern die Testfälle der Klassifikationsbaummethode das zustandsorientierte Modell, das verifiziert werden kann. Erste Ergebnisse zeigen, daß die Methodik selbst für große Testsequenzen anwendbar ist.

## 1 Einleitung

Insbesondere die frühen Phasen des Entwurfs elektronischer Systeme beinhalten mehrere Herausforderungen. In der Automobilindustrie wird z.B. Telelogic DOORS zur Definition von Anforderungsbeschreibungen häufig angewandt. Anforderungen werden hier meist in natürlichsprachlichen Sätzen in Verbindung mit Tabellenstrukturen gefasst, die die Grundlage für den Entwurf und Analyse des zu entwerfenden Systems bilden. In weiteren Analysen werden dann formale Verifikationen [7] und Simulationen [1] eingesetzt. Im Bereich der formalen Verifikation (z.B. in der EDA (Electronic Design Automation)) finden Modell- und Equivalence-Checking weite Anwendung [7]. Simulationen werden immer auf Basis eines Modells durchgeführt. Je nach Anwendungsbereich und Modellierungstiefe bzw. Abstraktionsgrad finden hier verschiedene Simulatoren z.B. Matlab, SystemC, SystemVerilog, oder VHDL ihre Anwendung. Während die Verfahren zur automatischen Testmuster-Generierung für spätere Entwurfsphasen ziemlich ausgereift sind [6, 4], ist die Testmuster-Erstellung in frühen Phasen ein noch relativ unbearbeitetes Gebiet und wird meist manuell durchgeführt. In jüngster Zeit findet hier die Idee des modellbasierten Tests größere Beachtung, wobei Informationen zur Erstellung der Testmuster aus der Struktur des jeweiligen Modells gewonnen werden.

Zum modellbasierten Test von elektronischen Steuergeräten im Automobil müssen zunächst die Anforderungsbeschreibungen strukturiert und in Sequenzen von Testmustern übersetzt werden. In vielen Fällen liegt hier das Modell als Matlab/Simulink-Modell vor. Zur strukturierten Ableitung modellbasierter Tests schlägt Conrad [3] Klassifikationsbäume als Zwischenschritt vor (siehe Abb. 1). Ein Klassifikationsbaum wird manuell aus den Anforderungsbeschreibungen und dem Modell erstellt und dient zur automatischen Ableitung von Testumgebungen mit denen das simulationsfähige Modell validiert wird. In diesem Kontext stellen wir eine Methodik zur formalen Verifikation der Testmustersequenzen bzgl. der Anforderungsspezifikation vor. Die Methodik basiert auf der Erstellung von Testmustersequenzen auf Basis der Klassifikationsbaummethode. Da die so strukturierte Testmustersequenz als Menge von Zustandsübergängen aufgefasst werden kann, kann aus ihr leicht ein endlicher Automat abgeleitet werden. Lassen sich außerdem Anforderungsbeschreibungen in temporallogische Formeln überführen, kann auf Basis dieser Formeln eine Verifikation mittels Modelchecking durchgeführt werden. Zur Echtzeitverifikation zeitannotierter Testsequenzen verwenden wir den Modelchecker RAVEN, auf dem die Zahlen unserer momentanen Untersuchungen basieren.

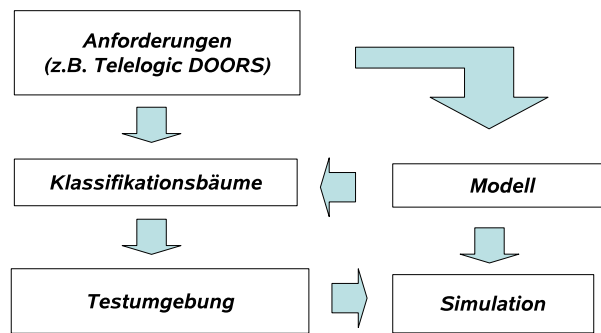


Abbildung 1: Modellbasierter Test auf Basis der Klassifikationsbaummethode

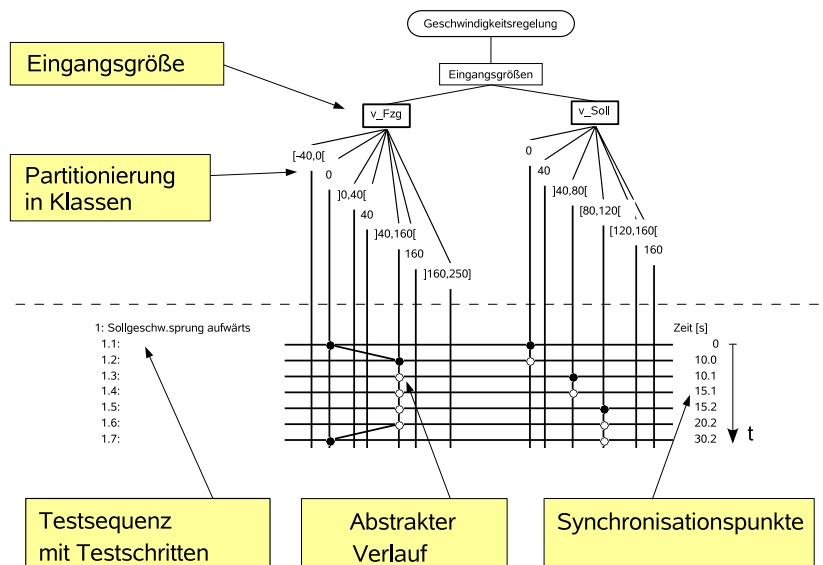


Abbildung 2: Erweiterte Klassifikationsbaummethode

Der nachfolgende Artikel gliedert sich wie folgt. Die nächsten beiden Abschnitte beschreiben die Klassifikationsbaummethode und den RAVEN Modelchecker. Kapitel 4 stellt unseren Ansatz vor, der dann in Kapitel 5 an einem Beispiel näher erläutert wird. Danach präsentieren wir unsere experimentellen Ergebnisse und schließen mit einem Ausblick auf zukünftige Arbeiten.

## 2 Klassifikationsbäume

Zur strukturierten Darstellung von Testfällen wurden Anfang der 90er Jahre bei der Daimler-Benz AG die Klassifikationsbäume entwickelt [5]. Ausgehend von einem Wurzelknoten erfolgt eine Zerlegung der möglichen Testeingaben in *Klassifikationen* und *Klassen*. Eine Klasse ist eine abstrahierte Menge möglicher Belegungen. Die Klassen bilden die Blätter des Baumes, wobei jeder Klasse eine Spalte in einer *Kombinationstabelle* zugeordnet wird. Eine Zeile der Kombinationstabelle stellt einen Testfall dar. Jedem Testfall werden Klassen von Eingabewerten derart zugeordnet, daß zu jeder Klassifikation genau eine Klasse ausgewählt wird. Die Erstellung von Klassifikationsbäumen und der Kombinationstabelle wird durch die auf der CP-Methode (Category-Partition) basierende Klassifikationsbaummethode unterstützt [8].

In der ursprünglichen Ausprägung beschreiben Klassifikationsbäume verschiedene Eingabekonstellationen für ein Testobjekt. Um auch zeitabhängige Testverläufe formulieren zu können, wurde die Notation seit 1998 schrittweise zur Klassifikationsbaum-Methode für eingebettete Systeme (CTM/ES) erweitert [3]. Der Klassifikationsbaum (siehe Abb. 2) wird speziell aus der technischen Schnittstelle des Testobjektes abgeleitet, d.h. jede Eingangsgröße des Testobjektes wird durch eine Klassifikation dargestellt. Der zulässige Wertebereich einer Eingangsgröße wird dann über Intervalle oder Einzelwerte in Klassen partitioniert.

Die abstrakte Darstellung von Testszenarien wird aus den einzelnen Testschritten aufgebaut, die als zeitliche Abfolge in die Zeilen der Kombinationstabelle abgebildet werden. Eine solche Folge von Testschritten wird als Testsequenz bezeichnet. Jeder Testschritt definiert damit die Eingaben des Testobjektes über eine Zeitspanne. Der Aktivierungszeitpunkt eines Testschritts wird als Synchronisationspunkt bzw. als Stützstelle bezeichnet. Diese Zeitpunkte werden in einer zusätzlichen Spalte zu jedem Testschritt angegeben. Der Signalverlauf zwischen den Synchronisationspunkten wird durch verschiedene Interpolationen beschrieben, wobei der Übergang z.B. als Sprung-, Rampen-, oder Sinusfunktion erfolgen kann. In der Kombinationstabelle werden Übergangsfunktionen durch verschiedene Linien zwischen den Synchronisationspunkten repräsentiert.

### 3 Echtzeit-Modelchecking mit RAVEN

Die Einführung von Modelchecking im Bereich des Entwurfs von elektronischen Systemen, insbesondere im Chip-Entwurf, basieren i. W. auf den Arbeiten von Clarke et al. [2] und dem daraus hervorgegangenen SMV Modelchecker. SMV verifiziert eine Menge von synchronen Zustandsautomaten anhand einer Menge von CTL-Formeln (Computation Tree Logic).

In unserer Arbeit benutzen wir den Echtzeit-Modelchecker RAVEN (Real-Time Analyzing and Verification Environment), der Standard-Modelchecking zur Verifikation von Echtzeitsystemen um den Zeitbegriff und zusätzliche Analysealgorithmen erweitert [9]. Ein Modell wird in RAVEN als I/O-Intervallstruktur beschrieben. Die Spezifikation erfolgt in CCTL (Clocked CTL).

I/O-Intervallstrukturen stammen von Kripkestrukturen ab mit der Erweiterung, dass  $[min, max]$ -Zeitintervalle an den Transitionen angegeben werden können. Die Ausführung einer Intervallstruktur ergibt sich folgendermaßen. Wir nehmen an, daß jede Intervallstruktur genau eine (Stopp-)Uhr besitzt. Die Uhr wird auf Null gesetzt, sobald ein neuer Zustand gesetzt wurde. Ein Zustand darf verlassen werden, wenn die aktuelle Zeit zur Verzögerungszeit an einer Transition paßt. Der Zustand *muß* verlassen werden, sobald die maximale Verzögerungszeit über alle Transitionen erreicht wurde.

CCTL ist eine Temporallogik mit Zeitbeschränkungen. Im Gegensatz zu klassischer CTL können die temporalen Operatoren **F** (eventually), **G** (globally) und **U** (until) durch Zeitintervalle beschränkt werden:  $[a, b]$ ,  $a \in \mathbb{N}_0, b \in \mathbb{N}_0 \cup \{\infty\}$ .  $\infty$  ist definiert durch:  $\forall i \in \mathbb{N}_0 : i < \infty$ . Das Intervall darf auch in der Form  $[b]$  angegeben werden. In diesem Fall wird die untere Schranke als Null angenommen, d.h.  $[b]$  entspricht  $[0, b]$ ,  $b \in \mathbb{N}_0 \cup \{\infty\}$ . Wird kein Intervall angegeben, so wird implizit ein Intervall  $[0, \infty]$  angenommen. Der **X**-operator (i.e., next) akzeptiert nur eine einfache Zeitbeschränkung der Form  $[a]$  ( $a \in \mathbb{N}$ ). Wird keine Beschränkung gesetzt, so wird sie zu Eins angenommen.

I/O-Intervallstrukturen und CCTL Formeln werden in der Eingabesprache RIL (RAVEN Input Language) spezifiziert. Eine RIL Spezifikation enthält

- a) globale Definitionen, z.B. bestimmte Zeitkonstanten und oft benutzte Formeln,
- b) die Beschreibung von I/O-Intervallstrukturen in parallel ausgeführten Modulen,
- c) CCTL Formeln, die bestimmte Eigenschaften des Modells spezifizieren.

Zur Erläuterung geben wir das kurzes Beispiel, bei dem es sich um ein Modul aus einem RIL Modell mit vier Zuständen und zwei Zustandsübergängen, von *wait* nach *failed* und von *wait* nach *accept* handelt.

```

MODULE consumer
SIGNAL
  state : {wait,reject,accept,failed}
INPUTS loadFail := GlobalFailure
        loaderIdle := (loader.state=loader.idle)
DEFINE rejectOrder := (state = reject)
INIT   state = wait
TRANS
|- state=wait
  -- loadFail --> state:=failed
  -- !loadFail & loaderIdle --> state:=accept
  ...

```

Das folgende Beispiel einer CCTL Formel beschreibt eine Eigenschaft des Eingabepuffers des consumer: jedem Akzeptieren einer Lieferung muß innerhalb der nächsten 100 Zeitschritte die Annahme (load) folgen.

```

AG((consumer.state = consumer.accept)
  -> AF[100]((loader.state = loader.wait)
             & AX(loader.state = loader.load)
             )
)

```

## 4 Klassifikationsbaum mit formaler Verifikation von Testsequenzen

Wie bereits erwähnt, erweitert die Klassifikationsbaummethode für eingebettete Systeme CTM/ES die ursprüngliche Klassifikationsbaummethode CTM um eine Komponente zur Definition von Testsequenzen und zur Beschreibung kontinuierlicher Signalverläufe. Dabei stellen die verschiedenen Klassifikationen mit ihrem Wertebereich  $X_i$  und die zugeordneten Klassen  $P_i(X_i) = \{\pi_{i,1}, \pi_{i,2}, \dots, \pi_{i,n}\}$  der CTM einen abstrakten Zustandsraum der Eingabevariablen dar. Jeder Eingabevariablen des Testobjekts entspricht dann eine Klassifikation. Eine Klasse  $\pi_{i,j}$  stellt ein Wertintervall aus dem Wertebereich einer Eingangsvariable dar. Der zeitliche Ablauf eines Test szenarios wird mit Hilfe von Stützstellen beschrieben. Wenn  $T = \{t_0, t_1, \dots, t_e\}$  mit  $t_0 < t_1 < \dots < t_e$  die Menge der Stützstellen eines Test szenarios ist, so heißen die Zeitintervalle  $[t_0, t_1], [t_1, t_2], \dots, [t_{e-1}, t_e]$  Testschritte. Eine Klassenfunktion  $\tilde{\pi}_i : T \rightarrow P_i(X_i)$  ordnet jeder Stützstelle eine Klasse und eine Wertefunktion  $v_i : T \rightarrow X_i$  ordnet jeder Stützstelle einen Wert zu. Eine Wertefunktion  $v_i$  heißt *kompatibel* zu der Klassenfunktion  $\tilde{\pi}_i$ , wenn zu jedem Zeitpunkt  $t$  gilt, daß  $v_i(t)$  zu der entsprechenden Klasse  $\tilde{\pi}_i(t)$  gehört.

Eine Testsequenz wird durch Vorgabe einer Klassenfunktion, einer kompatiblen Wertefunktion und einer Interpolationsvorschrift definiert. Die Interpolationsvorschrift gibt den Wert einer Eingangsvariablen zwischen den Stützstellen  $t_k$  an. Bei Verwendung von Rampenfunktionen und Sinus-Halbwellen als Interpolationsvorschrift ergibt sich dadurch ein stetiger Testdatenverlauf  $\bar{v}_i : \bar{T} \rightarrow X_i$ , wobei  $\bar{T} \supset T$  eine strikt geordnete Menge von Zeitpunkten im Sinne des klassischen Zeitbegriffs ist.

Aus einem Testdatenverlauf  $\bar{v}_i(t)$  und einer Diskretisierungsschrittweite  $\Delta T$  läßt sich nun eindeutig ein diskretisierter Testdatenverlauf  $\bar{v}_i^D : \mathbb{N}_0^{\leq \frac{t_e}{\Delta T}} \rightarrow X_i$  mit  $\bar{v}_i^D(k) = \bar{v}_i(k\Delta T)$  ableiten.  $\Delta T$  ist dabei so zu wählen, daß die Zeitdauer der Testsequenz  $t_e - t_0$  und die Zeitdauern  $t_i - t_0, 0 < i < e$  ganzzahlige Vielfache dieser Schrittweite sind.

Ein solcher diskretisierter Testdatenverlauf bildet dann eine Folge diskreter Zustände, die sich als Modell für einen Modelchecker eignet, wodurch es möglich wird die aus der CTM/ES generierten Testdatenverläufe durch Modelchecking zu verifizieren. Abb. 3 zeigt schematisch, wie aus Anforderungen Klassifikationsbäume erstellt und mit deren

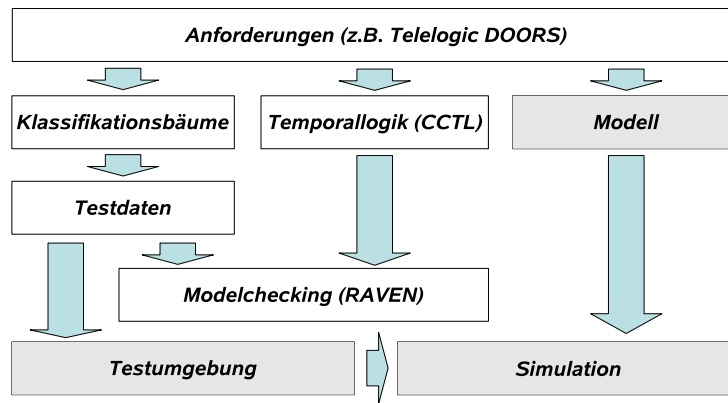


Abbildung 3: Verifikationsfluß für Testdaten

Hilfe Testdaten für eine Testumgebung generiert werden, die die Stimulierung und -steuerung eines Simulationsmodells realisiert. Die Testdaten können nun außerdem durch einen Modelchecker basierend auf den Anforderungen formal verifiziert werden. Im Zuge der CTM/ES ist in diesem Zusammenhang eine um Zeitintervalle erweiterte Temporallogik, wie CCTL, besonders geeignet. Dadurch lassen sich u.a. minimale Stabilitätsanforderungen an Zustandsfolgen formulieren, wie z.B.: “Die Bremse muß immer für mindestens 2000 ms betätigt werden.”

Für eine vollständige Analyse per Modelchecker werden alle Zeit-Wert-Paare  $(k, \bar{v}_1^D(k))$  des diskretisierten Testdatenverlaufs in ein Modell überführt, so dass jedem Index  $k$  ein Zustand entspricht. So lange ein Index  $> k$  existiert, wird dieser als Folgezustand eingetragen. Außerdem müssen die aus dem Testdatenverlauf abgeleiteten Eingabewerte dem Modelchecker zu jedem Zustand in geeigneter Form präsentiert werden, da ein Modelchecker wie RAVEN lediglich mit ganzzahligen Werten arbeitet, so daß die Testdaten skaliert und gerundet werden müssen. Die folgende Tabelle zeigt beispielhaft einen Testdatenverlauf für eine Variable  $v_1 \in \{x|x \in \mathbb{R}, 0 \leq x \leq 3,0\}$ , die nach der Klassifikationsbaummethode (CTM) zwischen den Stützstellen  $t_0 = 0,0$  s und  $t_1 = 1,0$  s mit  $v_1(t_0) = 3,0$  und  $v_1(t_1) = 0,5$  durch eine Rampenfunktion interpoliert wird. Die Tabelle zeigt die Wertefunktion  $v_1(t)$  an den Stützstellen, sowie den (interpolierten) Testdatenverlauf  $\bar{v}_1(t)$ . Die rechten beiden Spalten zeigen den diskretisierten Testdatenverlauf, der als Modell für den Modelchecker (MC) dient.

CTM			MC	
$\frac{t}{s}$	$v_1(t)$	$\bar{v}_1(t)$	$k$	$2\bar{v}_1^D(k)$
0,0	3,0	3,0	0	6
0,2	-	2,5	1	5
0,4	-	2,0	2	4
0,6	-	1,5	3	3
0,8	-	1,0	4	2
1,0	0,5	0,5	5	1

Es ist zu beachten, dass eine solche Modellbeschreibung in einem Deadlock endet. Dies ist ungewöhnlich, da in der Regel deadlockfreie Modelle für einen Modelchecker formuliert werden, was zur Ausgabe einer Warnung des Modelcheckers führen kann. Die eigentliche Verifikation der CCTL-Spezifikation ist dadurch aber nicht beeinträchtigt. In dem gegebenen Fall läßt sich eine Spezifikation der Art “v muß für mindestens 0,5 s größer als 2,5 sein.” mit den skalierten Werten wie folgt in CCTL formulieren:

AF AG[0,4] (v>5).

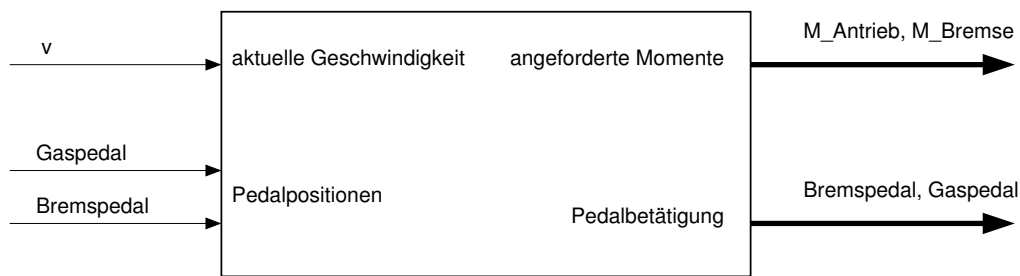


Abbildung 4: Funktionsblock Pedalinterpretation eines Tempomaten

Für den Modelchecker bedeutet diese CCTL Formel sinngemäß: Irgendwann muß für 5 diskrete Zeitschritte  $v > 5$  gelten. In dem folgenden Abschnitt wird die beschriebene Vorgehensweise noch näher an einem Beispiel eines Tempomaten erläutert. Es bleibt anzumerken, dass sich Zustandsfolgen, bei denen über längere Zeiträume keine Wertänderung auftritt, mit dem Modelchecker RAVEN einfach als verzögerte Zustandsübergänge beschreiben lassen. RAVEN enthält Optimierungen für derartige Zustandsübergänge, so dass sich auf diese Weise Laufzeit und Speicherbedarf verringern lassen.

## 5 Fallbeispiel

### 5.1 Tempomat

Bereits seit den 80er Jahren existiert der Tempomat als Komfortfunktion in Kraftfahrzeugen. Eine weiterentwickelte Variante mit Sicherheitsfunktion wird seit 1998 angeboten<sup>1</sup>. Dabei wird über Radar der Abstand zu vorausfahrenden Fahrzeugen ermittelt. Bei Unterschreiten eines Mindestabstands wird der Antrieb gedrosselt oder auch ein leichter Bremsvorgang eingeleitet, bis die Fahrzeuge in gleich bleibendem Abstand fahren. Sollte der Abstand sich trotzdem weiter verringern, muss der Fahrer spätestens nach einem Warnsignal eingreifen. Ein solcher Tempomat wird als *Tempomat mit Abstandsregelung* bezeichnet.

Ein Modul dieses Tempomaten interpretiert die Fahrereingaben, d.h., es bestimmt angeforderte Drehmomente in Abhängigkeit von der Pedalstellung. Dieses Modul wird mit “Pedalinterpretation” bezeichnet und ist in Abb. 4 mit Ein- und Ausgabesignalen dargestellt. Die aktuelle Fahrzeuggeschwindigkeit liegt am Eingang  $v$  an, die Stellung des Gas- bzw. Bremspedals liegt an den entsprechend benannten Eingängen an. Ausgegeben werden die angeforderten Abtriebsmomente für Antrieb und Bremse, sowie jeweils ein boolescher Wert, der die generelle Betätigung der Pedale signalisiert. Die folgenden Erläuterungen basieren auf dem Beispiel. Der systematische Test einer solchen Einheit mit Hilfe von Klassifikationsbäumen wird in [3] erläutert.

### 5.2 Klassifikationsbaum “Pedalinterpretation”

Der Klassifikationsbaum zum Testobjekt “Pedalinterpretation” zur Realisierung eines Tempomaten ist in Abb. 5 zu sehen. Die drei Eingänge  $v$ , *Bremse* und *Gas* sind als Klassifikationen im Klassifikationsbaum abgebildet. Zu jeder Klassifikation ist eine Menge von Klassen definiert, die den Wertebereich der Klassifikationen mit Hilfe von Intervallen überdecken. Die Wertebereiche sind wie folgt festgelegt:

$v$	$[-10, 70] \frac{\text{m}}{\text{s}}$
Bremspedal	$[0, 100] \%$
Gaspedal	$[0, 100] \%$

<sup>1</sup>z.B. von DaimlerChrysler als “DISTRONIC”

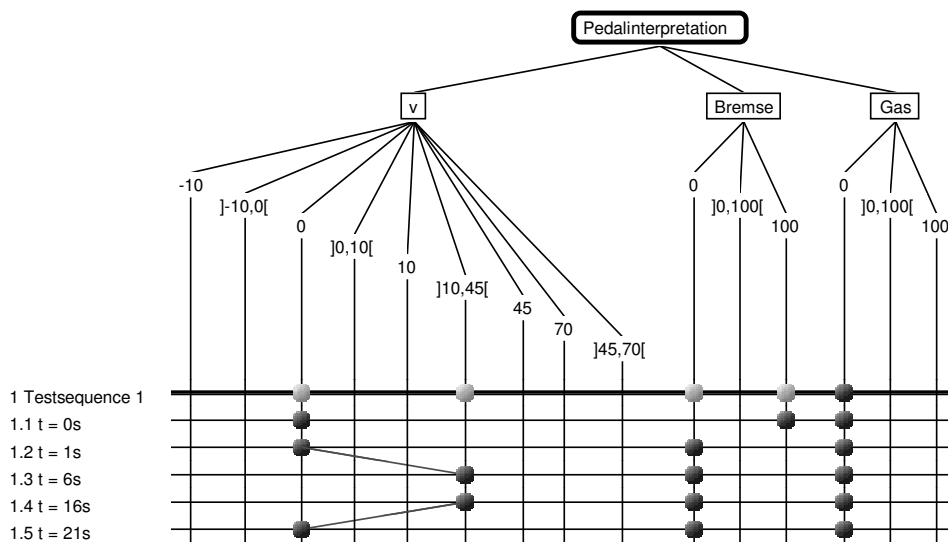


Abbildung 5: Klassifikationsbaum Pedalinterpretation

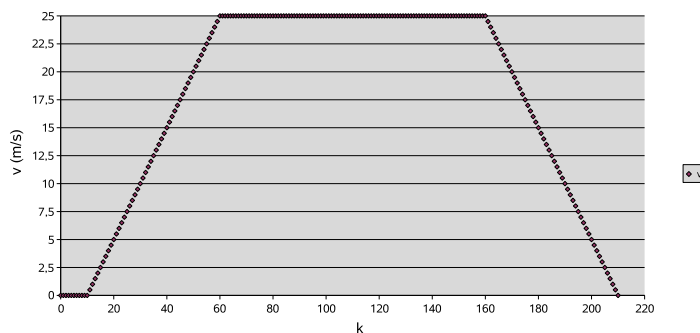


Abbildung 6: konkrete Testsequenz Pedalinterpretation

Der Wertebereich jeder Klassifikation wird hierbei in Klassen unterteilt. Für die Geschwindigkeit  $v$  gibt es 9 Klassen. 5 Klassen erhalten die ausgezeichneten Werte  $\{-10, 0, 10, 45, 70\}$ . Den verbleibenden 4 Klassen werden die Intervalle zwischen diesen Werten zugewiesen, d.h.  $\{]-10, 0[, ]0, 10[, ]10, 45[, ]45, 70[\}$ . Für jedes Pedal gibt es 3 Klassen. Die Extremwerte 0 % und 100 % bilden zwei Klassen und das Intervall  $]0, 100[$  % bildet die dritte Klasse.

Entsprechend der CTM/ES gibt Abb. 5 eine abstrakte Testsequenz über eine Kombinationstabelle vor. Auf der linken Seite ist die Testsequenz mit “Testsequence 1” bezeichnet. In der gleichen Spalte sind die einzelnen Testschritte numeriert und mit ihrem Aktivierungszeitpunkt annotiert. Über die restlichen Tabellenspalten wird zu jedem Testschritt eine Klassenkombination ausgewählt. Der Übergang zwischen aufeinander folgenden Testschritten erfolgt in der Regel als Sprung, z.B. der Übergang zum Zeitpunkt 1s für die Klassifikation *Bremse*. Als Übergangsfunktion lässt sich u.a. auch eine Rampenfunktion auswählen. Die Rampe wird im Zeitschritt 1..6s durch eine durchgezogene Linie zwischen den Tabellenmarkierungen angedeutet. Bei der Interpretation dieser Testsequenz ist zu beachten, daß es sich hier um einen Modultest handelt, d.h. die Zu- und Abnahme der Geschwindigkeit ist nicht zwingend mit einer Pedalbetätigung verbunden. Ein entsprechendes Fahrstreckenprofil könnte jedoch einen derartigen Verlauf am realen Fahrzeug hervorrufen. Änderungen der Ausgabewerte in Abhängigkeit von der Geschwindigkeit sind hier zu testen.

Zu der abstrakten Testsequenz wird eine konkrete Testsequenz mit dem in Abb. 6 gezeigten Ablauf für  $v$  folgendermaßen abgeleitet. Die Diskretisierungsschrittweite beträgt  $\Delta T = 0.1\text{ s}$  und für das Zeitintervall von  $[6, 16]\text{ s}$  wird für  $v$  ein fester Wert von  $25 \frac{\text{m}}{\text{s}}$  aus dem durch die Kombinationstabelle vorgegebenen Intervall  $]10, 45[\frac{\text{m}}{\text{s}}$  gewählt. Zusammen mit den durch den Klassifikationsbaum fest vorgegebenen Werten für *Bremse* und *Gas* wird die Testsequenz als Modell für den Modelchecker kodiert. Mit Hilfe der CCTL sind dann verschiedene temporale Eigenschaften der Testsequenz prüfbar.

Der folgende Absatz zeigt einen Ausschnitt aus dem RIL Code, wobei die Variablen  $vs$ , *bremse* und *gas* den Klassifikationen  $v$ , *Bremse* und *Gas* im Klassifikationsbaum entsprechen. Die Variable  $k$  repräsentiert hierbei die Zeit. Da RAVEN nur mit ganzzahligen Werten arbeitet, wurde  $vs$  derart skaliert, dass gilt  $[vs] = 0.1 \frac{\text{m}}{\text{s}}$ , d.h.  $vs$  zählt in Einheiten von  $0.1 \frac{\text{m}}{\text{s}}$ . Die Anfangswerte werden im Initialzustand INIT vorgegeben. Es wird für den Zeitpunkt  $t = 0\text{ s}$  eine Transition auf die Anfangswerte kodiert, die schon bei INIT eingetragen wurden. Laut der CTM Kombinationstabelle erfolgt erst bei  $t = 1\text{ s}$  wieder eine Wertänderung, so dass direkt eine Transition für  $k = 10$  ausgeführt wird, die jedoch mit einer Verzögerung um 9 Schritte behaftet ist.<sup>2</sup> Für  $k \in [10, 60]$ , d.h.  $t \in [1, 6]\text{ s}$  wird die Rampenfunktion in der Variable  $vs$  nachgebildet, bis bei  $k = 60$  der Wert 250 erreicht ist (entsprechend  $v = 25 \frac{\text{m}}{\text{s}}$ ), der über die folgenden 100 Schritte beibehalten wird. Für  $k \in [160, 210]$  wird die fallende Rampenfunktion für  $vs$  nachgebildet, womit die Testsequenz endet.

```

MODULE Pedal
SIGNAL
  vs : RANGE[0,250]
  bremse : RANGE[0,100]
  gas : RANGE[0,100]
  k : RANGE[0,211]
INIT
  vs == 0 & bremse == 100 & gas == 0 & k == 0
TRANS
|- k = 0 -- --> vs := 0 ; bremse := 100 ; gas := 0 ; k := 10
|- k = 10 -- :10 --> vs := 0 ; bremse := 0 ; gas := 0 ; k := 11
|- k = 11 -- --> vs := 5 ; bremse := 0 ; gas := 0 ; k := 12
|- k = 12 -- --> vs := 10 ; bremse := 0 ; gas := 0 ; k := 13
...
|- k = 59 -- --> vs := 245 ; bremse := 0 ; gas := 0 ; k := 60
|- k = 60 -- --> vs := 250 ; bremse := 0 ; gas := 0 ; k := 160
|- k = 160 -- :100 --> vs := 250 ; bremse := 0 ; gas := 0 ; k := 161
|- k = 161 -- --> vs := 245 ; bremse := 0 ; gas := 0 ; k := 162
|- k = 162 -- --> vs := 240 ; bremse := 0 ; gas := 0 ; k := 163
...
|- k = 208 -- --> vs := 10 ; bremse := 0 ; gas := 0 ; k := 209
|- k = 209 -- --> vs := 5 ; bremse := 0 ; gas := 0 ; k := 210
|- k = 210 -- --> vs := 0 ; bremse := 0 ; gas := 0 ; k := 211
|- k = 211 -- --> KEEP
END
...

```

Wir gehen nun davon aus, dass die folgenden Anforderungen, die nach CCTL zu übersetzen sind, durch die Testsequenz erfüllt werden sollen:

1. Für  $0.5\text{ s}$  nach Einschalten muss die Bremse betätigt sein.
2. Die Geschwindigkeitsvorgabe muss für mindestens  $12\text{ s}$  größer als  $10 \frac{\text{m}}{\text{s}}$  sein.
3. Der Zustand “Bremse unbetätigt” muss immer für mindestens  $20\text{ s}$  beibehalten werden.
4. Die Geschwindigkeitsvorgabe muss ohne Betätigung der Bremse größer als  $20 \frac{\text{m}}{\text{s}}$  sein, für mindestens  $10\text{ s}$ .

Diese Anforderungen lassen sich dann relativ intuitiv nach CCTL übersetzen:

<sup>2</sup>Die Standardverzögerung für Transitionen in RAVEN ist 1, so dass eine Verzögerung um 9 Schritte als 10 darzustellen ist.

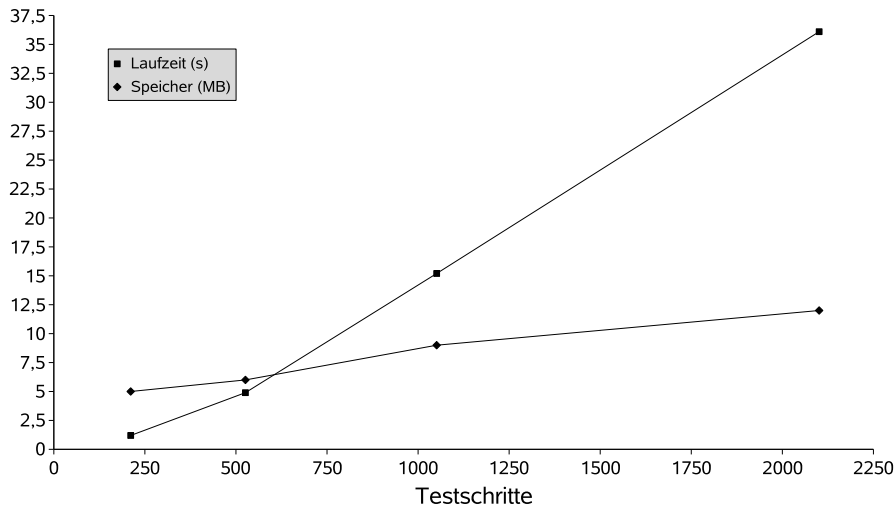


Abbildung 7: RAVEN Laufzeit und Speicherbedarf

1. AG[0,49] (Pedal.bremse = 100)
2. AF AG[0,119] (Pedal.v > 100)
3. AG ( (Pedal.bremse > 0) -> AX ( (Pedal.bremse = 0) -> AG[199] ( Pedal.bremse = 0 ) ) ) & (Pedal.bremse = 0) -> AG[199] ( Pedal.bremse = 0 )
4. AF AG[0,99] (Pedal.v > 200 & Pedal.bremse = 0)

## 6 Versuchsergebnis

Die Laufzeit des Modelcheckers RAVEN für die oben beschriebene Testsequenz lag bei ca. 1.22 s bei einem Speicherverbrauch von unter 5 MB auf einem Intel P4, 2.2 GHz, 1 GB RAM. Die 7 geprüften CCTL Formeln hatten dabei nur einen Einfluss von < 100 ms auf die Laufzeit. Bei einer Diskretisierungsschrittweite von 0.1 s ergaben sich 211 Testschritte. Um zu prüfen, wie Laufzeit und Speicherverbrauch des Modelcheckers mit der Testsequenzlänge skaliert, wurden drei weitere Testsequenzen mit Diskretisierungsschrittweiten von 40 ms, 20 ms und 10 ms generiert. Dadurch ergaben sich verfeinerte Testsequenzen mit jeweils 526, 1051 und 2101 Schritten. Die folgende Tabelle zeigt den Zeitaufwand und den Speicherbedarf des Modelcheckers in Abhängigkeit von der Diskretisierungsschrittweite und von der Testsequenzlänge. Die Ergebnisse sind in Abb. 7 graphisch aufbereitet. Es ist zu erkennen, dass die Laufzeit für die drei längsten Testsequenzen nahezu linear ansteigt.

Testsequenz		RAVEN	
$\Delta T$	Testschritte	Laufzeit	Speicherbedarf
100 ms	211	1,2 s	5 MB
40 ms	526	4,9 s	6 MB
20 ms	1051	15,2 s	9 MB
10 ms	2101	36,1 s	12 MB

## 7 Zusammenfassung

In diesem Artikel stellen wir eine Methodik zur formalen Verifikation der Testmustersequenzen bzgl. der Anforderungsspezifikation auf Basis der Klassifikationsbaummethode

zum Modelbasierten Test von elektronischen Steuergeräten vor. Details dieses Verfahrens erläuterten wir anhand des Tests eines Tempomaten aus dem Automobilbau. Erste Evaluierungen zeigen vielversprechende Ergebnisse. Es bleibt jedoch festzuhalten, dass das verwendete Beispielmodell nur wenige Eingabevariablen besitzt. Ferner war die gewählte Auflösung im Zeit- und Wertebereich relativ gering gewählt, so dass der Zustandsraum durch die Modelcheckervariablen noch nicht besonders groß war. Die Laufzeit- und Speicherwerte des Modelcheckers sind jedoch noch in einem Bereich, der für eine Anwendung annehmbar scheint. Die temporallogische Prüfung von Testdaten ermöglicht somit auch die Erstellung sehr langer verifizierter Testsequenzen. Je nach Prüfziel für die CCTL-Spezifikationen ist jedoch im Rahmen der Klassifikationsbaummethode ausreichend Spielraum vorhanden, um Modelchecking auch auf einem höheren Abstraktionsniveau anzuwenden und damit den Modelchecking-Aufwand zu verringern, falls der Zustandsraum durch eine konkrete Testsequenz zu groß werden sollte.

## Danksagung

Die hier beschriebene Arbeit wurde durch das BMBF-Projekt IMMOS gefördert.

## Literatur

- [1] J. Banks, J. Carson, B. L. Nelson, and D. Nicol. *Discrete-Event System Simulation*. Prentice Hall, 4 edition, 2004. ISBN 0-131-44679-7.
- [2] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Lecture Notes in Computer Science*, 131:244–263, 1981.
- [3] M. Conrad. *Modell-basierter Test eingebetteter Software im Automobil*. Deutscher Universitäts-Verlag, Wiesbaden, 2004.
- [4] A. Crouch. *Design-For-Test For Digital IC's and Embedded Core Systems*. Prentice Hall PTR, 1999. ISBN 0-130-84827-1.
- [5] M. Grochtmann and K. Grimm. Classification trees for partition testing. *Softw. Test., Verif. Reliab.*, 3(2):63–82, 1993.
- [6] N. K. Jha and S. Gupta. *Testing of Digital Systems*. Cambridge University Press, 2003. ISBN 0-521-77356-3.
- [7] T. Kropf. *Introduction to formal hardware verification*. Springer, 1999. ISBN 3-540-65445-3.
- [8] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Commun. ACM*, 31(6):676–686, 1988.
- [9] J. Ruf. RAVEN: Real-time analyzing and verification environment. *Journal on Universal Computer Science (J.UCS)*, Springer, Heidelberg, Feb. 2001.