BDD-basierte Verifikation von Realzeit-Systemen

Dirk Beyer and Andreas Noack*

Lehrstuhl für Software-Systemtechnik, BTU Cottbus

Zusammenfassung Diese Arbeit behandelt die effiziente Erreichbarkeitsanalyse von Timed Automata. Wir beschreiben eine Erreichbarkeitseigenschaften erhaltende Diskretisierung der Zeit. Diese ermöglicht es, Konfigurationsmengen von Timed Automata als Binary Decision Diagrams (BDDs) darzustellen. Die kompakte BDD-Repräsentation großer Mengen erfordert geeignete Variablenordnungen. Zur deren Bestimmung nutzen wir Strukturinformationen aus der Modellierungsnotation Cottbus Timed Automaton. Wir belegen die erzielten Effizienzverbesserungen durch Meßwerte.

1 Einleitung

Da in einer zunehmenden Anzahl von Anwendungsbereichen Steuerungsaufgaben von Computersystemen übernommen werden, spielt die Korrektheit, insbesondere bei sicherheitskritischen Anlagen, eine immer größere Rolle. Dabei kommt auch der formalen Spezifikation und Verifikation mehr und mehr Bedeutung zu. Ein Problem dabei ist, daß gerade bei größeren Systemen die automatische Verifikation, also der Nachweis, ob das Systemmodell die vorgegebenen Eigenschaften erfüllt, meist noch nicht problemlos möglich ist.

In dieser Arbeit wird für die Modellierung von Realzeitsystemen das Modell der zeitbehafteten Automaten (Timed Automata) verwendet. Dabei wird das Modell des finiten Automaten durch diskrete Variablen und kontinuierlich den Wert verändernde Uhrenvariablen erweitert. Ein Zustand eines Timed Automaton enthält ein logisches Prädikat, welches erfüllt sein muß, solange der Automat in diesem Zustand ist (Invariante). Ein Zustandsübergang enthält ein logisches Prädikat, welches eine notwendige Bedingung zur Auswahl dieses Übergangs darstellt (Wächter), sowie eine Variablenzuweisung, durch die der neue Wert der Variablen bestimmt wird. Zur Synchronisation von parallel laufenden Automaten wird das Konzept der Synchronisationsmarken (CSP) verwendet.

Der Formalismus dieser Automaten bietet jedoch keine Konzepte zur modularen Strukturierung von Systemen. Die existierenden Notationen wurden von Beyer und Rust als Cottbus Timed Automata (CTA) um die folgenden Konzepte erweitert [BR98]: Modulare Strukturierung der Systembeschreibung, verschiedene Zugriffsarten für Variablen und Signale, sowie vereinfachte Modellierung von wiederkehrenden Systemteilen.

Durch Erreichbarkeitanalyse können Sicherheitseigenschaften eines CTA nachgewiesen werden. Dazu wird die Menge aller erreichbaren Konfigurationen des CTA berechnet und festgestellt, ob deren Schnittmenge mit einer Menge verbotener Konfigurationen leer ist. Entscheidend für die Effizienz ist die Datenstruktur, die zur Repräsentation von Konfigurationsmengen verwendet wird.

Bei der Repräsentation von Mengen diskreter Zustände haben sich Binary Decision Diagrams (BDDs) [Bry86] bereits vielfach bewährt. Hinzu kommt die Repräsentation von Mengen von Uhrenwerten, also Teilmengen von $\mathbb{R}^d$. Bisher wurden dazu meist Difference Bound Matrices (DBMs) [Dil89] verwendet. Diese sind aber ineffizient bei der Darstellung von nicht-konvexen Mengen und bei einer hohen Anzahl diskreter Zustände. Die Verwendung von BDDs auch zur Repräsentation von Uhrenwerten wird in [ABK+97] eingeführt. Basierend darauf zeigen wir in dieser Arbeit, wie Konfigurationsmengen von CTAs durch BDDs dargestellt werden können und wie Informationen aus der Struktur des CTA für eine effiziente Erreichbarkeitsanalyse genutzt werden können.

* Erreichbar unter: BTU, Postfach 10 13 44, D-03013 Cottbus, Germany; Tel. +49(355)69-3802, Fax.:-3810; {db|an}@informatik.tu-cottbus.de

2 Cottbus Timed Automata

Der Grundgedanke des in der Cottbuser Arbeitsgruppe entwickelten Modellierungsformalismus liegt in der Verknüpfung des bestehenden formalen Automaten-Modells mit dazu passenden Konzepten der Modularität. Im folgenden Unterabschnitt wird informell der Aufbau eines CTA-Moduls erläutert. Danach wird eine formale Einführung in das verwendete Automatenmodell gegeben. Zur Illustration wird im letzten Unterabschnitt das Beispiel des gegenseitigen Ausschlusses nach Fischer gezeigt.

2.1 Informale Einführung in den CTA-Formalismus

Eine CTA-Systembeschreibung besteht aus einer Menge von Modulen. Ein Modul wird als Topmodul bezeichnet und von keinem anderen Modul als Schablone benutzt. Alle anderen Module stellen Schablonenmodule dar. Somit kann ein System als hierarchische Enthaltenseinsstruktur modelliert werden, und mehrmals in ähnlicher Weise auftretende Komponenten müssen nur einmal notiert werden. Jedes Modul enthält die folgenden Bestandteile:

- Einen eindeutigen **Bezeichner**. Da ein CTA-Modell mehrere Module enthalten kann, ist eine eindeutige Identifizierung erforderlich.
- Eine **Schnittstellendeklaration**. Hier werden die verschiedenen vom Modul benutzten Signale und Variablen deklariert und mit den entsprechenden Einschränkungstypen versehen. Signale werden zur Synchronisation von parallel laufenden Modulen benutzt. Sie werden wie Synchronisationsmarken im CSP-Konzept behandelt. Variablen werden zum Speichern von diskreten Werten und insbesondere zur Modellierung der kontinuierlich verstreichenden Zeit verwendet.
- Einen **Timed Automaton**. Ein Modul kann einen Automaten enthalten, der das Verhalten des Moduls beschreibt. Er besteht aus einer endlichen Menge von diskreten Zuständen, einer endlichen Menge von Transitionen zwischen diesen Zuständen und einem Alphabet von Signalen.
- Eine **Initial-Konfiguration**. Sie enthält Bedingungen für die Anfangswerte der Variablen und legt die Startzustände der Automaten fest.
- **Modul-Instanzen**: Ein Modul kann Instanzen anderer Module enthalten. Hiermit können enthaltene Subsysteme modelliert werden. Sich ähnelnde Subsysteme müssen nicht mehrmals definiert, sonden nur mehrmals instanziiert werden. Eine Instanz wird durch die folgenden Merkmale bestimmt:
  - Ein Bezeichner zur Identifikation dieser Modulinstanz.
  - Ein Bezeichner, der das Schablonenmodell angibt, von dem die Instanz erzeugt werden soll.
  - Eine Zuordnung je einer Schnittstellenkomponente des instanziierten Moduls zu einer Schnittstellenkomponente des enthaltenden Moduls: Damit werden Schnittstellenvariablen und -signale des instanziierten Moduls mit den entsprechenden Komponenten des enthaltenden Moduls identifiziert.

Durch die Einführung der Modulkonzepte entstehen folgende Vorteile im Umgang mit den Modellen:

- Eine Systembeschreibung kann hierarchisch gruppiert werden. Subsysteme sind einfach austauschbar durch eine wohldefinierte Schnittstelle.
- Signale und Variablen haben einen Einschränkungstyp, der die Zugriffsart einer Variable oder eines Signals auf Lokal, Nur-lesen, Exklusiv-schreiben oder Mehrfach-einschränkbar festlegt.
- Wiederkehrende Teilsysteme müssen nicht mehrmals modelliert werden, sondern können mittels Instanziierungsmechanismen aus einer Schablone erzeugt werden.

Eine vollständige formale Definition der Semantik von CTA-Modellen ist in [BR99] zu finden. In der vorliegenden Arbeit wird dieser Formalismus, der in seiner allgemeinen Form auch hybride Aspekte unterstützt, auf Uhren (und somit auf Realzeit) eingeschränkt. Dabei wird als Grundlage die von Alur und Dill gegebene formale Definition der Timed Automata verwendet [AD94,Alu99].

2.2 Formale Definition der Timed Automata

Zur formalen Definition von Timed Automata müssen wir zunächst festlegen, welche **Uhrenbedingungen** zugelassen sind. Für eine Menge X von Uhren ist die Menge $\Phi(X)$ von Uhrenbedingungen φ definiert durch die Grammatik

$$\varphi := x \leq c \mid x \geq c \mid x < c \mid x > c \mid \varphi_1 \wedge \varphi_2,$$

mit $x \in X$ und $c \in \mathbb{N}$. Eine Uhrenbelegung v ist eine totale Abbildung von der Menge der Uhren X in die Menge der nichtnegativen reellen Zahlen $\mathbb{R}_+$. $V(X)$ bezeichnet die Menge aller Uhrenbelegungen von X. Die Uhrenbelegung, die allen Uhren den Wert 0 zuweist, wird mit v_0 bezeichnet. Für $\delta \in \mathbb{R}_+$ ist $v + \delta$ die Uhrenbelegung, die jeder Uhr x den Wert $v(x) + \delta$ zuweist. Für $Y \subseteq X$ bezeichnet $v[Y := 0]$ die Uhrenbelegung von X, die jeder Uhr aus Y den Wert 0 und den anderen Uhren denselben Wert wie v zuweist.

Ein **Timed Automaton** A ist ein Tupel $(L, L^0, X, \Sigma, I, E)$. Dabei ist

- L eine endliche Menge von Zuständen,
- $L^0 \subseteq L$ eine Menge von Initialzuständen,
- X eine endliche Menge von Uhren,
- Σ eine endliche Menge von Synchronisationsmarken,
- I eine totale Abbildung, die jedem Zustand aus L eine Invariante aus $\Phi(X)$ zuordnet,
- $E \subseteq L \times \Sigma \times \Phi(X) \times 2^X \times L$ eine Menge von Zustandsübergängen. Ein Zustandsübergang $(s, a, \varphi, \lambda, s')$ repräsentiert einen Übergang von Zustand s zu Zustand s'. Der Wächter φ muß erfüllt sein, damit der Übergang schalten darf. Die Menge λ beschreibt die Menge der Uhren, die der Übergang auf 0 rücksetzt.

Für eine konzisere Darstellung werden häufig zusätzlich diskrete Variablen eingeführt, die als Wertebereich eine endliche Teilmenge der natürlichen Zahlen haben und ihren Wert ausschließlich beim Schalten von Zustandsübergängen ändern. Diskrete Variablen werden hier nicht explizit betrachtet, da sie nur eine zusätzliche Schreibweise für Zustände sind.

Komplexe Systeme können durch **parallele Komposition** mehrerer Timed Automata beschrieben werden. Die Semantik einer Komposition zweier Timed Automata A_1 und A_2 entspricht der des Produktautomaten $A_1 \| A_2$.

Seien $A_1 = (L_1, L_1^0, X_1, \Sigma_1, I_1, E_1)$ und $A_2 = (L_2, L_2^0, X_2, \Sigma_2, I_2, E_2)$ Timed Automata. Dann ist das Produkt $A_1 \| A_2$ der Timed Automaton $(L_1 \times L_2, L_1^0 \times L_2^0, X_1 \cup X_2, \Sigma_1 \cup \Sigma_2, I, E)$ mit $I(s_1, s_2) = I_1(s_1) \wedge I_2(s_2)$ und folgenden Zustandsübergängen:

- für $a \in \Sigma_1 \cap \Sigma_2$, für jedes $(s_1, a, \varphi_1, \lambda_1, s_1') \in E_1$ und $(s_2, a, \varphi_2, \lambda_2, s_2') \in E_2$ ist $((s_1, s_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (s_1', s_2')) \in E$,
- für $a \in \Sigma_1 \setminus \Sigma_2$, für jedes $(s_1, a, \varphi_1, \lambda_1, s_1') \in E_1$ und $s_2 \in L_2$ ist $((s_1, s_2), a, \varphi_1, \lambda_1, (s_1', s_2)) \in E$,
- für $a \in \Sigma_2 \setminus \Sigma_1$, für jedes $(s_2, a, \varphi_2, \lambda_2, s_2') \in E_2$ und $s_1 \in L_1$ ist $((s_1, s_2), a, \varphi_2, \lambda_2, (s_1, s_2')) \in E$.

Die Zustände des Produktautomaten sind also Paare von Zuständen der Komponenten und die Invarianten sind Konjunktionen der Invarianten der entsprechenden Komponenten-Zustände. Zustandsübergänge mit gleichen Marken werden synchronisiert.

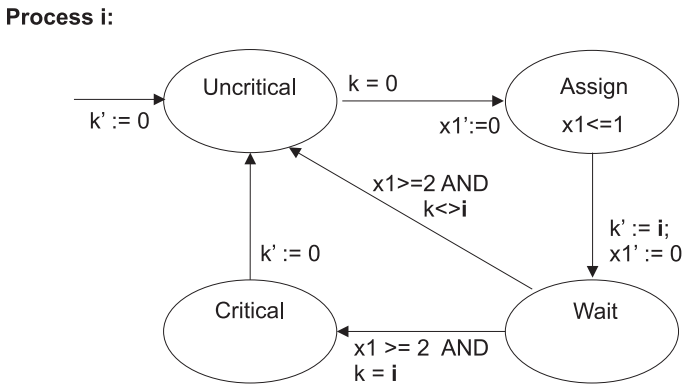

Abbildung1. Fischer's mutual exclusion protocol

2.3 Semantik

Die Semantik eines Timed Automaton wird durch ein zugeordnetes Transitionssystem definiert.

Ein **Transitionssystem** S ist ein Tupel $(Q, Q^0, \Sigma, \rightarrow)$. Dabei ist Q eine Menge von Konfigurationen, $Q^0 \subseteq Q$ eine Menge von Initialkonfigurationen, Σ eine Menge von Marken, und $\rightarrow \subseteq Q \times \Sigma \times Q$ eine Menge von Transitionen. Das System startet in einer Initialkonfiguration, und kann mit a von Konfiguration q nach Konfiguration q' übergehen, wenn $q \xrightarrow{a} q'$. Wir schreiben $q \rightarrow q'$, wenn $q \xrightarrow{a} q'$ für eine Marke a.

Einem Timed Automaton $A = (L, L^0, X, \Sigma, I, E)$ ist das Transitionssystem $S = (L \times V(X), L^0 \times \{v_0\}, \Sigma \cup \mathbb{R}_+, \rightarrow)$ zugeordnet, wobei $\rightarrow$ zwei Arten von Transitionen enthält:

- Transitionen durch das Vergehen von Zeit:
  Für $(s, v) \in L \times V(X)$, $\delta \in \mathbb{R}_+$ gilt $(s, v) \xrightarrow{\delta} (s, v + \delta)$, falls für alle $0 \leq \delta' \leq \delta$ die Invariante $I(s)$ von $v + \delta'$ erfüllt wird.
- Diskrete Transitionen:
  Für $(s, v) \in L \times V(X)$, $(s, a, \varphi, \lambda, s') \in E$ gilt $(s, v) \xrightarrow{a} (s', v[\lambda := 0])$, falls v den Wächter φ erfüllt.

Die so definierte Semantik wird auch als kontinuierliche Semantik von A bezeichnet, in Abgrenzung zu der im Abschnitt 3.3 eingeführten diskreten Semantik. Im folgenden definieren wir die Mengen der Läufe und der erreichbaren Zustände allgemein für einen Timed Automaton $A = (L, L^0, X, \Sigma, I, E)$ mit einer Semantik $S = (Q, Q^0, \Sigma_S, \rightarrow)$.

Eine endliche Folge von Konfigurationen $q_0, q_1, ..., q_{2k}$ ist ein **Lauf** von A bei Semantik S, wenn $q_0 \in Q^0$ und es für alle $i \in \{0, 1, ..., k-1\}$ ein $\delta_i \in \mathbb{R}_+$ mit $q_{2i} \xrightarrow{\delta_i} q_{2i+1}$ und ein $a_i \in \Sigma$ mit $q_{2i+1} \xrightarrow{a_i} q_{2i+2}$ gibt. $Run_S(A)$ bezeichnet die Menge der Läufe von A bei Semantik S.

Eine Folge von Zuständen $s_0, s_1, ..., s_k$ ist ein **zeitabstrakter Lauf**, wenn es Uhrenbelegungen $v_0, v_0', v_1, v_1', ..., v_k \in V(X)$ gibt, so daß $(s_0, v_0), (s_0, v_0'), (s_1, v_1), (s_1, v_1'), ..., (s_k, v_k)$ ein Lauf ist. $Run_S'(A)$ bezeichnet die Menge der zeitabstrakten Läufe von A bei Semantik S.

Ein Zustand s ist **erreichbar**, wenn es einen zeitabstrakten Lauf $s_0, s_1, ..., s$ gibt. $Reach_S(A)$ bezeichnet die Menge der erreichbaren Zustände von A bei Semantik S.

Beispiel. Als Beispiel zur Illustration der Timed Automata wird das Protokoll für gegenseitigen Ausschluß von Fischer genutzt [Lam87]. Das System besteht aus mehreren Prozessen wie dem in Abbildung 1, die je eine lokale Uhr x_i haben, und einer gemeinsam benutzten diskreten Variable k.

Der Timed Automaton besteht aus vier Zuständen. Nach der Initialisierung befindet sich der Automat im Zustand 'Uncritical' und die gemeinsam benutzte Variable k wurde auf den Wert '0' gesetzt, es hat also noch kein Prozeß den Wunsch zum Betreten der kritischen Sektion angemeldet. Im Zustand 'Uncritical' kann sich der Automat für eine unbegrenzte Zeit aufhalten. Hat kein Prozeß den Wunsch nach Betreten der kritischen Sektion angemeldet hat, kann er in den Zustand 'Assign' übergehen. Dieser Zustand modelliert die Tatsache, daß die Zuweisung der Prozeß-Nummer einige Zeit in Anspruch nimmt. Spätestens nach einer bestimmten Zeit (hier 1 Zeiteinheit) wird der Automat aufgrund der Invariante zu einem Zustandsübergang nach 'Wait' gezwungen. Nun kann kein anderer Automat mehr von 'Uncritical' nach 'Assign' übergehen, da k auf den Wert der Prozeß-Nummer gesetzt ist. Da im Zustand 'Wait' länger gewartet werden muß als im Zustand 'Assign', ist beim Verlassen von 'Wait' kein Prozeß mehr in 'Assign'. Genau ein Prozeß kann nun entsprechend des Wertes der Variablen k die kritische Sektion betreten. Alle anderen Prozesse müssen zurück in den Zustand 'Uncritical' und einen erneuten Versuch starten.

3 Repräsentation von Konfigurationsmengen durch BDDs

Im Abschnitt 3.1 wird ein Algorithmus zur Erreichbarkeitsanalyse dargestellt, der Mengen von Konfigurationen eines Timed Automaton verarbeitet. Wir repräsentieren diese Mengen durch die im Abschnitt 3.2 beschriebene Datenstruktur der Binary Decision Diagrams (BDDs). Dazu müssen die im allgemeinen unendlichen Mengen von Uhrenbelegungen auf endliche Mengen reduziert werden. Abschnitt 3.3 beschreibt eine solche Reduktion für eine Teilmenge der Timed Automata, die abgeschlossenen Timed Automata. Abschnitt 3.4 stellt vergleichend andere Repräsentationen von Mengen von Uhrenbelegungen dar, die nicht auf abgeschlossene Timed Automata beschränkt sind.

3.1 Erreichbarkeitsanalyse von Timed Automata

Das Erreichbarkeitsproblem ist die Frage, ob in einem Timed Automaton A ein Element einer Zustandsmenge E erreichbar ist, also ob $E \cap Reach(A) \neq \emptyset$. Einfache Sicherheitseigenschaften, also die Nichterreichbarkeit bestimmter Fehlerzustände, können als Erreichbarkeitsprobleme formuliert werden. Durch Einführung von Beobachtungsautomaten lassen sich aber auch Eigenschaften wie beschränkte Reaktionszeiten (nach jeder diskreten Transition mit der Marke a muß innerhalb von t Zeiteinheiten eine diskrete Transition mit der Marke b auftreten) auf Erreichbarkeitsprobleme reduzieren.

Der folgende Algorithmus gibt genau dann $true$ zurück, wenn ein Timed Automaton $A = (L, L^0, X, \Sigma, I, E)$ eine Zustandsmenge E erreichen kann.

$R := L^0 \times \{v_0\}$
do
  $R_{prev} := R$
  $R := R \cup \{q \mid \exists q' \in R : q' \rightarrow q\}$
  if $R \cap (E \times V(X)) \neq \emptyset$ **then return** $true$
while $R \neq R_{prev}$
return $false$

Entscheidend für die effiziente Implementierung des Algorithmus ist die Verwendung einer Datenstruktur, die Konfigurationsmengen kompakt repräsentiert und die benötigten Operationen Vergleich, Schnitt, Vereinigung und Berechnung der durch Transitionen entstehenden Nachfolgekonfigurationen ermöglicht.

3.2 Binary Decision Diagrams

Ein Binary Decision Diagram (BDD) [Bry86] repräsentiert eine Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$, und damit gleichzeitig die Menge von Booleschen Vektoren, für die diese Funktion 1 ergibt. BDDs zeichnen sich dadurch aus, daß sie auch sehr große Mengen oft kompakt darstellen.

Ein BDD ist ein gerichteter azyklischer Graph, der durch Reduktion eines binären Entscheidungsbaumes entsteht. Ein binärer Entscheidungsbaum besteht aus Entscheidungsknoten, 0-Terminalknoten und 1-Terminalknoten. Jeder Entscheidungsknoten ist einer Variablen zugeordnet und hat zwei ausgehende Kanten, eine 0-Kante und eine 1-Kante. Die durch den Entscheidungsbaum repräsentierten Vektoren entsprechen den Pfaden vom Wurzelknoten zu den 1-Terminalknoten. Ein binärer Entscheidungsbaum wird durch die Anwendung folgender Reduktionsregeln zu einem BDD:

1. Verschmelze alle isomorphen Unterbäume.
2. Eliminiere alle Knoten, deren zwei Kanten auf denselben Knoten zeigen.

Wir verwenden nur geordnete BDDs, bei denen auf jedem Pfad von der Wurzel zu einem Terminalknoten die Variablen in der gleichen Reihenfolge auftreten. Bei fester Variablenordnung ist die Darstellung einer Booleschen Funktion eindeutig.

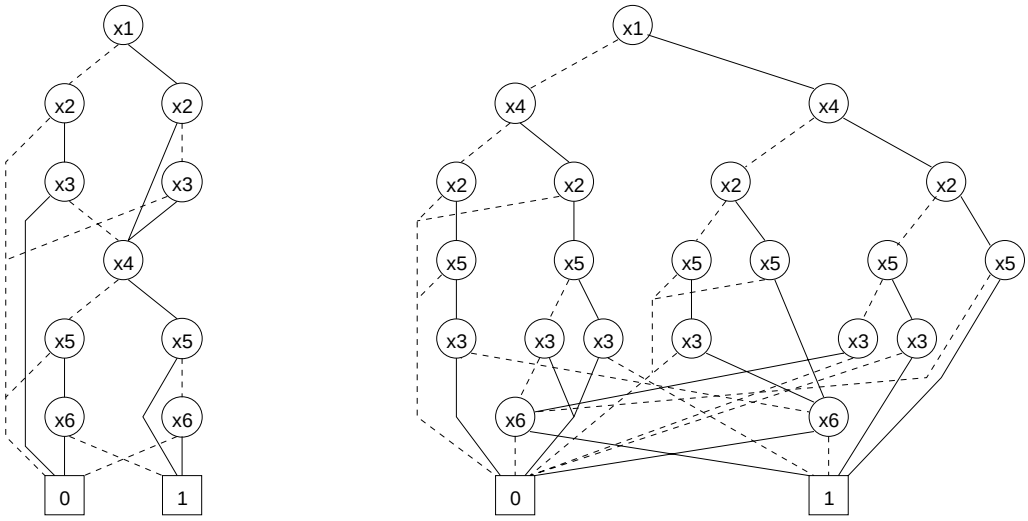

Abbildung2. BDD-Darstellungen der Menge $M \times M$ mit $M = \{(0,1,0),(1,0,1),(1,1,0),(1,1,1)\}$ bei verschiedenen Variablenordnungen. 0-Kanten sind gestrichelt, 1-Kanten durchgehend dargestellt.

Die Wahl der Variablenordnung kann einen großen Einfluß auf die Knotenzahl des BDDs haben, da besonders die Wirksamkeit der ersten Reduktionsregel, die meist den weitaus größeren Anteil an der Reduktion der Knotenzahl hat, stark von der Variablenordnung abhängt. Wir geben Variablenordnungen von der Wurzel des BDDs zu den Terminalknoten an. Zum Beispiel bedeutet für ein System aus zwei finiten Automaten A und B und einer diskreten Variablen x die Variablenordnung (A, x, B), daß auf jedem Pfad von der Wurzel des BDDs zu einem Terminalknoten Bits des binär kodierten Zustands von A vor Bits von x und Bits von x vor Zustandsbits von B durchlaufen werden.

Anhand eines Beispielsystems aus zwei finiten Automaten A_1 und A_2 sollen Regeln für gute Variablenordnungen erläutert werden. Zunächst seien beide Automaten unabhängig, das heißt sie kommunizieren nicht miteinander. Das aus beiden Automaten komponierte System bezeichnen wir

als A und die Mengen der erreichbaren Zustände von A, A_1 und A_2 als $R(A)$, $R(A_1)$ und $R(A_2)$. Dann ist $R(A) = R(A_1) \times R(A_2)$ und $|R(A)| = |R(A_1)| \cdot |R(A_2)|$. Für eine Menge M bezeichne $\|M\|$ die Knotenzahl ihrer BDD-Darstellung. Bei den Variablenordnungen (A_1, A_2) und (A_2, A_1), wenn also die Zustandsbits von A_1 und A_2 nicht vermischt werden, ist $\|R(A)\|$ nur die Summe von $\|R(A_1)\|$ und $\|R(A_2)\|$ (Abbildung 2).

Wir nehmen nun an, A_1 und A_2 kommunizieren über eine gemeinsame diskrete Variable x, deren Wertebereich $Range(x)$ relativ zu $R(A_1)$ und $R(A_2)$ klein ist. Die Menge der erreichbaren Konfigurationen des so komponierten Systems A' sei $R(A') = \bigcup_{v \in Range(x)} (\{v\} \times R_v(A_1) \times R_v(A_2))$. Bei der Variablenordnung (x, A_1, A_2) ist dann die Größe des BDDs für $R(A')$ nur etwa $\sum_{v \in Range(x)} (\|R_v(A_1)\| + \|R_v(A_2)\|)$. Man erhält eine gute Variablenordnung also durch Anordnung der gemeinsamen Variablen vor den Automaten.

Verallgemeinernd ergeben sich folgende Eigenschaften einer guten Variablenordnung [Min96]:

1. Stark voneinander abhängige Variablen liegen dicht beeinander.
2. Variablen mit großem Einfluß befinden sich weit oben, nahe der Wurzel.

3.3 Diskretisierung abgeschlossener Timed Automata

Eine Menge von Konfigurationen eines Timed Automaton ordnet Zuständen Mengen von Uhrenbelegungen zu. Die Mengen von Uhrenbelegungen können unendlich sein, und müssen, da BDDs nur endliche Mengen darstellen, auf endliche Mengen reduziert werden.

Wir geben eine Reduktion für abgeschlossene Timed Automata an. Abgeschlossene Timed Automata haben nur Uhrenbedingungen φ der Form $\varphi := x \leq c \mid x \geq c \mid \varphi_1 \wedge \varphi_2$, die Relationszeichen $<$ und $>$ dürfen nicht auftreten. Aussagen über abgeschlossene Timed Automata gelten auch für Systeme aus mehreren parallel komponierten abgeschlossenen Timed Automata, da der Produktautomat ebenfalls abgeschlossen ist.

Zu abgeschlossenen Timed Automata definieren wir eine diskrete Semantik. Zur besseren Unterscheidung bezeichnen wir die in Abschnitt 2.3 definierte Semantik als kontinuierliche Semantik. Bei diskreter Semantik hat ein Timed Automaton nur endlich viele Konfigurationen, da nur ganzzahlige Uhrenwerte betrachtet werden und für jede Uhr alle Uhrenwerte, die größer sind als die größte mit der Uhr verglichene Konstante, zu einem Wert zusammengefaßt werden. Da die Menge der zeitabstrakten Läufe bei diskreter und kontinuierlicher Semantik gleich ist, kann die Erreichbarkeitsanalyse die diskrete Semantik verwenden. Analoge Resultate für andere Modelle finden sich in [HMP92] und [AMP98].

Sei $A = (L, L^0, X, \Sigma, I, E)$ ein abgeschlossener Timed Automaton. Die Menge der ganzzahligen Uhrenbelegungen $V_I(X)$ ist definiert als Menge der Abbildungen, die jeder Uhr $x \in X$ einen Wert aus $\{0, 1, ..., C(x), C(x)+1\}$ zuweisen. Dabei ordne die Funktion C jeder Uhr $x \in X$ die größte Zahl zu, mit der sie in Uhrenbedingungen von A verglichen wird. Für ein $\delta \in \mathbb{N}$ und eine Uhrenbelegung $v \in V_I(X)$ bezeichnet $v +_I \delta$ die Uhrenbelegung von X, die jeder Uhr x das Minimum von $v(x) + \delta$ und $C(x) + 1$ zuweist.

Die diskrete Semantik von A ist das Transitionssystem $(L \times V_I(X), L^0 \times \{v_0\}, \Sigma \cup \mathbb{N}, \rightarrow)$. Die Elemente von $L \times V_I(X)$ werden als diskrete Konfigurationen von A bezeichnet. Wie bei der kontinuierlichen Semantik enthält $\rightarrow$ zwei Arten von Transitionen:

- Für $(s, v) \in L \times V_I(X)$, $\delta \in \mathbb{N}$ gilt $(s, v) \xrightarrow{\delta} (s, v +_I \delta)$, falls für alle $0 \leq \delta' \leq \delta$ die Invariante $I(s)$ von $v + \delta'$ erfüllt wird.
- Für $(s, v) \in L \times V_I(X)$, $(s, a, \varphi, \lambda, s') \in E$ gilt $(s, v) \xrightarrow{a} (s', v[\lambda := 0])$, falls v den Wächter φ erfüllt.

Satz: Die Menge der zeitabstrakten Läufe eines abgeschlossenen Timed Automaton A ist bei kontinuierlicher Semantik S und diskreter Semantik I gleich: $Run'_S(A) = Run'_I(A)$.

Beweisskizze: Sei $A = (L, L^0, X, \Sigma, I, E)$ ein abgeschlossener Timed Automaton mit kontinuierlicher Semantik $S = (Q_S, Q_S^0, \Sigma_S, \rightarrow_S)$ und diskreter Semantik $I = (Q_I, Q_I^0, \Sigma_I, \rightarrow_I)$.

Offensichtlich gilt $Run_I(A) \subseteq Run_S(A)$ und damit $Run'_I(A) \subseteq Run'_S(A)$.

Zu zeigen bleibt, daß es für jeden Lauf $(s_0, v_0), (s_0, v_1), (s_1, v_2), (s_1, v_3), ..., (s_k, v_{2k})$ in $Run_S(A)$ einen Lauf $(s_0, v'_0), (s_0, v'_1), (s_1, v'_2), (s_1, v'_3), ..., (s_k, v'_{2k})$ in $Run_I(A)$ gibt. Für einen Lauf $(s_0, v_0), (s_0, v_1), (s_1, v_2), (s_1, v_3), ..., (s_k, v_{2k})$ aus $Run_S(A)$ gelte $(s_i, v_{2i}) \xrightarrow{\delta_i}_S (s_i, v_{2i+1})$ und $(s_i, v_{2i+1}) \xrightarrow{a_i}_S (s_{i+1}, v_{2i+2})$ für $0 \leq i < k$. Wir definieren $\delta'_i := \lfloor \sum_{l=0}^{i} \delta_l \rfloor - \sum_{l=0}^{i-1} \delta'_l$ für $0 \leq i < k$. Dann läßt sich durch Induktion über i zeigen, daß für alle $0 \leq i < k$ gilt $(s_i, v'_{2i}) \xrightarrow{\delta'_i}_I (s_i, v'_{2i+1})$ und $(s_i, v'_{2i+1}) \xrightarrow{a_i}_I (s_{i+1}, v'_{2i+2})$ mit $v'_0 = v_0$ und geeigneten $v'_1, ..., v'_{2k}$.

Der Induktionsbeweis beruht darauf, daß für $0 \leq i \leq 2k$ und alle Uhren $x \in X$ gilt $v_i(x) - 1 < v'_i(x) < v_i(x) + 1$. Da alle Uhrenbedingungen abgeschlossen sind, nur ganzzahlige Konstanten enthalten, und v'_i ganzzahlig ist, erfüllt v'_i alle Uhrenbedingungen, die v_i erfüllt.

Folgerung: Die Menge der erreichbaren Zustände eines abgeschlossenen Timed Automaton A ist bei kontinuierlicher Semantik S und diskreter Semantik I gleich: $Reach'_S(A) = Reach'_I(A)$.

Wir können die Erreichbarkeitsanalyse abgeschlossener Timed Automata also auf die diskreten Konfigurationen beschränken. Deren Anzahl ist $|L| \cdot \prod_{x \in X}(C(x) + 2)$. Auch wenn die Größe der BDD-Darstellung von Konfigurationsmengen nicht proportional zu deren Kardinalität ist, ist praktisch eine Abhängigkeit zu beobachten. Werden die Konstanten, mit denen die Uhren verglichen werden, beispielsweise verdoppelt, so vergrößert sich die Zahl der diskreten Konfigurationen etwa um den Faktor $2^{|X|}$. Die BDD-Darstellung der Menge der erreichbaren Konfigurationen wächst oft ähnlich stark, obwohl sich die Menge der zeitabstrakten Läufe nicht verändert.

Eine Möglichkeit zur Verkleinerung der Konstanten ist das Teilen aller Konstanten durch ihren größten gemeinsamen Teiler. Die Menge der zeitabstrakten Läufe Run' des Automaten verändert sich dadurch nicht. Ist der größte gemeinsame Teiler der Konstanten eines Automaten A zu klein (z.B. 1), kann A durch geeignetes Erniedrigen von unteren und Erhöhen von oberen Schranken in einen Automaten A' mit höherem größten gemeinsamen Teiler transformiert werden [AIKY95]. Wegen $Run(A) \subseteq Run(A')$ gilt $Reach(A) \subseteq Reach(A')$, aus der Unerreichbarkeit eines Zustandes in A' folgt also die Unerreichbarkeit dieses Zustandes in A.

Mit einer ähnlichen Approximation kann die Erreichbarkeitsanalyse nicht abgeschlossener Timed Automata auf die Analyse der Menge der diskreten Konfigurationen reduziert werden. Dazu wird ein nicht abgeschlossener Timed Automaton A in einen abgeschlossenen Timed Automaton A' umgewandelt, indem alle Relationszeichen $<$ bzw. $>$ in Uhrenbedingungen durch $\leq$ bzw. $\geq$ ersetzt werden. Auch hier folgt aus der Unerreichbarkeit eines Zustandes in A' die Unerreichbarkeit dieses Zustandes in A.

3.4 Repräsentation von Konfigurationsmengen beliebiger Timed Automata

In diesem Abschnitt betrachten wir zwei weitere Verfahren, unendliche Mengen von Uhrenbelegungen zu repräsentieren, und vergleichen sie mit dem im letzten Abschnitt dargestellten Ansatz der Diskretisierung. Da die Verfahren nicht auf abgeschlossene Timed Automata beschränkt sind, wird dabei zugleich deutlich, welche Nachteile der Wegfall dieser Beschränkung bringt.

Grundlage für die Analyse von Timed Automata ist die Beobachtung, daß die unendliche Menge der Belegungen einer Uhrenmenge X in eine endliche Menge von Klassen partitioniert werden

kann, die als Regionen bezeichnet werden [AD94]. Bestehen zwei Konfigurationen aus dem gleichen Zustand und zur gleichen Region gehörenden Uhrenbelegungen, sind die Mengen aller von diesen Konfigurationen ausgehenden zeitabstrakten Läufe gleich.

Mengen von Regionen lassen sich als Konjunktionen und Disjunktionen von Uhrenbedingungen der Formen $x \sim c$ und $x - y \sim c$ darstellen, mit $x, y \in X$, $\sim \in \{<, >, \leq, \geq\}$ und $c \in \mathbb{N}$. Eine beliebige Konjunktion solcher Bedingungen läßt sich kanonisch als eine **Difference Bound Matrix** (DBM) genannte $(|X| + 1) \times (|X| + 1)$-Matrix darstellen [Dil89]. Im Gegensatz zu BDDs sind DBMs unempfindlich für die Erhöhung der Konstanten, mit denen die Uhren verglichen werden. Es gibt aber folgende Effizienzprobleme:

- Eine DBM repräsentiert nur Konjunktionen von Uhrenbedingungen. Disjunktionen von DBMs lassen sich im allgemeinen nicht als eine DBM darstellen, meist werden Listen von DBMs verwendet. Dies erhöht den Speicheraufwand, und, da die Darstellung nicht mehr kanonisch ist, die benötigte Rechenzeit für den Vergleich von Mengen.
- Die getrennte Darstellung von Zustand und Uhrenbelegung führt zu hohem Speicheraufwand, wenn sehr viele diskrete Zustände mit paarweise verschiedene Mengen von Uhrenbelegungen erreichbar sind.

Difference Bound Matrices werden von den Tools Uppaal [LPY97] und Kronos [DOTY96] verwendet. Abschnitt 4.1 enthält vergleichende Zeitmessungen.

Als zweite Möglichkeit kann auch bei nicht eingeschränkten Timed Automata die Menge der Uhrenbelegungen so **diskretisiert** werden, daß die Menge der zeitabstrakten Läufe erhalten bleibt [GPV94,ABK+97]. Dabei müssen, damit die Menge der diskretisierten Uhrenbelegungen mindestens einen Vertreter jeder Region enthält, als Uhrenbelegungen die Vielfachen von $1/(|X| + 1)$ betrachtet werden. Im Vergleich zur auf ganzzahlige Uhrenbelegungen beschränkten Diskretisierung erhöht sich der Speicheraufwand für die BDD-Darstellung der Menge der erreichbaren Konfigurationen ähnlich wie bei einer Erhöhung aller Konstanten in Uhrenvergleichen um den Faktor $(|X| + 1)$ (siehe Abschnitt 3.3). Außerdem sind die Operationen zur Berechnung der durch Transitionen entstehenden Konfigurationen aufwendiger.

4 Effiziente Implementierung der BDD-basierten Erreichbarkeitsanalyse

Nachdem wir im vorigen Abschnitt die Erreichbarkeitsanalyse von abgeschlossenen Timed Automata auf die Analyse endlicher Konfigurationsmengen abgebildet haben, stellen wir im folgenden zwei Möglichkeiten vor, die Effizienz dieser Analyse zu erhöhen. Dabei nutzen wir die hierarchische Struktur von CTA-Modellen.

4.1 Variablenordnung

Bei der Erreichbarkeitsanalyse eines CTA-Moduls werden Konfigurationsmengen des Produktautomaten seiner einzelnen Timed Automata verarbeitet. Eine diskrete Konfiguration eines solchen Produktautomaten besteht aus den Zuständen der einzelnen Automaten, den Werten der diskreten Variablen und ganzzahligen Werten der Uhren. Im folgenden bezeichnen wir den Zustand eines Automaten, diskrete Variablen und Uhren als Variablen.

Die Werte von Variablen mit einem Wertebereich der Kardinalität k werden durch $\lceil \log k \rceil$ Bit breite Binärzahlen kodiert. Damit können Konfigurationen durch Bitvektoren dargestellt werden. Es bleibt die Frage nach der Anordnung der Bits im Vektor.

Aus den in Abschnitt 3.2 erläuterten Regeln ergibt sich zunächst, daß die zur selben Variablen gehörenden Bits benachbarte Positionen einnehmen (Regel 1) und die Bits einer Variablen in der Reihenfolge absteigender Wertigkeit geordnet werden (Regel 2).

Die Anordnung der Variablen bestimmen wir aus der hierarchischen Struktur des CTA. Aus Regel 1 folgt, daß alle Variablen desselben Moduls benachbarte Positionen in der Variablenordnung einnehmen sollten. Wegen Regel 2 sollten sich globale Variablen in der Variablenordnung weiter oben als lokale Variablen befinden. Diese Anforderungen erfüllt jede Prefixlinearisierung des Hierarchiebaumes des CTA.

Prozesse	4	5	6	7	8	10	12	14	16	32	64
CTA Var.Ord. 1	0.3	0.4	0.8	1.3	2.3	4.0	8.9	13.6	22.7	208	1920
CTA Var.Ord. 2	0.3	0.6	1.6	3.9	9.4	46.3	249	MO			
CTA Var.Ord. 3	0.3	1.0	5.0	21.6	110	MO					
Uppaal	0.5	13.0	657	MO							
Kronos	3.0	191	MO								

Tabelle1. Rechenzeiten zur Berechnung der Erreichbarkeitsmenge von Fischer's Mutual Exclusion Protocol. Alle Angaben in CPU-Sekunden einer Sun Ultra 1 mit 200 MHz. MO bedeutet, daß mehr als 64 MB Speicher benötigt wurden.

Als Beispiel betrachten wir drei Variablenordnungen des in Abschnitt 2.3 beschriebenen CTA für Fischer's Mutual Exclusion Protocol mit n Prozessen. Tabelle 1 zeigt die Rechenzeiten für die Berechnung der erreichbaren Konfigurationen für diese Variablenordnungen. Zum Vergleich sind die Rechenzeiten der DBM-basierten Tools Uppaal und Kronos angegeben.

Variablenordnung 1: Eine Prefixlinearisierung der Hierarchie des CTA: (k, Zustand Prozeß 1, x_1, Zustand Prozeß 2, x_2, ..., Zustand Prozeß n, x_n).

Variablenordnung 2: Die globale Variable k wird, im Widerspruch zu Regel 2, an die letzte Position gesetzt: (Zustand Prozeß 1, x_1, ..., Zustand Prozeß n, x_n, k).

Variablenordnung 3: Die Verwendung von Difference Bound Matrices (siehe Abschnitt 3.4) entspricht einer Variablenordnung, bei der sich die Zustände der Automaten und diskreten Variablen vor den Uhren befinden, beispielsweise (k, Zustand Prozeß 1, ..., Zustand Prozeß n, x_1, ..., x_n). Diese Variablenordnung widerspricht Regel 1, da Zustand und lokale Uhr desselben Prozesses weit voneinander entfernte Positionen einnehmen.

Beim Vergleich von Uppaal und Kronos mit unserer BDD-basierten Implementierung ist zu beachten, daß die Rechenzeiten unserer Implementierung stark von den Größen der Konstanten in Uhrenvergleichen abhängig sind (siehe Abschnitt 3.3). Die Meßwerte zeigen aber, daß die Rechenzeit der BDD-Implementierung bei günstiger Variablenordnung nur in polynomieller Abhängigkeit von der Zahl der Prozesse wächst, im Vergleich zu mindestens exponentiellem Wachstum bei DBMs.

4.2 Modifizierte Breitensuche

Der im Abschnitt 3.1 angegebene Algorithmus zur Erreichbarkeitsanalyse führt eine Breitensuche in der Menge der erreichbaren Konfigurationen des Timed Automaton aus. Er kann auch als Fixpunktiteration gesehen werden, mit der Menge der erreichbaren Konfigurationen als Fixpunkt. Oft tritt das

Problem auf, daß die BDD-Repräsentationen von als Zwischenergebnis erhaltenen Konfigurationsmengen deutlich größer sind als die der gesamten Erreichbarkeitsmenge.

Die Ursache des Problems ist, daß während der Breitensuche zusätzliche Abhängigkeiten zwischen Variablen auftreten, BDDs aber von unabhängigen Variablen profitieren (siehe Abschnitt 3.2). Betrachten wir als Beispiel ein System aus zwei finiten Automaten A_1 und A_2, die nicht miteinander kommunizieren. Die Menge der erreichbaren Zustände des aus A_1 und A_2 komponierten Systems A ist $R(A) = R(A_1) \times R(A_2)$, die Größe des BDDs für $R(A)$ ist nur die Summe der Größen der BDDs für $R(A_1)$ und $R(A_2)$. Die Mengen der Konfigurationen, die A, A_1 bzw. A_2 von ihren Initialkonfigurationen aus mit höchstens k Transitionen erreichen können, seien $R_k(A)$, $R_k(A_1)$ bzw. $R_k(A_2)$. Dann ist $R_k(A) = \bigcup_{i=0}^{k}(R_i(A_1) \times R_{k-i}(A_2))$. Vor Erreichen des Fixpunktes gibt es also eine künstliche Abhängigkeit zwischen den erreichten Konfigurationen von A_1 und A_2.

Effizienter als die einfache Breitensuche ist eine modifizierte Breitensuche [BCL$^+$94]. Dabei wird die Menge der erreichbaren Konfigurationen eines Systems berechnet, indem solange Fixpunkte von Subsystemen berechnet werden, bis der Fixpunkt des gesamten Systems erreicht ist. Ein verbesserter Algorithmus zur Erreichbarkeitsanalyse eines CTA-Moduls berechnet also Fixpunkte der einzelnen enthaltenen Modulinstanzen und Timed Automata, bis der globale Fixpunkt (oder ein Element der Menge der Fehlerzustände) erreicht ist.

Literatur

[ABK$^+$97] Eugene Asarin, Marius Bozga, Alain Kerbat, Oded Maler, Amir Pnueli, and Anne Rasse. Data-structures for the verification of timed automata. In Oded Maler, editor, *Proceedings of the 1st International Workshop on Hybrid and Real-Time Systems (HART'97)*, LNCS 1201, pages 346–360, Berlin, 1997. Springer-Verlag.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AIKY95] R. Alur, A. Itai, R. P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118:142–157, 1995.

[Alu99] Rajeev Alur. Timed automata. In *11th International Conference on Computer-Aided Verification*, LNCS 1633, pages 8–22. Springer-Verlag, 1999.

[AMP98] Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits. In R. de Simone and D. Sangiorgi, editors, *Proceedings Concur'98*, LNCS 1466, pages 470–484, Berlin, 1998. Springer-Verlag.

[BCL$^+$94] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on CAD*, 13(4):401–424, April 1994.

[BR98] Dirk Beyer and Heinrich Rust. Modeling a production cell as a distributed real-time system with cottbus timed automata. In Hartmut König and Peter Langendörfer, editors, *Formale Beschreibungstechniken für verteilte Systeme (FBT'98)*, pages 148–159, June 1998.

[BR99] Dirk Beyer and Heinrich Rust. A formalism for modular modelling of hybrid systems. Technical Report 10/1999, BTU Cottbus, 1999.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, C-35(8):677–691, 1986.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212, Berlin, 1989. Springer-Verlag.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pages 208–219, Berlin, 1996. Springer-Verlag.

[GPV94] Aleks Göllü, Anuj Puri, and Pravin Varaiya. Discretization of timed automata. In *Proceedings of the 33rd IEEE conference on decision and control*, pages 957–958, 1994.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *Proceedings of the 19th International Colloquium on Automata, Languages, and Programming (ICALP 92)*, LNCS 623, pages 545–558, 1992.

[Lam87] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, October 1997.

[Min96] Shin-ichi Minato. *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Academic Publishers, Boston, 1996.