Formale Verifikation von Realzeit-Systemen mittels Cottbus Timed Automata

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Dr. rer. nat.)

genehmigte Dissertation

vorgelegt von

Diplom-Informatiker

Dirk Beyer

geboren am 15. Januar 1972 in Finsterwalde

Gutachter:	Prof. Dr. Claus Lewerentz
Gutachter:	Prof. Dr.-Ing. Monika Heiner
Gutachter:	Prof. Dr. Werner Damm
Tag der mündlichen Prüfung:	26. November 2002

Meinen Eltern

Danksagung

Da ich während meiner Tätigkeit in Cottbus nicht 'kontextfrei' gearbeitet habe und eine solche Arbeit schwerlich ohne die Hilfe und Unterstützung anderer entsteht, möchte ich mich bei all jenen bedanken, die mich geprägt haben und von denen ich lernen konnte. Bei vier Personen möchte ich mich speziell bedanken:

Mein Betreuer Claus Lewerentz sorgte für eine angenehme Atmosphäre in der Arbeitsgruppe. Seine ganzheitliche Interpretation einer Promotion als Weiterentwicklung der Persönlichkeit setzte er mit vielfältiger Unterstützung in den verschiedensten Bereichen um, wofür ich ihm sehr danke. Durch seinen Anleitungsstil und seine 'Softskills' motivierte er mich auch in aussichtslosen Situationen. In vielen Gesprächen über Forschung, Lehre und Privatleben konnte ich von ihm lernen.

Heinrich Rust gab mir den Anstoß zum Thema dieser Arbeit; in der Diskussion mit ihm entstanden die Cottbus Timed Automata. Er nahm sich immer Zeit für meine Probleme und gab mir Feedback zu meiner Arbeit. Ich danke ihm auch für die zahlreichen Gespräche über die 'unscharfen' Dinge des (wissenschaftlichen) Lebens.

Der intensiven Zusammenarbeit mit Andreas Noack, meinem ehemaligen Diplomanden und jetzigen Kollegen, verdanke ich wesentliche Resultate des 3. Kapitels. Meine Begeisterung für BDDs wurde durch ihn vertieft, und er gab mir viele seiner Erfahrungen mit dieser Datenstruktur weiter. Seine akribische Korrekturarbeit machte ihn zu einem wertvollen Partner.

Ganz besonders bedanke ich mich bei meiner Frau Simone: Täglich unterstützte sie mich, indem sie mir den Rücken von den Verpflichtungen des Alltags freihielt und mich in der Familie Halt finden ließ. Sie stand immer hinter meinen Vorhaben und stärkte mich mit Ermutigung und Zuspruch. Auch die gegen Null konvergierende Anzahl an Druckfehlern in dieser Arbeit habe ich ihr zu verdanken.

Inhaltsverzeichnis

Vorwort und Übersicht

Unsere Welt wird immer **komplexer** – oder besser: Wir selbst machen sie in unserem Automatisierungs- und Informationszeitalter immer schwieriger zu verstehen, indem wir immer komplexere Systeme konstruieren und in unseren Lebensalltag integrieren. ”Komplexität” ist zu einem Schlagwort geworden[1]. Manchmal sind Systeme aber gar nicht so kompliziert, wie sie erscheinen. Das Verständnis hängt davon ab, wieviel Ordnung des Systems erkennbar ist.

Eines der Schlüsselkonzepte zur Beherrschung der von uns zu lösenden Probleme ist das Erkennen und Nutzen von **Struktur**. Die beiden Hauptprobleme, mit deren Lösung sich die vorliegende Arbeit beschäftigt, sind die formalisierte Beschreibung von technischen Systemen (Modellierung) und der anschließende Nachweis der Korrektheit dieser Beschreibung (Verifikation).

Die vorgeschlagenen Lösungsansätze beruhen jeweils auf der Idee ”Strukturierung schafft Ordnung”. Bei der Modellierung bedeutet dies, die für das Problem relevanten Strukturelemente eines Systems explizit in der Beschreibung auszudrücken und das System-Modell somit verständlicher zu gestalten. Bei der Verifikation bedeutet es, die im Modell vorhandene Struktur für die Datenstrukturen und Algorithmen der vollautomatischen Analyse des Modells zu benutzen und somit eine effiziente Verifikation zu ermöglichen.

Im ersten Kapitel wird die Arbeit thematisch eingeordnet. Wichtige Konzepte werden eingeführt. Entsprechend der Verwendung in dieser Arbeit werden verschiedene System-Arten vorgestellt, und es wird definiert, was unter Spezifikation und Verifikation im Kontext formaler Methoden verstanden wird. Der Stand der Forschung bezüglich der Ansätze für die behandelten Hauptthemen wird referiert: Formalismen zur Modellierung, Datenstrukturen zur Repräsentation des Zustandsraumes und Werkzeugimplementierungen für automatische Verifikation. Offene Probleme werden identifiziert und ihnen die Lösungsvorschläge dieser Arbeit zugeordnet. Zuletzt wird der Schwerpunkt der Arbeit fokussiert, benachbarte Themengebiete werden abgegrenzt.

Aufgrund der Probleme der existierenden Modellierungsformalismen definiert Kapitel 2 einen erweiterten Formalismus zur strukturierten Modellierung: **Cottbus Timed Automata**. Zunächst werden Timed Automata formal definiert. Die Stärke des CTA-Formalismus liegt in der Anwendung eines **Modulkonzeptes**, was den *ersten Schritt zur*

[1] Der Begriff der Komplexität wird nicht nur in der Physik und Informatik, sondern neuerdings auch in völlig neuen Gebieten angewendet: Donald E. Knuth hat z. B. einen Aufsatz zur Komplexität von Liedern geschrieben [Knu84].

Bewältigung großer Systeme darstellt. Damit wird ein Modellierungsformalismus entwickelt, der die Vorteile der Timed Automata um die Möglichkeit der Konstruktion hierarchisch strukturierter Modelle für große Systeme erweitert. Über die klassische Interpretation der Timed Automata hinausgehend werden in dieser Arbeit weitere Semantiken formal einheitlich dargestellt, um die darauf aufbauenden Konzepte zu fundieren. Ein wesentlicher Beitrag liegt in der Einführung einer **ganzzahligen Diskretisierung** des kontinuierlichen Zustandsraumes der Timed Automata. Für die resultierende Semantik wird die Äquivalenz zur klassischen, kontinuierlichen Semantik bewiesen. Diese Semantik ist die Grundlage für die effiziente Verifikation.

Kapitel 3 beschäftigt sich mit der **effizienten Verifikation** von CTA-Modellen. Alle derzeit publizierten Ansätze zur algorithmischen Analyse von auf Timed Automata basierenden Modellen scheitern bereits bei kleinen Modellen am enorm hohen Aufwand an Speicher und Rechenzeit für die verwendeten Datenstrukturen und Algorithmen. Auf der Grundlage der in Kapitel 2 eingeführten ganzzahligen Diskretisierung wird die Datenstruktur **Binary Decision Diagram** zur Repräsentation genutzt, da sie die für die Erreichbarkeitsanalyse erforderlichen Operationen in effizienter Weise realisiert. Ein wesentlicher Beitrag dieses Kapitels ist ein spezielles Verfahren zum statischen Variablenordnen. Alle bisherigen BDD-basierten Ansätze zur Verifikation von Realzeit-Systemen lassen diesen bei der BDD-Anwendung so wichtigen Aspekt außer acht. Die im CTA-Modell vorhandene Struktur liefert aber gerade wichtige Informationen, die als Anhaltspunkte zur Berechnung einer guten Variablenordnung dienen können. Ausgehend vom Konzept des Kommunikationsgraphen wird eine obere Schranke für die Speicherkomplexität der Transitionsrelation angegeben und formal bewiesen. Diese Schranke wird in angepaßter Form als Schätzung für den Speicherbedarf der Erreichbarkeitsmenge umformuliert und für einen heuristischen Algorithmus zur Berechnung einer guten Variablenordnung eingesetzt. Dieses **schätzungsbasierte Variablenordnen** stellt eine wesentliche Neuerung im Realzeit-Bereich dar, ist der *zweite Schritt zur Bewältigung großer Systeme*. Bei sehr komplexen Modellen erreicht jedoch auch diese BDD-Repräsentation ihre Grenzen. Um dennoch die algorithmische Analyse von großen Systemen zu ermöglichen, wurde eine Verfeinerungsanalyse entwickelt. Diese Analyse dient dem Nachweis der Korrektheit von Verfeinerungsschritten bzgl. der Sicherheitseigenschaften. Durch die Bereitstellung dieser zusätzlichen Technik im Zusammenhang mit der Modulstruktur des Modells wird das **modulare Beweisen** ermöglicht, der *dritte Schritt zur Bewältigung großer Systeme*. Die Effizienz der BDD-basierten Verifikation hängt von einer Anzahl von Parametern ab, die im letzten Abschnitt des Kapitels ausführlich diskutiert werden: Repräsentation und Ordnung der Transitionsrelation, On-the-fly-Analyse, Behandlung von inaktiven Uhren und die Wahl der Hash-Funktion für die BDD-Operationen.

Für die Modellierung von hybriden Systemen existiert eine den Timed Automata verwandte Automatenklasse: die hybriden Automaten. Diese Automaten lassen sich problemlos in den im Kapitel 2 vorgeschlagenen CTA-Formalismus einbetten. Die Erweiterung des Formalismus um diese Automatenklasse wird in Kapitel 4 vorgestellt und am Beispiel verdeutlicht. Mehrere Automatenklassen, die der Mächtigkeit nach zwischen Timed Automata und hybriden Automaten liegen, werden vergleichend behandelt. Um eine scharfe Grenze zu den Timed Automata zu ziehen, werden Berechenbarkeitsresulta-

te angegeben. Für den erweiterten Modellierungsformalismus wird eine **kompositionelle Semantik** eingeführt. Die Semantik eines System-Modells kann auf der Basis der Semantik seiner Bestandteile definiert werden. Abschließend werden die Grundlagen für die algorithmische Analyse hybrider Modelle erörtert.

Der zweite, praktische Teil der Arbeit wurde anwendungsorientiert und weniger formal ausgeführt. Um die Ergebnisse empirisch validieren und Fallstudien durchführen zu können, wurde das **Werkzeug Rabbit** entwickelt, welches die behandelten Konzepte implementiert. Diese Werkzeugimplementierung wird in Kapitel 5 beschrieben. Bei der Entwicklung wurde folgendermaßen vorgegangen: Für die Modellierungsnotation des Werkzeugs (Eingabesprache) wurde eine repräsentationsunabhängige rechnerinterne Darstellung definiert. Ebenso wurden alle Grundfunktionen des Analysewerkzeugs in der Architektur eines Frameworks von der konkreten Repräsentation unabhängig implementiert. Dieses Framework kann mit beliebigen Repräsentationen ausgefüllt werden. In der aktuellen Version sind eine BDD-Bibliothek nach Kapitel 3 und eine DDM-Bibliothek nach Kapitel 4 integraler Bestandteil. Auf spezifische Besonderheiten der Repräsentationen wird eingegangen. Nachdem die Eingabesprache für Modell und Verifikation überblicksartig erläutert wurde, werden empirische Ergebnisse vorgestellt. Zur Demonstration des Umgangs mit hybriden Modellen werden einige Beispiel-Systeme aus der Literatur modelliert und verifiziert. Für den Bereich der Realzeit-Systeme, dem Schwerpunkt dieser Arbeit, werden die fünf meist-zitierten Benchmark-Modelle behandelt. Die CTA-Modelle werden beschrieben und für jedes Beispiel erfolgt eine Gegenüberstellung mit den Performance-Ergebnissen anderer bekannter Werkzeuge. Vom hier vorgestellten Werkzeug Rabbit werden alle Analyseaufgaben mit bedeutend besserer Performance durchgeführt. Bis auf ein Beispiel werden alle Modelle mit **polynomiellem Aufwand** verifiziert, wohingegen die bisherigen Werkzeuge mindestens exponentiellen Aufwand benötigen. Damit wird die Effizienz des vorgestellten Werkzeugs deutlich.

Im Kapitel 5 wurden für den Nachweis der Effizienz lediglich mehr oder weniger künstliche Benchmark-Modelle verwendet. Damit wird nachgewiesen, daß Rabbit für diese Beispiele effizienter als vergleichbare Werkzeuge ist. Der Bezug zur industriellen Praxis soll in Kapitel 6 durch die **Fallstudie** einer realen, umfangreichen Verifikationsaufgabe hergestellt werden: Ein formales Modell für die Fertigungsanlage des Lehrstuhls Software-Systemtechnik wird konstruiert und verifiziert. Die reaktiven Komponenten der Fertigungsanlage (ohne Zeit) allein weisen bereits den doppelten Umfang der KORSO-Fallstudie auf [LL95]. Nach einigen Ausführungen zu den Dokumenten und Prozeß-Phasen wird der Aufbau der Fertigungsanlage beschrieben. Wichtige Modellierungskonzepte werden verdeutlicht und einzelne Modell-Komponenten exemplarisch beschrieben. Abschließend wird die Verifikation einiger Anforderungen durchgeführt, wobei auch auf die Möglichkeit des modularen Beweisens eingegangen wird. Diese Fallstudie bestätigt, daß das im Rahmen dieser Arbeit entstandene Werkzeug Rabbit auch mit größeren Modellen aus der Praxis effizient und erfolgreich umgehen kann.

Kapitel 1

Thematische Einordnung

Eine Vision der Softwaretechniker ist es, eines Tages große Softwaresysteme *so konstruieren* zu können, daß Fehlerfreiheit in der gleichen Selbstverständlichkeit garantiert werden kann, wie das bei anderen technischen Systemen heute üblich ist. Die Motivation dafür entspringt nicht in erster Linie solchen Programm-Fehlern, die den Benutzern der Systeme in der täglichen Arbeit lästig werden, sondern die durch ihre Störung der Produktionsabläufe erhebliche Schäden in der Wirtschaft anrichten bzw. die durch Fehlverhalten sogar Menschenleben gefährden [Lev95, LT93].

Die Konstruktion von eingebetteten Systemen, die sehr starke Realzeit-Anforderungen zu erfüllen haben, ist eine immer bedeutsamer werdende Aufgabe in den verschiedensten Anwendungsbereichen, z. B. in der Medizin, in der Transporttechnik oder in der Produktionsautomatisierung. Um eine fehlerfreie Konstruktion solcher Systeme zu erreichen, müssen alternative Entwicklungsprozesse untersucht werden. Es geht darum, Fehler so früh wie möglich zu eliminieren oder gar nicht erst entstehen zu lassen. Formale Methoden bauen auf einer präzisen mathematischen Grundlage auf und führen der Erwartung nach zu einem Konstruktionsprozeß, dessen Produkte weniger Fehler haben. Dies tritt ein, weil die Eigenschaften der kritischen Systeme besser (da gründlicher und genauer) verstanden werden (vgl. [Rus94]).

Darüberhinaus werden diese sicherheitskritischen Systeme immer komplexer und die Aufgabe, fehlerfreie Steuersoftware zu programmieren, immer schwieriger. Deshalb beschäftigt sich diese Arbeit damit, einen geeigneten Modellierungsformalismus und effiziente Verifikationsverfahren zu entwickeln. Dabei wird ein Anwendungsgebiet betrachtet, in welchem nach Verfahren zur Konstruktion zuverlässiger Software zur Steuerung von Produktionsanlagen gesucht wird (siehe z. B. Abbildung 1.1 auf der nächsten Seite).

Die vorliegende Arbeit entstand in einer Forschungsgruppe am Lehrstuhl Softwaretechnik, in einem Arbeitsumfeld also, wo es neben der Behandlung von theoretischen Fragestellungen insbesondere um die Lösung praktischer Probleme geht. Deshalb werden die hier vorgeschlagenen Konzepte auch in einer Werkzeugimplementierung validiert, die den Verifikationsprozeß in einer effizienten Weise unterstützen soll. Bei den Entwicklungsarbeiten des Werkzeugs werden neben dem Ziel, bestmögliche Performance zu erreichen, auch softwaretechnische Aspekte berücksichtigt, die z. B. zu einer stabilen Entwicklungsbasis mit flexibler Architektur des Werkzeugs an sich führen. Durch die Betrachtungswei-

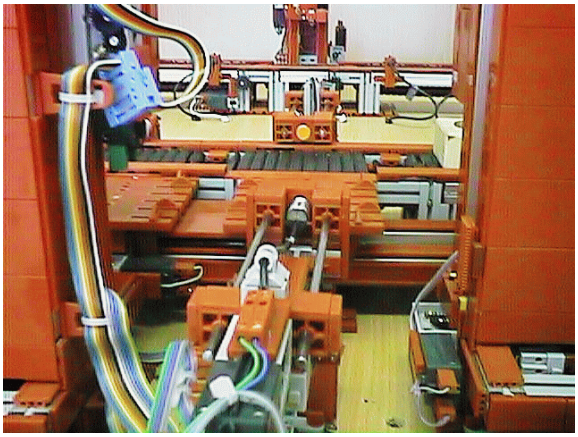


Abbildung 1.1: Blick auf das vom Autor für Versuchszwecke und Studentenpraktika aufgebaute Fischer-Technik-Modell.

se der Softwaretechnik ist auch der Schritt motiviert, abschließend anhand einer Fallstudie nachzuweisen, ob sowohl die Modellierungskonzepte als auch das Analysewerkzeug auf größere Systeme anwendbar sind.

Als Herausforderung für diesen Problembereich sei Bev Littlewood vom Zentrum für Softwarezuverlässigkeit der London City University zitiert:

> *" . . . if you want to build systems with ultrahigh reliability which provide very complex functionality and you want a guarantee that they are going to work with this very high reliability . . . you can't do it!"* *(zitiert nach [BH95])*

Zunächst wird eine Einordnung in das Fachgebiet der automatenbasierten Verifikation von Realzeit-Systemen gegeben. Die wesentlichen Konzepte werden eingeführt. Weiterhin ist Ziel des Kapitels, den Stand der Wissenschaft im behandelten Fachgebiet darzulegen und die Lösungsansätze sowie Herausforderungen auf diesem Gebiet zu vermitteln. Es geht um Modellierungsformalismen, Verifikationsverfahren und Datenstrukturen, bestehende Werkzeuge, hybride Ansätze und Fallstudien. Zuletzt werden die in der Arbeit behandelten Probleme gegen andere Bereiche abgegrenzt.

1.1 Allgemeine Begriffe und Definitionen

Eine zentrale Aufgabe, die sich der Mensch immer wieder stellt, ist das Verändern der realen Welt, in der er lebt. Da die reale Welt viel zu kompliziert ist, um sie verstehen zu können, betrachtet er stets *Teile der realen Welt* unter bestimmten, eingeschränkten Sichtweisen. Durch Abstraktion entsteht aus dem realen System 'Welt' ein Modell der Welt. Um Mehrdeutigkeiten aus dem Weg zu gehen, werden im folgenden die erforderlichen Grundbegriffe definiert und daraus abgeleitete Begriffe angegeben. Die Begriffsbildung erfolgt in starker Anlehnung an Definitionen aus der Literatur [CGP99, Bal96b, Bal98],

jedoch wurden diese angepaßt, um ein konsistentes und auf den Fokus dieser Arbeit zugeschnittenes Begriffsgerüst festzulegen.

Ein **System** *ist eine Menge von Komponenten, die miteinander kommunizieren und sich gegenseitig beeinflussen können. Systeme können mit ihrer Umgebung kommunizieren, das Systemverhalten kann von Größen der Umgebung abhängen, sowie Größen der Umgebung verändern. Solche Systeme werden häufig auch* **offene Systeme** *genannt. Bei einem* **geschlossenen System** *handelt es sich um ein System, das nicht mit der Umgebung kommuniziert.*

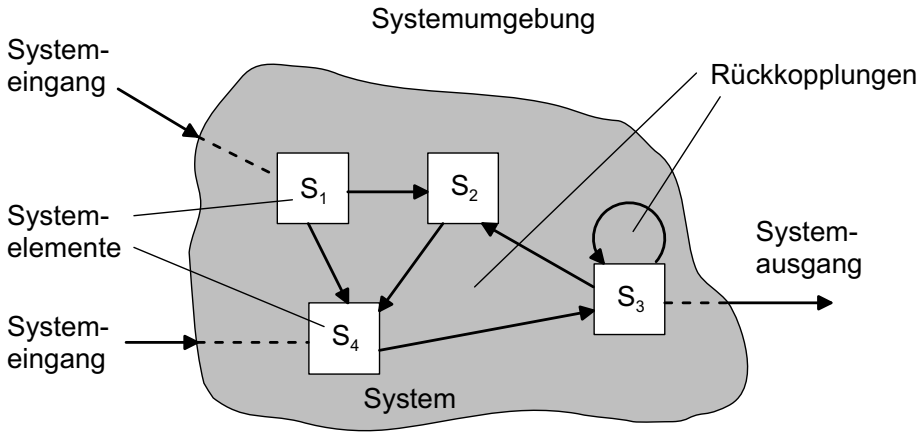


Abbildung 1.2: Grundlegende Systembegriffe (nach [Pag91], S. 3).

In der System-Theorie wurde der Begriff des Systems ganz allgemein als ”Menge von Elementen, die in Wechselbeziehung stehen” ([vB69], S. 55) definiert. Ein aus den Systemelementen S_1, S_2, S_3, S_4 zusammengesetztes System wird in der Abbildung 1.2 veranschaulicht.

Eine *Steuerung* (Controller, Steuerungssystem) ist ein in die Umgebung eingebettetes offenes System. Der Zweck eines solchen Systems besteht gerade darin, entsprechend der über beobachtete Größen (Systemeingänge) festgestellten Konfiguration der Umgebung auf die Umgebung über zu steuernde Größen (Systemausgänge) einzuwirken. Es ist also kein primäres Ziel dieser Systeme, Größen zu berechnen oder Informationen zu verwalten. Ist eine Eingangsgröße von der Ausgangsgröße abhängig, wird von einem *Regelungssystem* gesprochen [Föl84]. Durch diese Rückkopplung wird die Regelgröße (die zu regelnde Ist-Größe) ständig gemessen und an den Soll-Verlauf angenähert. Ein Steuerungssystem, welches als Bestandteil eines größeren Systems in dieses eingebaut und fest verknüpft ist, wird deshalb auch als *eingebettetes System* bezeichnet.

Abstraktion *ist der Vorgang des Zusammenfassens und Vernachlässigens von Informationen. Abstraktion verallgemeinert Aussagen und bildet Klassen von Begriffen.*

Ein (System-) **Modell** *ist eine Beschreibung des Aufbaus und des Verhaltens eines Systems auf einer Abstraktionsebene, innerhalb der alle in Bezug auf den Modellzweck relevanten Eigenschaften des Systems berücksichtigt, die nicht relevanten Eigenschaften jedoch vernachlässigt werden.*

Von Minski wurde der Begriff noch allgemeiner definiert [Min65]: ”Ein Objekt A ist ein Modell eines Objektes B für einen Beobachter C, wenn der Beobachter A benutzen kann, um Fragen, die ihn über B interessieren, zu beantworten.”

In dieser Arbeit wird unter Modell grundsätzlich ein *formales Modell* verstanden, d. h. das Modell wird mittels mathematischer Formalismen definiert und Eigenschaften des Modells können mit mathematischen Berechnungen ermittelt werden. Modellierung ist der Prozeß, bei dem für ein bestimmtes System ein formales Modell entwickelt wird, wobei ein Wechselspiel von Abstrahieren und Definieren stattfindet.

Bei der Entwicklung von Steuerungen wird zunächst ein Modell der Steuerung entwickelt. Je nachdem, in welcher Weise die zu betrachtenden Größen von der Zeit abhängig sind, wird zwischen reaktiven Systemen, Realzeit-Systemen und hybriden Systemen unterschieden. Diese Begriffe deuten auf die Verwendung von Modellen verschiedener Abstraktionsstufen bzgl. der zeitlichen, kontinuierlichen Größen.

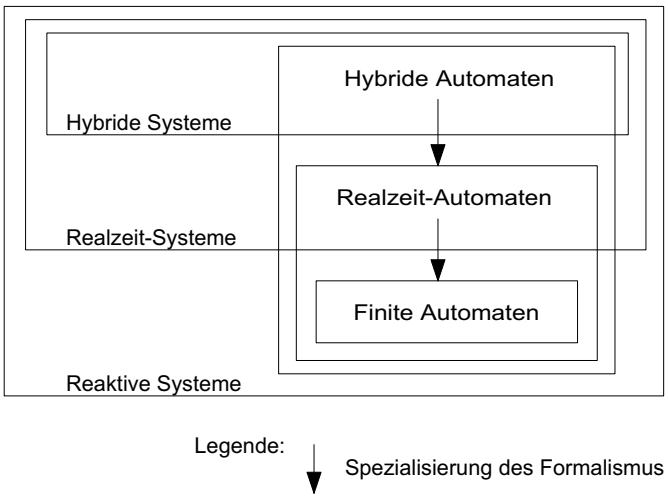


Abbildung 1.3: Hierarchie von Systemen mit unterschiedlicher Berücksichtigung zeitabhängiger Größen.

Bei einem *reaktiven System* kommt es auf die logisch richtigen Reaktionen auf Ereignisse aus der Umwelt in der richtigen Reihenfolge an; es werden qualitative Zeitaspekte betrachtet. Ein reaktives System terminiert nicht, sondern reagiert ständig auf seine Umgebung. Darüber hinaus gehört bei einem *Realzeit-System* der richtige Zeitpunkt der Reaktion zur Korrektheit des Systems; es werden quantitative Aspekte betrachtet. Werden bei einem System außer der Zeit noch weitere kontinuierliche Größen betrachtet, deren Ableitung nach der Zeit Werte ungleich eins annehmen darf, so wird von einem *hybriden System* gesprochen. Der Begriff ’hybrides System’ wird hier nicht in seiner allgemeinen Bedeutung (also als System, in dem zwei unterschiedliche Konzepte vereint werden) verwendet; hier ist vielmehr konkret die Modellierung einerseits von Zeiten und andererseits von Größen, deren Ableitung nach der Zeit nicht konstant eins ist, gemeint.

Nach Manna und Pnueli bilden reaktive Systeme, Realzeit-Systeme und hybride Systeme eine Hierarchie von Verfeinerungen [MP93], wie in Abbildung 1.3 auf der vorherigen Seite dargestellt. Die Einordnung eines Systems in eine dieser Klassen ist nicht vom System selbst abhängig, sondern von den betrachteten Aspekten bei der Modellierung (Abstraktion). Demnach ist ein Realzeit-System bzgl. seines Verhaltens ein spezielles reaktives System mit sich zeitlich verändernden Größen; ein hybrides System ist ein Realzeit-System, bei dem weitere kontinuierliche Größen modelliert werden können, deren Ableitung nach der Zeit verschieden von eins sein kann. Eine System-Art wird um so spezieller, je mehr Aspekte des Verhaltens des Systems eingeschränkt werden; zur Beschreibung eines Systems mit komplexeren Bedingungen ist jedoch ein ausdrucksstärkerer Formalismus notwendig. Mit dem mächtigsten Formalismus in der Hierarchie, dem hybriden Automaten, können Systeme jeder der drei Arten beschrieben werden. Die finiten Automaten hingegen können nur reaktive Systeme ohne zeitliche Aspekte beschreiben.

In Bezug auf eine Fertigungsanlage wie die von Abbildung 1.1 auf Seite 2 können demnach bei der Modellierung mit finiten Automaten die Aspekte des reaktiven Verhaltens wie Systemzustände und Reaktionen auf Ereignisse beschrieben werden. Realzeit-Aspekte, z. B. Zeitbegrenzungen für die Bearbeitung oder Reaktionszeiten, können mittels Realzeit-Automaten modelliert werden. Wird die Anlage schließlich als hybrides System aufgefaßt, können mittels hybrider Automaten auch Größen wie Geschwindigkeit oder Temperatur modelliert werden. Werden die Eigenschaften eines Systems betrachtet, so kann einem reaktiven System z. B. die Eigenschaft "Ereignis a tritt nach Ereignis b auf", einem Realzeit-System die Eigenschaft "a tritt nach Erreichen der Zeit t auf" und einem hybriden System die Eigenschaft "a tritt nach Erreichen der Temperatur ϑ auf" zugeordnet werden.

In der Literatur wird der Begriff 'Spezifikation' teilweise für jegliche System-Beschreibungen verwendet, die den Anspruch an Exaktheit haben. Hier soll zwischen einer möglichst abstrakten Beschreibung des Systems und den Beschreibung auf unterschiedlichen Abstraktionsebenen, von der abstrakten bis hin zu bereits implementierungsnahen Beschreibungen, unterschieden werden. Zweitere sind die Modelle, für erstere wird der Begriff 'Spezifikation' verwendet.

> *Die* **Spezifikation** *ist eine exakte Beschreibung der (nachzuweisenden) Eigenschaften eines Systems. Die Spezifikation bezieht sich dabei auf das (formale) Modell.*

Eine *formale Spezifikation* beruht auf einem Formalismus, d. h. die Beschreibung besitzt eine mathematisch präzise Semantik. Die Spezifikation beschreibt also Eigenschaften des Modells, nicht unbedingt des Systems (obwohl dies natürlich das Ziel ist). Sie kann verschiedenartige Eigenschaften des Modells beschreiben; in dieser Arbeit wird der Nachweis von Sicherheitseigenschaften mit einer elementaren Verifikationsmethode betrachtet, der Erreichbarkeitsanalyse.

> *Bei der* **formalen Verifikation** *wird überprüft (analysiert, bewiesen), ob das Modell die in der formalen Spezifikation fixierten Eigenschaften hat (Korrektheitsbeweis); falls ja, dann gilt das Modell als* **korrekt** *bzgl. der Spezifikation.*

Im Gegensatz zur Verifikation werden bei der Validierung die von der Anforderungsanalyse (Requirements Engineering) festgelegten Anforderungen (Needs) und Annahmen überprüft. Demnach ist die Verifikation lediglich eine Implikation[1] im folgenden Sinne: Falls alle Annahmen erfüllt sind und falls das System eine Implementierung des Modells ist, dann wird das System der Spezifikation entsprechend korrekt arbeiten.

Für zustandsbasierte Formalismen wird die Spezifikation häufig als (temporal-) logische Formel aufgefaßt. Wenn das entwickelte System-Modell ein Modell dieser Formel (im logischen Sinne) ist, dann erfüllt das System-Modell die Spezifikation. Das Verfahren zur Überprüfung dieser Eigenschaft wird *Model-Checking*[2] genannt und ist in den letzten 10 Jahren populär geworden, u. a. deshalb, weil es ein vollautomatisches Verfahren zur Analyse des Gesamt-Verhaltens des Modells ist.

Eigens zum Zweck der Beschreibung von Eigenschaften reaktiver Systeme wurden spezielle modale Logiken entwickelt: die Linear Temporal Logic (LTL) von Pnueli [Pnu77] und deren Weiterentwicklungen, z. B. die Computation Tree Logic (CTL) von Clarke und Emerson [CE81], sowie zeitbasierte Versionen [Koy92, ACD93] (zur Begründung siehe auch [Lam83b]).

Eine *formale Methode* entsteht, wenn verschiedene Phasen des Produkt-Entwicklungsprozesses durch die Anwendung von formalen Spezifikations- und Verifikationstechniken unterstützt werden. Verifikation ist nicht notwendigerweise Bestandteil der formalen Methode (Modelle oder deren Eigenschaften formal zu notieren, kann bereits stark zum Verständnis des Systems und somit zur Fehlervermeidung beitragen), aber die in einem Entwicklungsprozeß eingebettete formale Methode ist Voraussetzung für den sinnvollen Einsatz von formaler Verifikation in der Praxis. Die Software-Techniker wünschen sich die Unterstützung aller Phasen bis hin zur automatischen Generierung von ausführbarem Code.

1.2 Bestehende Ansätze, Probleme, Herausforderungen

Dieser Abschnitt führt in das Gebiet der Verifikation von formalen Modellen für Realzeit-Systeme ein. Für diese Arbeit besonders interessant sind diejenigen Ansätze, die eine werkzeugunterstützte Bearbeitung der Verifikationsaufgabe ermöglichen. Der Nachweis bestimmter Eigenschaften zustandsbasierter Modelle durch vollautomatische Verfahren (z. B. Erreichbarkeitsanalyse oder Model-Checking) steht im Vordergrund.

Die folgenden Abschnitte behandeln die folgenden Fragestellungen: Welche *Modellierungsformalismen* stehen in diesem Kontext zur Verfügung und welche Möglichkeiten bieten sie zur Entwicklung von Modellen? Wie können diese Modelle *verifiziert* werden, welche Ansätze bestehen zur algorithmischen Analyse sowie zur Repräsentation in Datenstrukturen und welche Probleme bestehen dabei? In welcher Weise wird die Ve-

[1] In den Ursprüngen der Logik wurde die Implikationsregel auch ”Phantasieregel”genannt, was die Bedeutung in gewisser Weise expliziter macht.

[2] Obwohl im allgemeinen versucht wird, deutsche Bezeichnungen zu verwenden, wird bei einigen Begriffen, die einen Eigennamen darstellen oder deren deutsche Bezeichnung nicht verbreitet ist, eine Ausnahme gemacht: z. B. Model-Checking, Binary Decision Diagrams, Timed Automata, Update-System.

rifikation von zustandsbasierten Realzeit-Modellen von *Werkzeugen* unterstützt, worin unterscheiden sich die verschiedenen Implementierungen? Können die Ansätze für die Realzeit-Systeme auch auf *hybride Systeme* übertragen werden, welche Modellierungsformalismen, Verifikationsstrategien und Werkzeuge sind in diesem Bereich vorhanden? Wie können die theoretischen Konzepte und praktischen Implementierungen bewertet werden, welche Benchmark-Beispiele existieren zum Vergleich verschiedener Werkzeuge, wurden die Ansätze mit großen *Fallstudien* validiert?

Bei der Einordnung und Diskussion dieser Fragen werden Arbeiten referiert, die auf die vorliegende Arbeit inspirierend gewirkt haben und auf deren Ergebnisse aufgebaut wird. Es wird der Kontext dieser Arbeit verdeutlicht. Für jeden der angesprochenen Themenbereiche werden offene *Probleme* definiert sowie *Forderungen* konkretisiert. Ein Ausblick auf die Inhalte der vorliegenden Arbeit wird durch die Formulierung entsprechender eigener *Lösungsvorschläge* gegeben, auf die dann jeweils in einem eigenen Kapitel ausführlich eingegangen wird.

1.2.1 Modellierungsformalismen

In gewöhnlichen Programmiersprachen wie z. B. Pascal ist im allgemeinen die totale Korrektheit nicht nachweisbar (da im allgemeinen nicht einmal die Terminierung eines Programmes beweisbar ist) [Tur37, HA72]. In einem genügend mächtigen formalen System sind nicht alle Eigenschaften (wahre Aussagen über das System) beweisbar [Göd31]. Daher wird die Mächtigkeit der Formalismen, die zur Modellierung von Systemen bzw. zur Formulierung von Eigenschaften eingesetzt werden, soweit eingeschränkt, bis der Nachweis einerseits überhaupt möglich und andererseits algorithmisch in einer möglichst effizienten Weise durchführbar ist.

Die **endlichen Zustandsautomaten** werden bereits seit einigen Jahrzehnten von Ingenieuren zur Modellierung verwendet. Grund dafür ist nicht nur die formale Fundierung, sondern auch, daß Zustände und Zustandsübergänge zusammen mit ihrer visuellen Darstellung als Graph ein natürliches und intuitives Beschreibungsmittel bilden. Vor rund 50 Jahren wurde mit der Untersuchung der Grundlagen der Automatentheorie begonnen [MP43, Kle56]. Für die unterschiedlichsten Zwecke wurden viele Varianten (z. B. Mealy-Automaten [Mea55], Moore-Automaten [Moo56], kommunizierende Automaten, Statecharts [Har87]) und verwandte Formalismen (z. B. Petrinetze [Pet62], Ereignis-Diagramme) geschaffen.

All diesen Ansätzen ist eine zustandsbasierte Semantik (meist als Transitionssystem) gemeinsam. Solange eine endliche Semantik existiert, kann die Verifikation durch vollautomatische Beweiswerkzeuge unterstützt werden. Für eine große Klasse von Modellen aus dem Bereich der reaktiven Systeme ist ein effizientes Model-Checking durch die Verwendung von Binary Decision Diagrams (BDDs) möglich [BCM$^+$92]. Traditionelle Techniken des Model-Checking unterstützen aber nicht die explizite Modellierung von Zeit. Deshalb sind diese für die Analyse von Realzeit-Systemen unzureichend, da deren Korrektheit davon abhängt, ob bestimmte Ereignisse zur richtigen Zeit auftreten. Durch die Hinzunahme von kontinuierlicher Zeit wird die natürliche Semantik unendlich. Soll

eine algorithmische Analyse ermöglicht werden, muß eine äquivalente endliche Semantik konstruiert werden.

Um neben reaktiven Systemen auch Realzeit-Systeme modellieren und verifizieren zu können, wurden verschiedene Formalismen um Konzepte wie Timer[3] und Uhren[4] erweitert. Der am weitesten verbreitete Ansatz, zeitliche Beschränkungen in ein Prozeßmodell zu integrieren, besteht darin, den Transitionen untere und obere Zeitschranken zuzuordnen. Beispiele dafür sind zeitbehaftete Petri-Netze [Ram74, MF76, BD91, Pop91, PZH97], Prozeß-Algebren wie Timed Communicating Sequential Processes (Timed CSP) [RR88] oder den Temporal Calculus of Communicating Processes nach Moller und Tofts [MT90], Timed Transition Systems [Ost90, HMP94], Timed I/O-Automata [LA90] und Modecharts [JM87]. Die vorliegende Arbeit basiert auf dem von Alur und Dill eingeführten Konzept der **Timed Automata** [AD90], welches auf einem von Dill bereits früher vorgeschlagenen Formalismus beruht [Dil90]. Dabei werden Büchi- bzw. Muller-Automaten [Büc60, Tho90] um eine endliche Menge von Uhren (Variablen, die ihren Wert mit dem Fortschreiten der Zeit kontinuierlich ändern) und um Zustandsübergänge mit Wächtern (logische Prädikate über diese Uhren) erweitert. Die Akzeptanzbedingung dieser Automaten erzwingt den Fortschritt bzw. die Unendlichkeit der zeitbehafteten Wörter (Timed Word, Timed Language). Mehrere parallel laufende Automaten können mittels Synchronisation nach dem Konzept der Communicating Sequential Processes (CSP) von Hoare kommunizieren [Hoa85]. Diese Automaten wurden sowohl auf der Ebene der formalen Sprachen [AD94], als auch auf der Ebene der algorithmischen Analyse gründlich untersucht [ACD93, HNSY94]. Um eine intuitive Notation zu erhalten, wurde in späteren Definitionen der Timed Automata die Akzeptanzbedingung durch eine Invariante (logisches Prädikat über die Uhren) ersetzt [Alu99].

Es folgt eine nicht-formale Beschreibung dieser Automatenklasse, um vorab einen Eindruck von den Timed Automata zu vermitteln. Ein Timed Automaton besteht wie ein finiter Automat aus einer endlichen Menge von Zuständen mit einer Teilmenge von Initialzuständen und einer Menge von Zustandsübergängen. Jeder Zustandsübergang ist mit einer Synchronisationsmarke (aus einer endlichen Menge) markiert. Darüberhinaus enthält der Timed Automaton eine endliche Menge von Uhren. Während sich der Automat in einem Zustand befindet, vergeht Zeit und die Werte der Uhren ändern sich dementsprechend. Ein Zustandsübergang verbraucht keine Zeit. Jedem Zustand ist eine Invariante (ein logisches Prädikat über die Uhrenbelegung) zugeordnet, die erfüllt sein muß, solange der Automat in dem Zustand bleibt. Wird die Invariante falsch, so muß der Automat den Zustand sofort verlassen. Jedem Zustandsübergang ist ein Wächter (ein logisches Prädikat über die Uhrenbelegung) zugeordnet, der erfüllt sein muß, um den Zustandsübergang zu ermöglichen, und eine Menge von rückzusetzenden Uhren. Damit ist der Wert jeder Uhr durch die seit ihrem letzten Rücksetzen vergangene Zeit bestimmt. Die Konfiguration (Situation) eines solchen Automaten ist bestimmt durch den Zustand und durch die Uhrenbelegung. Eine formale Definition der Timed Automata folgt in Kapitel 2.

[3] Bei einem Timer wird die Zeit von einem bestimmten Wert ausgehend rückwärts bis auf den Wert 0 gezählt, und nach Ablauf der festgelegten Zeitspanne wird ein Signal gegeben.

[4] Bei einer Uhr wird die Zeit vom Wert 0 ausgehend vorwärts gezählt und abgefragt, wieviel Zeit seit dem letzten Rücksetzen vergangen ist.

Bei der Modellierung mit Timed Automata besteht das Problem, daß die bestehenden Ansätze nicht für die Anwendung auf größere, reale Systeme konzipiert worden sind. Die praktische Anwendung in größeren Projekten erfordert die Integration von Aspekten aus der Softwaretechnik, d. h. Eigenschaften der Modelle wie Flexibilität und Verständlichkeit. In anderen Bereichen wurden diese Probleme durch das Einführen von Modulkonzepten, hierarchischer Strukturierung und der Benutzung von Abstraktionsschichten gelöst [Par72, PCW84].

Für einige Modellierungsnotationen, denen ein anderer Basisformalismus als Timed Automata zugrunde liegen, existieren bereits Ansätze zur hierarchischen Modellierung (z. B. Statecharts [Har87], SDL [BH89], Reactive Modules [AH96]). Teilweise wurden auch Konzepte für die Realzeit-Modellierung integriert. Inzwischen ist die erfolgreiche Modellierungsnotation Statecharts Bestandteil der Unified Modelling Language (UML). Der Formalismus berücksichtigt jedoch die kontinuierlichen Größen nicht in dem Maße wie z. B. Timed Automata, da die semantische Basis auf einer synchronen, zentralen Zeittakt-Steuerung beruht [Har87]. Bereits im Modell sind durch diesen synchronen Zeittakt nur diskrete Schritte denkbar. Es existiert keine kontinuierliche Semantik. Alle zu einem Zeitpunkt möglichen diskreten Transitionen werden synchron (gemeinsam) geschaltet. Um die Semantik für diesen Formalismus gab es lange Zeit Diskussionen; u. a. wurde eine Semantik durch die Werkzeugimplementierung festgelegt [HN96].

Problem 1 *In Experimenten zur Modellierung eingebetteter Realzeit-Systeme mit bestehenden, auf Timed Automata basierenden Notationen wurde das folgende Hauptproblem identifiziert: Die Modellierung von größeren Systemen ist nur schwer möglich, da ein Modell aus einer Menge von miteinander kommunizierenden Automaten besteht. D. h. bereits bei kleinen Beispiel-Modellen geht die dem System inhärente Struktur verloren und das Modell wird unübersichtlich und schwer zu verstehen. Das Modell kann nicht modular gegliedert werden, da die bestehenden Ansätze keine Strukturierung vorsehen.*

Forderung 1 *Die automatenbasierte Modellierung größerer Realzeit-Systeme muß praktisch möglich sein. In der Softwaretechnik bewährte Konzepte der Strukturierung müssen auch hier integriert werden. Einerseits muß die dem System innewohnende Struktur im Modell ausgedrückt werden können. Andererseits muß eine klare Modellgliederung erreicht werden, die den Modellierungsprozeß durch höhere Verständlichkeit des Modells unterstützt.*

Lösung 1 *Auf Timed Automata basierende Notationen sind durch ihre klar definierten semantischen Grundlagen sehr gut für die Modellierung von Realzeit-Systemen geeignet. Es wird ein diesen Formalismus erweiterndes Modulkonzept vorgestellt, mit dem verschiedene* **Abstraktionsebenen** *definiert und der Aufbau von* **Enthaltenseinshierarchien** *ermöglicht wird.*

Aufgrund der modularen Struktur der Modelle können ganze Systemteile durch andere ersetzt werden, da das Verhalten einer Komponente gekapselt ist und der Zugriff nur über eine klar definierte Schnittstelle erfolgen kann. Das Vorgehen der **schrittweisen Verfeinerung** *kann bei der Modell-Entwicklung eingesetzt werden. Grobe und verfeinerte Versionen des Modells können nebeneinander existieren und verifiziert werden.*

Diese Überlegungen führen zu einer neuen, modularen Notation für die Modellierung von Realzeit-Systemen: Cottbus Timed Automata (CTA). In Kapitel 2 wird die Definition dieser neuen Notation angegeben und es werden verschiedene Semantiken vorgestellt.

1.2.2 Verifikationsverfahren und Repräsentationen

In diesem Abschnitt werden zunächst verschiedene Vorgehensweisen bei der Verifikation von Realzeit-Modellen diskutiert und anschließend einige mit der Repräsentationsdatenstruktur verbundene Probleme erörtert. Grundsätzlich wird bei der Verifikation nach der Art der Formulierung der Spezifikation unterschieden zwischen Spezifikation mittels Erreichbarkeitsanweisungen und Spezifikation mittels Modell.

Spezifikation mittels Erreichbarkeitsanweisungen. Bei dieser Vorgehensweise werden Berechnungsschritte für die Verifikation in einer sequentiellen Anweisungssprache formuliert. Durch die Anweisungen werden die geforderten Eigenschaften überprüft. Typischerweise wird eine Menge von Konfigurationen definiert, die im Modell nicht erreicht werden dürfen, die Fehlermenge. Nachdem ausgehend von der Menge der Initialkonfigurationen die Menge aller erreichbaren Konfigurationen berechnet worden ist, kann geprüft werden, ob der Durchschnitt dieser Erreichbarkeitsmenge mit der Fehlermenge leer ist. Am Ergebnis kann das Analyseresultat abgelesen werden.

Der *Invariantentest* ist eine abgewandelte Form der Erreichbarkeitsanalyse: Statt zu spezifizieren, welche Konfigurationen *nicht erreicht* werden dürfen, wird eine Obermenge von Konfigurationen definiert, die im Modell *erreicht* werden dürfen (alle Konfigurationen, die eine Invariante erfüllen). Von den Erreichbarkeitsalgorithmen wird überprüft, ob nur solche Konfigurationen erreichbar sind, die durch die Invariante spezifiziert wurden.

Spezifikation mittels Modell. Im folgenden werden die wichtigsten Anwendungen der Methode ”Spezifikation mittels Modell” behandelt. Bei der Verifikation werden die Eigenschaften nicht durch reine Analyseanweisungen beschrieben und überprüft, sondern es erfolgt ein Wechselspiel zwischen Erreichbarkeitsanalyse, zusätzlichen Modellen und Verfeinerungsanalyse. Es werden vier Varianten unterschieden: Verifikation mittels Beobachtungsmodell und Erreichbarkeitsanalyse, Verifikation mittels Eigenschaftsmodell und Verfeinerungsanalyse, modulares Beweisen, und eigenschaftserhaltende Transformation. Diese Strategien sind verwandt und unterscheiden sich nur in dem Zweck und dem Vorgehen bei der Analyse des Modells.

Verifikation mittels Beobachtungsmodell und Erreichbarkeitsanalyse. Mittels Erreichbarkeitsanweisungen kann im allgemeinen nur die Erreichbarkeit bzw. Nichterreichbarkeit von Zuständen überprüft werden. Sicherheitseigenschaften, die nicht direkt durch Erreichbarkeitsanweisungen nachweisbar sind, können durch die Modellierung der Eigenschaften als ”Beobachtungsmodell” spezifiziert werden. Ein Beobachtungsmodell ist ein Modell, welches das Verhalten seiner Umgebung beobachtet, aber nicht beeinflußt.

Bei einem Beobachtungsmodell wird davon ausgegangen, daß ein abgeschlossenes System von außen ”beobachtet” wird. Es werden die für den Beobachtungszweck interessanten synchronisierten Schritte und Zeit-Schritte des beobachteten Modells registriert und der innere Zustand des Beobachtungsautomaten entsprechend geändert. Tritt das durch die Analyse gesuchte Verhalten auf, wird in einen speziellen Fehlerzustand gewechselt. Zur

Verifikation der durch den Beobachtungsautomaten modellierten Eigenschaft wird durch die Erreichbarkeitsanalyse lediglich überprüft, ob einer der Fehlerzustände des Beobachtungsmodells erreichbar ist.

Beim LTL-Model-Checking wird für die zu verifizierende Formel ein Automat erzeugt und überprüft, ob die vom Produktautomaten des Modell-Automaten und des (negierten) Formel-Automaten erzeugte Sprache leer ist. Ein Beobachtungsautomat kann einem ähnlichen Zweck dienen wie der Automat für die LTL-Formel beim LTL-Model-Checking.

Verifikation mittels Eigenschaftsmodell und Verfeinerungsanalyse. Eine Alternative zu dem o. g. Vorgehen bildet die Strategie der Verifikation durch Verwendung von abstrakten Modellen und der Verfeinerungsanalyse. Dabei werden die nachzuweisenden Sicherheitseigenschaften in einem Modell mit den gleichen Schnittstellenelementen modelliert, wobei das Modell eine Abstraktion des ursprünglichen Modells darstellt.

Bei der Verfeinerungsanalyse steht ein abstrakteres Modell einem verfeinerten Modell gegenüber, wobei das zweite ein spezielleres Verhalten als das abstrakte Modell haben soll. Es wird überprüft, ob das abstrakte Modell für jeden Schritt des verfeinerten Modells einen entsprechenden Schritt (mit der gleichen Synchronisationsmarke oder mit dem gleichen Zeitschritt) ausführen kann. Mit anderen Worten: es wird geprüft, ob das abstraktere Modell das verfeinerte Modell simulieren kann.

Der wesentliche Unterschied zwischen den letzten beiden Ansätzen besteht also im verwendeten Analysealgorithmus. Das Beobachtungsmodell wird zur Spezifikation von Eigenschaften verwendet, die das Modell erfüllen soll. Nun wird per Erreichbarkeitsanalyse ein System verifiziert, welches aus dem ursprünglichen und dem Beobachtungsmodell besteht. Beim Eigenschaftsmodell hingegen werden die nachzuweisenden Sicherheitseigenschaften in Form einer abstrakteren Version des Modells modelliert, wobei es in diesem Modell nur um das Festlegen der Eigenschaften geht, d. h. es wird so wenig wie möglich vom Systemverhalten modelliert. Die Verifikation erfolgt nun per Verfeinerungsanalyse, die sicherstellt, daß das konkrete Gesamtmodell des Systems eine speziellere Version des Eigenschaftsmodells ist. D. h. es darf im konkreten Systemmodell nicht *mehr* Verhalten als im Eigenschaftsmodell möglich sein.

Modulares Beweisen. Dieses Verfahren verwendet sowohl die Erreichbarkeitsanalyse als auch die Verfeinerungsanalyse. Ausgangspunkt ist meist, daß sich das fertige Systemmodell aufgrund seiner Komplexität nicht mehr vollautomatisch analysieren läßt. Es wird nach einer abstrakteren Version des Gesamtmodells gesucht, welches sich analysieren läßt. Dabei sind im Modell einige Details des konkreten Modells ausgeblendet. Einige Komponenten-Module sind durch abstraktere Versionen derselben ausgetauscht worden.

Ist die Erreichbarkeitsanalyse dieses abstrakteren Modells erfolgreich, so wird im zweiten Schritt für jedes ausgetauschte Komponenten-Modul nachgewiesen, daß das im konkreten Systemmodell enthaltene Modul eine Verfeinerung des in der Erreichbarkeitsanalyse benutzten, abstrakteren Moduls ist.

Im Gegensatz zum Eigenschaftsmodell handelt es sich bei den abstrakten Modellen in dieser Technik um Modelle von *Teilen* des Systems. Dieses abstraktere Modell spielt in der Verfeinerungsanalyse eine etwas andere Rolle: Das abstraktere Modell wird nicht als Hilfskonstrukt zum Überprüfen von Sicherheitseigenschaften des *Gesamtsystems* an-

gesehen, sondern als abstraktere Version eines *Teils* des eigentlichen Modells, welches die interessanten Sicherheitseigenschaften enthält, die bei der Verfeinerung erhalten bleiben müssen.

Eigenschaftserhaltende Transformation. Auch bei dieser Technik werden Erreichbarkeitsanalyse und Verfeinerungsanalyse verwendet, es existieren abstrakte und konkrete Modelle (von Teilen des Systems). Der Anwendungszweck besteht im Nachweis einer eigenschaftserhaltenden Modelltransformation (speziell: Verfeinerung). Es wird mit einem sehr abstrakten Modell begonnen und zunächst die Korrektheit dieses Modells per Erreichbarkeitsanalyse nachgewiesen. Der Kern dieser Strategie ist, daß das Modell nach jedem Verfeinerungsschritt korrekt sein soll. Es wird jeder Schritt der fortlaufenden Modellentwicklung auf Erhalt der Sicherheitseigenschaften hin überprüft. Diese Überprüfung erfolgt mittels Verfeinerungsanalyse, wobei jeweils eine neue Verfeinerung einer Komponente gegen die bisher im Modell befindliche Beschreibung dieser Komponente geprüft wird. Die Sicherheitseigenschaften des entstehenden Gesamtmodells selbst brauchen am Ende nicht mehr mittels Erreichbarkeitsanalyse nachgewiesen werden.

Repräsentation der Modelle. Für die Erreichbarkeits- und Verfeinerungsanalyse ist eine geeignete Datenstruktur zur Repräsentation von Transitionssystem und Erreichbarkeitsmenge auszuwählen, die die benötigten Operationen der algorithmischen Analyse (z. B. Nachfolgerberechnung) in einer möglichst effizienten Weise unterstützt.

Da eine Konfiguration eines Timed Automaton einerseits aus einer diskreten Komponente (dem diskreten Zustand des Automaten) und einer kontinuierlichen Komponente (der reellwertigen Belegung aller Uhren) besteht, sind die folgenden zwei Probleme zu lösen:

1. Die Semantik eines auf Timed Automata basierenden Modells beruht auf dem Produktautomaten aller im Modell enthaltenen Timed Automata. Daher wächst die Anzahl der potentiell zu repräsentierenden diskreten Zustände exponentiell mit der Anzahl der im Modell enthaltenen Automaten.

2. Der Wertebereich einer jeden Uhr ist die Menge der reellen Zahlen. Dadurch ist der zu repräsentierende kontinuierliche Zustandsraum zunächst unendlich groß. Durch die Technik der symbolischen Repräsentation müssen unendlich große Mengen in geeigneten endlichen Datenstrukturen dargestellt werden.

Eine zusätzliche Schwierigkeit besteht darin, daß die Zustandsräume nicht voneinander unabhängig sind. In den meisten Modellen hängen die Zustände der Automaten von den Uhrenbelegungen ab und umgekehrt. Dadurch ist es nicht ohne weiteres möglich, in effizienter Weise die diskreten Zustände z. B. durch Binary Decision Diagrams zu repräsentieren und die kontinuierlichen Teile des Zustandsraumes durch eine Matrix-basierte Datenstruktur. Hinzu kommt, daß zur Darstellung diskreter Zustandsräume vorteilhafte Datenstrukturen für die Repräsentation von kontinuierlichen Zustandsräumen nicht unbedingt vorteilhaft sind.

Ein Ansatz, den unendlichen kontinuierlichen Zustandsraum in einer endlichen Datenstruktur zu repräsentieren, besteht im Konzept des **Regionenautomaten**. Jede Region

repräsentiert hier eine Äquivalenzklasse von Konfigurationen. Zu jedem Timed Automaton existiert ein solcher Regionenautomat mit endlich vielen Regionen [AD90, ACD90]. Als Datenstruktur zum Speichern der Regionen werden spezielle Matrizen vorgeschlagen [Dil90], die je eine konvexe Menge von Konfigurationen repräsentieren: **Difference Bound Matrices** (DBM). In den bekannten und weitverbreiteten Werkzeugen zur Verifikation von Timed Automata (Uppaal [BLL+96] und Kronos [DOTY96]) wird der Ansatz verfolgt, die diskreten Zustände explizit aufzuzählen und Mengen von Uhrenbelegungen durch Matrix-basierte Datenstrukturen zu speichern. Die Verwendung von Difference Bound Matrices ist aber ineffizient bei der Darstellung nicht-konvexer Mengen von Uhrenbelegungen und bei einer großen Anzahl diskreter Zustände.

Die Datenstruktur **Binary Decision Diagram** (BDD) ist zur Repräsentation eines diskreten Zustandsraumes gut geeignet und in letzter Zeit populär geworden [BCM+92]. Ein rein BDD-basierter Ansatz, der auf einer Diskretisierung des kontinuierlichen Zustandsraumes beruht, wurde von Asarin, Bozga et al. mit der Datenstruktur **Numerical Decision Diagram** (NDD) verfolgt [ABK+97, BMPY97]. Diese Datenstruktur wurde auch in einer Werkzeugimplementierung realisiert, führte jedoch nicht zum erwünschten Erfolg.

Eine Kombination von DBMs und BDDs wurde erstmals von Wong-Toi vorgeschlagen [WT94]. Dabei werden Mengen diskreter Zustände durch BDDs repräsentiert und Mengen von Uhrenbelegungen durch DBMs. In der Arbeit von Balarin werden DBMs binär kodiert und BDDs zur Darstellung der Konfigurationsmengen genutzt [Bal96a]. Eine weitere Datenstruktur für die Repräsentation stellen die **Clock Difference Diagrams** (CDD) dar [LPWY99]. Durch ein CDD können erstmals nicht nur konvexe Mengen von Uhrenbelegungen, sondern auch nicht-konvexe Vereinigungen von Mengen von Uhrenbelegungen repräsentiert werden. Ein Vergleich von Behrmann, Larsen et al. weist nach, daß bei der Anwendung von CDDs der Speicherbedarf enorm reduziert werden kann, was jedoch gleichzeitig zum Anstieg der Rechenzeiten führt [BLP+99].

Die von Møller et al. vorgeschlagene Datenstruktur **Difference Decision Diagram** (DDD) stellt eine Variante der CDDs dar und versucht die Trennung der diskreten Zustände von den Uhrenbelegungen aufzuheben [MLAH99]. Für einen Performance-Vorteil dieser Datenstruktur werden keine aussagekräftigen empirischen Belege angegeben.

Durch alle bisher dargestellten Ansätze wurden die Probleme bei der Repräsentation des Zustandsraumes der Timed Automata nur teilweise gelöst.

Problem 2 *Durch die explizite Aufzählung der diskreten Zustände ist für die Speicherung (und somit auch für die algorithmische Behandlung) exponentieller Aufwand in Abhängigkeit von der Automatenanzahl im Modell erforderlich. Dies führt dazu, daß der Zustandsraum bereits bei kleinen Modellen (oft schon bei weniger als 10 Automaten) nicht mehr effizient repräsentiert werden kann.*

Bei der Repräsentation von Uhrenbelegungen mittels Matrizen entsteht für eine konvexe Menge von Uhrenbelegungen ein quadratischer Speicheraufwand in Abhängigkeit von der Anzahl der Uhren. Für jeden der exponentiell vielen diskreten Zustände wird eine Matrix für die Uhrenbelegungen abgespeichert. Falls gleiche Matrizen nicht mehrmals gespeichert werden (Anwendung kanonischer Matrizen), kann die Anzahl der Matrizen

kleiner sein als die der diskreten Zustände, jedoch führt dies bei nichttrivialen Modellen nicht zu ausreichenden Einsparungen.

Nicht-konvexe Mengen von Uhrenbelegungen können nur durch mehrere Matrizen bzw. aufwendig zu berechnende Datenstrukturen repräsentiert werden. Dadurch erhöht sich der Aufwand für die meisten Operationen erheblich.

Forderung 2 *Es ist eine geeignete Repräsentation zu finden, um*

- *das durch die Datenstruktur erzwungene exponentielle Wachstum der Repräsentation der diskreten Zustände in der Erreichbarkeitsmenge zu verhindern,*
- *auch nicht-konvexe Mengen von Uhrenbelegungen kanonisch zu repräsentieren und*
- *sowohl den diskreten als auch den kontinuierlichen Teil des Zustandsraums in einer einheitlichen Datenstruktur zu speichern.*

Lösung 2 *Für eine geringfügig eingeschränkte Teilklasse von Timed Automata wird eine* **ganzzahlige Diskretisierung** *der Uhrenbelegungen als Voraussetzung für eine effiziente Verifikation angegeben. Die Äquivalenz zwischen ganzzahliger Diskretisierung und Verwendung reellwertiger Uhrenbelegungen bzgl. der Erreichbarkeit diskreter Zustände wird formal bewiesen.*

Sowohl der diskrete als auch der kontinuierliche (nunmehr diskretisierte) Zustandsraum werden **einheitlich** *mittels Binary Decision Diagrams* **symbolisch** *repräsentiert. Auch nicht-konvexe Mengen werden* **kanonisch** *repräsentiert.*

Die neuen Konzepte für eine effiziente BDD-Repräsentation, die unter anderem ein schätzungsbasiertes Variablenordnen und auch die Berücksichtigung anderer wichtiger Parameter einschließen, werden in Kapitel 3 eingeführt.

1.2.3 Werkzeuge

Einige bekannte Verifikationswerkzeuge für Realzeit-Systeme werden kurz vorgestellt. Zunächst werden die speziell auf Timed Automata basierenden Werkzeuge genannt, im Anschluß daran werden einige Werkzeuge für verwandte Formalismen behandelt. Die Werkzeuge[5] für vollautomatische Analyse sind dabei von besonderem Interesse.

Eine ausführliche Gegenüberstellung der Werkzeuge Kronos, HyTech, Uppaal und anderen ist von Lötzbeyer durchgeführt worden [Löt96]. Ein weit gefaßter Überblick über den Stand der Verifikationsmethoden und Werkzeuge für reaktive Systeme wurde von Lewerentz und Lindner in einer vergleichenden Studie zusammengefaßt [LL95].

Das Werkzeug **Uppaal** wurde an den Universitäten Uppsala und Aalborg entwickelt [BLL+96, LPY97]. Es stellt eine Notation zur Modellierung zur Verfügung, in der das Systemmodell als Menge miteinander kommunizierender Prozesse aufgebaut ist. Der zugrundeliegende Formalismus ist eine Variante des Timed Automaton, die sich im

[5] Unter einem vollautomatischen Verifikationswerkzeug wird ein Werkzeug verstanden, welches das Analyseresultat bei Bereitstellen von Modell und Spezifikation ohne das Einbeziehen des Benutzers liefert.

wesentlichen durch drei Abweichungen von dem ursprünglichen Formalismus unterscheidet:

1. Die Synchronisation erfolgt nach dem Kommunikationskonzept von Milner [Mil89] über Kanäle, die in ihrer Richtung jeweils festgelegt sind.

2. Eine spezielle Variante der Uhren erlaubt die Modellierung von hybriden Größen, deren Ableitung nach der Zeit in einem festgelegten Intervall liegt.

3. Zur vereinfachten Modellierung von diskreten Größen werden auch diskrete Variablen angeboten. In den Zustandsübergängen können nicht nur die Uhren zurückgesetzt werden, sondern auch den diskreten Variablen neue Werte zugewiesen werden.

Eigenschaften werden als Formel einer Temporallogik spezifiziert. Als Analysemöglichkeiten stehen Erreichbarkeitstests über diesen temporallogischen Formeln und die Simulation zur Verfügung.

Kronos wurde im Forschungslabor VERIMAG in Grenoble entwickelt [DOTY96, Yov97]. Die Modellierung eines Systems erfolgt als Menge von Automaten. Es wird die ursprüngliche Variante der Timed Automata verwendet. Die Kommunikation der Automaten untereinander wird über Synchronisationsmarken nach dem CSP-Konzept von Hoare realisiert [Hoa85]. Es werden keine diskreten Variablen zugelassen, d. h. alle diskreten Größen müssen mittels expliziter diskreter Zustände der Automaten modelliert werden. Zur Modellierung von kontinuierlichen Größen stehen rücksetzbare Uhren zur Verfügung, also analoge Veränderliche, deren Ableitung nach der Zeit konstant eins ist und die auf null rückgesetzt werden dürfen. Die Spezifikation wird in einer temporallogischen Formel angegeben und mittels Model-Checking analysiert.

Rein **BDD-basierte Ansätze** wurden bisher nur bei **Kronos** verfolgt [ABK$^+$97, BMPY97]. Das Werkzeug wurde prototypisch implementiert, jedoch nicht der Öffentlichkeit freigegeben. Für Performance-Vergleiche standen nur Meßergebnisse aus Aufsätzen der Kronos-Forschungsgruppe zur Verfügung. Die aktuelle Version von Kronos verwendet zur Repräsentation von Uhrenbelegungen DBMs und zur Repräsentation der Zustände der Automaten BDDs [BDM$^+$98]. Die BDD-Implementierung von Kronos nutzt die entscheidenden Vorteile der BDD-Repräsentation nicht aus.

Auch das Werkzeug **Veriti** von Wong-Toi nutzt für die Erreichbarkeitsalgorithmen eine Kombination von DBMs und BDDs [WT94].

Ein weiteres auf BDD-basierte Ansätze beruhendes Werkzeug wurde von Wang vom Institute of Information Science (Taiwan) entwickelt: **RED** [Wan01]. Es wurde eine Datenstruktur benutzt, die von Wang Clock Restriction Diagram (CRD) genannt wird. Das Werkzeug ist jedoch ausschließlich für die Verifikation von symmetrischen Modellen mit einer skalierbaren Anzahl von Prozessen bestimmt [Wan00].

In der Werkzeug-Implementierung **Timed COSPAN** [AIKY95], welche im Bell-Forschungslabor von Alur et al. als Realzeit-Erweiterung des bestehenden Werkzeugs COSPAN [HK90] entwickelt worden ist, wird eine Sicherheitseigenschaft durch Sprach-Inklusion nachgewiesen. Die Sprache wird dabei definiert als Menge von Markierungsfolgen. Bei der Sprach-Inklusion wird überprüft, ob die Sprache, die vom Modell erzeugt

wird, eine Teilmenge der Sprache ist, die von der Eigenschaft erzeugt wird. Dieses Werkzeug bietet für die Berechnungen zwei Repräsentationen an: den Regionenautomaten und den Zonenautomaten. Die Repräsentation des Regionenautomaten beruht auf einer BDD-basierten Datenstruktur und die des Zonenautomaten auf DBMs [AK96].

Andere Werkzeuge nutzen die Möglichkeiten der Strukturierung, basieren jedoch auf Modellierungsformalismen, die sich von den Timed Automata unterscheiden. Die hierarchische Modellierung mittels Statecharts und die Simulation solcher Modelle wird seit vielen Jahren durch das Werkzeug **Statemate** unterstützt [HLN+90]. Da dieses Werkzeug auf einer synchronen Semantik beruht, wird es in dieser Arbeit nicht näher untersucht.

In den letzten vier Jahren entstand in Berkeley ein Werkzeug, welches auf dem Formalismus *Reactive Modules* [AH96] beruht und eine BDD-Repräsentation benutzt: **Mocha**. Der Formalismus unterstützt einen modularen Aufbau des Modells und bietet auch die Überprüfung einer Verfeinerungsrelation an, jedoch nur für Modelle ohne Realzeit-Aspekte. Eine frühere Version von Mocha (”cMocha”, C-Implementierung) unterstützte auch die Verifikation von Realzeit-Modellen [AHM+98], basierend auf dem Formalismus *Timed Modules* [AH97]. Die aktuelle Version von Mocha (”jMocha”, Java-Implementierung) [AdAG+01] ist für die vorliegende Arbeit uninteressant, da einerseits Realzeit-Aspekte auch für die Erreichbarkeitsanalyse nicht mehr unterstützt werden. Andererseits wird auch in dieser Version eine auf dynamischem Variablenordnen basierende BDD-Repräsentation verwendet, die die modulare Struktur des Modells nicht nutzt.

Salsa ist ein Werkzeug für die Verifikation von reaktiven Systemen ohne Realzeit-Aspekte [BS00]. Interessant an diesem Werkzeug ist, daß die Sicherheitseigenschaften durch pure Induktion über die Transitionsrelation nachgewiesen werden. Dadurch können die Analysen auch für große Modelle in Bruchteilen von Sekunden durchgeführt werden. Jedoch ist das Analyseresultat nicht exakt in der Hinsicht, daß bei sehr vielen Analysen ’Falsch-Positive’ erzeugt werden, d. h. die Analyse kann einen Fehler im Modell anzeigen, obwohl es korrekt ist. Dann kann auf diese grobe Art nicht mittels Induktion auf Korrektheit geschlossen werden. Der Anwender hat das ausgegebene Gegenbeispiel zu untersuchen und kann die zu analysierenden Eigenschaften verfeinern. Zeigt die Analyse keinen Fehler an, ist die Korrektheit des Modells bezüglich der verifizierten Eigenschaften nachgewiesen.

Bowen und Hinchey behandeln die Frage, warum den formalen Methoden der Durchbruch in der industriellen Praxis nicht gelingt. In ihrem Artikel deklarieren sie die Aussage, daß formale Methoden nicht von Werkzeugen unterstützt würden, als einen der vielen Mythen über formalen Methoden (siehe [BH95], Mythos 9). Auf den letzten Seiten wurde ausgeführt, daß selbst für den speziellen Bereich der zustandsbasierten Verifikation von Realzeit-Systemen eine Vielzahl von Werkzeugimplementierungen existieren. Welches Problem haben die Werkzeuge für die Verifikation von Realzeit-Systemen?

Die existierenden Werkzeuge zur Verifikation von Realzeit-Systemen können nur mit kleinen Beispiel-Modellen umgehen. Die Werkzeuge Kronos und Uppaal wurden mit auf Matrizen basierenden Repräsentationen implementiert; die bestehenden BDD-basierten Werkzeugimplementierungen bringen das volle Potential der BDD-Repräsentation nicht zur Geltung.

Problem 3 *Auf Timed Automata basierende Verifikation wurde bereits in mehreren Werkzeugen realisiert. Jedoch haben diese Werkzeugimplementierungen das Problem, daß sie bereits bei kleinen Modellen einen praktisch unakzeptablen Rechenzeit- und Speicherverbrauch aufweisen.*

Es ist schwierig, in bestehende Werkzeuge neue Datenstrukturen zur Repräsentation der Transitionssysteme einzubauen, um Experimente mit verschiedenen Repräsentationsvariationen durchführen zu können.

Forderung 3 *Die im Bereich der finiten Automaten erfolgreichen Techniken sollten, auf den Bereich der Timed Automata angewendet, auch gute Ergebnisse liefern. Durch das Anwenden von Diskretisierungen können auch BDDs als Datenstruktur verwendet werden. Die vollautomatische Verifikation von Timed Automata sollte mittels BDD-Repräsentation effizient durchgeführt werden können. Dabei müssen jedoch bestimmte Parameter (z. B. Variablenordnung, Repräsentation der Transitionsrelation) in einer die Effizienz der Analysealgorithmen positiv beeinflussenden Weise gewählt werden.*

Ein neues Werkzeug soll dem Entwurf nach so konzipiert sein, daß neue Datenstrukturen leicht einbezogen werden können und daß in Abhängigkeit vom verwendeten Modell zur Laufzeit die günstigste Repräsentation gewählt werden kann.

Lösung 3 *Im Rahmen dieser Arbeit wird ein effizientes Werkzeug entwickelt, das den Nachweis der Realisierbarkeit der vorgestellten theoretischen Konzepte erbringt. Bei einem Leistungsvergleich mit existierenden Werkzeugen ist für alle betrachteten Modelle eine erhebliche* **Leistungssteigerung** *erzielt worden. Für einige Modell-Klassen kann die Erreichbarkeitsanalyse in* **polynomieller** *Platz- und Speicher-Komplexität durchgeführt werden.*

Das Werkzeug für vollautomatische Verifikation von **modularen** *Realzeit-Modellen zeichnet sich durch den* **flexiblen** *Einsatz verschiedener Repräsentationsbibliotheken aus und kann somit unterschiedlichen vergleichenden Experimenten dienen.*

Es wird eine **eigene BDD-Bibliothek** *entwickelt, die sich von den herkömmlichen, allgemeinverwendbaren BDD-Bibliotheken darin unterscheidet, daß zur Vermeidung unnötigen Rechenaufwands keine zusätzlichen, nicht benötigten Fähigkeiten unterstützt werden.*

Das Problem des Variablenordnens wird in der Weise gelöst, daß die Variablen vor Beginn der Analyse durch eine **schätzungsbasierte Heuristik** *nach dem Prinzip des* **statischen Variablenordnens** *angeordnet werden. Bei der Schätzung wird die Kommunikationsstruktur im Modell berücksichtigt und somit das* **Wissen des Modellierers** *über die Zusammenhänge im Modell. Durch diese Technik wird eine sehr gute Kompression des Zustandsraumes erreicht.*

Zusätzlich zur Erreichbarkeitsanalyse wird eine effiziente Implementierung der Überprüfung einer **Simulationsrelation** *vorgestellt, mit der es in Kombination mit der Erreichbarkeitsanalyse möglich ist, große Modelle zu analysieren. Durch diese Erweiterung können verschiedene Verifikationsstrategien angewendet werden.*

Die flexible Architektur und modulare Notation der neuen Werkzeugimplementierung Rabbit wird in Kapitel 5 vorgestellt. Darüber hinaus wird über die Anwendung der Kon-

zepte auf mehrere Benchmark-Beispiele berichtet, und es werden Performance-Vergleiche durchgeführt.

1.2.4 Modellierung und Verifikation hybrider Systeme

Da in der industriellen Praxis auch hybride Systeme eine Rolle spielen [MN99], wurden auch für diesen Bereich Modellierungsformalismen, Verifikationsalgorithmen und Datenstrukturen sowie Werkzeuge entwickelt. Auch wenn für die Verifikation hybrider Systeme aufgrund der Komplexität ihrer Modelle momentan keine effizienten Verifikationsalgorithmen zur Verfügung stehen, sollen hier einige Ansätze vorgestellt werden.

Modellierungsformalismen. Hybride Systeme können mit dem Formalismus der **hybriden Automaten** (Hybrid Automata) modelliert werden, die von Henzinger definiert und untersucht worden sind [Hen96, ACHH93]. Dieses Automatenmodell stellt eine Erweiterung der Timed Automata dar: neben Uhren sind auch kontinuierliche Größen erlaubt, deren Ableitung nach der Zeit ungleich eins ist. Für diese Modellklasse ist die Erreichbarkeitsanalyse im allgemeinen nicht entscheidbar. Da die algorithmische Analyse für viele Systemmodelle terminiert und somit sinnvoll ist, ist sie theoretisch fundiert untersucht worden [ACH$^+$95].

Die Modellierungssprache **Charon** basiert auf hierarchischen hybriden Modulen [AGH$^+$00]. Derzeit existiert keine Werkzeug-Realisierung für die Verifikation solcher Modelle. **Masaccio** ist ein Formalismus für hybride dynamische Systeme, der die Modellierung von kontinuierlichen Komponenten erlaubt und auf Differentialgleichungen basiert [Hen00]. Auch für diesen Formalismus liegt keine Werkzeugunterstützung vor.

Bei **HyCharts**, einer von Grosu, Stauner und Broy entwickelten Modellierungsnotation [GSB98], handelt es sich um eine graphische Beschreibungstechnik für hybride Systeme, die eine hierarchische Strukturierung des Modells erlaubt. In Anlehnung an Statecharts ist ein HyChart ein Graph, dessen Knoten wiederum Graphen enthalten können und somit eine Hierarchie aufgebaut werden kann. Auch die Semantik lehnt sich an Konzepte von Statecharts an. Für die Verifikation muß ein solches Modell in die Notation der hybriden Automaten überführt werden; ein Werkzeug für die direkte Verifikation von HyCharts steht nicht zur Verfügung.

Verifikationsverfahren und Repräsentation. Bei der Verifikation hybrider Modelle steht die Erreichbarkeitsanalyse im Vordergrund. Da die Fixpunktiteration bei der Berechnung der Menge aller erreichbarer Konfigurationen im allgemeinen nicht terminiert, werden nur elementare Erreichbarkeitsoperationen in Form einer Anweisungssprache erlaubt.

Für die Verifikation hybrider Modelle (basierend auf hybriden Automaten) kann die DBM-Repräsentation nicht verwendet werden, da die Ableitungen der einzelnen analogen Variablen durch beliebige lineare Ausdrücke festgelegt werden können. Daher wird die **Double Description Method** (DDM) verwendet [FP96]. Es handelt sich um eine duale Repräsentation von konvexen Polyedern, bei der eine Matrix eine Menge von Einschränkungen (Constraints) und somit einen Durchschnitt repräsentiert, die andere Matrix einen durch eine Menge von Strahlen aufgespannten Raum und somit eine Vereinigung.

Die mengentheoretischen Operationen Vereinigung und Schnitt sind effizient, nach jeder Operation muß die Dualität wieder hergestellt werden.

Werkzeugunterstützung. Das Werkzeug **HyTech** wurde an der Universität Berkeley in Zusammenarbeit mit dem Intel-Entwicklungslabor Hillsboro entwickelt [HHWT95b]. Es beruht auf dem Formalismus der hybriden Automaten, neben diskreten Variablen und Uhren werden auch Variablen zur Modellierung solcher kontinuierlichen Größen zugelassen, deren Ableitung nach der Zeit ungleich eins ist (analoge Variablen). Diese Ableitungen werden in jedem diskreten Zustand des Automaten durch lineare, prädikatenlogische Ausdrücke festgelegt. Beim Übergang in einen anderen Zustand kann sich die Ableitung also verändern. Die Variablentypen "Diskret", "Uhr", "Stoppuhr" und "Analog" sind zugelassen. Durch eine analoge Variable kann eine physikalische Größe modelliert werden, die kontinuierlich ihren Wert ändert. Diese Wertänderung ist bestimmt durch die Ableitung der Variablen nach der Zeit, die auf ein bestimmtes Intervall festgelegt wird. Uhren stellen analoge Variablen dar, bei der die Ableitung immer konstant eins betragen muß. Die Stoppuhren sind analoge Variablen, bei denen die Ableitung entweder null oder eins ist. Größen, deren Ableitung konstant null sein soll, werden durch diskrete Variablen modelliert. Alle Variablen dürfen in Zuweisungen bei den Zustandsübergängen beliebig gesetzt werden. Die Synchronisation der Automaten wird über Synchronisationsmarken nach dem CSP-Konzept von Hoare realisiert [Hoa85]. Die mit Uppaal und Kronos beschreibbaren Systeme sind somit auch mit HyTech beschreibbar, darüberhinaus bietet HyTech weitere Modellierungsmöglichkeiten für hybride Systeme. HyTech beinhaltet Algorithmen zur Erreichbarkeitsanalyse und zur parametrischen Analyse.

Mit HyTech steht ein Werkzeug zur Verifikation hybrider Systeme zur Verfügung, welches jedoch keine hierarchische Modellierung gestattet, und es existieren mehrere Modellierungsformalismen, die nicht durch Werkzeuge unterstützt werden.

Problem 4 *Es existiert kein Formalismus, der die hierarchisch strukturierte Modellierung hybrider Systeme erlaubt und gleichzeitig durch ein Werkzeug für die Verifikation unterstützt wird.*

Forderung 4 *Ebenso wie Timed Automata bilden die hybriden Automaten aufgrund ihrer klaren Semantik und gut untersuchten Theorie grundsätzlich eine gute Basis für Erweiterungen. Für hierarchische Modellierung muß nach Möglichkeiten der Erweiterung dieses Basisformalismus gesucht werden, so daß Modelle in einer strukturierten Weise beschrieben werden können.*

Es ist weiter zu untersuchen, ob die Verifikation von Realzeit- und hybriden Modellen in einem einzigen Werkzeug vereint werden kann.

Lösung 4 *In dieser Arbeit werden die Konzepte der* **Modularität***, die für Timed Automata zur Lösung von Problem 1 beitrugen, auf die Modellierung hybrider Systeme übertragen. Der so entstehende Formalismus erlaubt die Konstruktion von* **Enthaltenseinsbeziehungen** *und die übersichtliche Modellierung anhand verschiedener* **Abstraktionsebenen**.

Datenstrukturen und Verifikationsalgorithmen werden untersucht und in einer **Werkzeugimplementierung** *bereitgestellt. Jedoch wird für hybride Systeme kein separates*

Werkzeug entwickelt, sondern es erfolgt eine **Integration** *der für hybride Modelle erforderlichen Bestandteile als zusätzliche Repräsentation in das für Realzeit-Systeme entwickelte Werkzeug. Das Werkzeug wählt durch statische Analyse des zu verifizierenden Modells automatisch die* **adäquate Repräsentation** *aus.*

In Kapitel 4 werden die Erweiterung des Formalismus aus Kapitel 2 für die Modellierung hybrider Systeme vorgestellt und die zur Verifikation notwendigen Datenstrukturen untersucht. Neben der Präsentation der Möglichkeiten für Realzeit-Systeme wird in Kapitel 5 auch die Integration der Repräsentation und Verifikation hybrider Systeme ausführlich dargestellt.

1.2.5 Fallstudien

Nach der Entwicklung neuer theoretischer Konzepte müssen diese jeweils empirisch validiert werden, um den Nachweis zu erbringen, inwieweit sie bei der tatsächlichen Entwicklung von Modellen praktikabel sind. Weiterhin muß für eine Werkzeugimplementierung gezeigt werden, ob die Anforderungen an das Laufzeitverhalten erfüllt werden, d. h. welcher Rechenzeit- und Speicheraufwand für die algorithmische Verifikation notwendig ist.

Für den Vergleich der Performance einer Implementierung mit vergleichbaren Werkzeugen werden meist sogenannte Benchmark-Beispiele verwendet. Dabei handelt es sich um kleine vereinfachte Modelle von Systemen, die jeweils unterschiedliche Anforderungen an die Analysealgorithmen stellen. Eine wichtige Eigenschaft der Benchmark-Beispiele ist die Skalierbarkeit. Um die durch Rechenzeit und Speicherbedarf gesetzte Grenze der Modellgröße für die automatische Verifikation ermitteln zu können, werden Modelle unterschiedlicher Größe (meist bzgl. der Anzahl von Prozessen) genutzt. Im Kapitel 5 werden u. a. Modelle für Fischers Protokoll für gegenseitigen Ausschluß, eine AND-Schaltung, das Token-Ring-FDDI-Protokoll und das CSMA/CD-Protokoll als skalierbare Benchmark-Beispiele verwendet.

Der Vorteil solcher allgemein anerkannter und vielfach genutzter Beispiele ist die Möglichkeit der experimentellen Gegenüberstellung der Performance-Ergebnisse verschiedener Ansätze, da die Modelle oft schon in der entsprechenden Modellierungsnotation verfügbar sind. Die Benchmark-Modelle haben jedoch einige entscheidende Nachteile. Einerseits ist die Struktur dieser Modelle regelmäßig und einfach. Daher können die Möglichkeiten der strukturierten Modellierung an diesen Beispielen nur in begrenztem Umfang erprobt werden. Andererseits kommen bei diesen Beispielen bestimmte in der Praxis auftretende Aspekte nicht zur Geltung. In dem dieser Arbeit zugrundeliegenden Anwendungsgebiet der Steuerung von Fertigungsanlagen treten kompliziertere Strukturen bei der Modellierung auf und die Modelle sind aufgrund der starken Verflechtung innerhalb des Modells keineswegs skalierbar.

Ein gelungenes Beispiel für eine praxisnahe Problemstellung ist die KORSO-Fallstudie von Lewerentz und Lindner [LL95]. Diese Fallstudie wurde allgemein anerkannt, und in nahezu allen wichtigen Formalismen wurden Modelle konstruiert sowie Eigenschaften nachgewiesen. Diese Fallstudie beinhaltete keine Realzeit-Aspekte; es handelt sich um ein rein reaktives System. Im Rahmen des KORSYS-Projekts wurde eine

Aufgabenstellung für eine Fallstudie mit Realzeit-Aspekten entwickelt (”Production Cell II”, flexible Fertigungsanlage mit Realzeit-Eigenschaften [LM96]). Leider fand diese Fallstudie nicht die gewünschte Beachtung in den Forschungsgruppen.

Problem 5 *Zur Validierung der Performance von Werkzeugimplementierungen existieren Benchmark-Beispiele. Um jedoch die Modellierungskonzepte für größere Systeme validieren und die Verifikation von Modellen mit einer weniger regelmäßigen Struktur durchführen zu können, ist eine Fallstudie notwendig. Zwar wurden im Bereich der Timed Automata bereits Fallstudien durchgeführt (z. B. [BGK+96]), jedoch erreichen die Modelle nicht den geforderten Umfang.*

Forderung 5 *Für den Anwendungsbereich der Steuerung von Fertigungsanlagen soll eine große Fallstudie durchgeführt werden, an der die neu entwickelten theoretischen Konzepte zur hierarchischen Modellierung erprobt werden. Aspekte des Vorgehens bei der Entwicklung von größeren Modellen sind zu untersuchen. Darüberhinaus soll überprüft werden, ob die Werkzeugimplementierung auch zur Verifikation größerer Systeme geeignet ist.*

Lösung 5 *In dieser Arbeit werden die Ergebnisse eines umfangreichen Modellierungsprojektes vorgestellt. Es wird ein* **Modell für eine Fertigungsanlage** *entwickelt, welche die der KORSO-Fallstudie an* **Umfang und Komplexität übertrifft**. *Die Steuerung der Fertigungsanlage muß 44 Sensoren abfragen und 28 Motoren unter Berücksichtigung von Realzeit-Eigenschaften ansteuern. Das Modell wird unter Berücksichtigung von Aspekten der Softwaretechnik wie Flexibilität und Übersichtlichkeit durch den Einsatz von* **modularer Strukturierung** *und Verwendung verschiedener Abstraktionsschichten gestaltet.*

Es wird nachgewiesen, daß dieses Modell mit Hilfe der vorliegenden Werkzeugimplementierung verifiziert werden kann. Insbesondere das Ausnutzen der Struktur des Modells für die Datenstrukturen und Algorithmen führt (z. B. durch eine angemessene Variablenordnung in der BDD-Repräsentation) zu einer **effizienten Analyse**.

Die Fertigungsanlage sowie das strukturierte Modell dieser Anlage werden in Kapitel 6 vorgestellt. Dabei wird ein Überblick über die Architektur des Modells gegeben und die Vorgehensweise bei der Verifikation einiger Eigenschaften exemplarisch erörtert.

1.3 Abgrenzung

Ausgangspunkt der Arbeit am Lehrstuhl Software-Systemtechnik der Brandenburgischen Technischen Universität war der Gedanke, zuverlässige Steuerungssoftware für reaktive Systeme unter Berücksichtigung von Realzeit-Aspekten zu konstruieren. Nachdem festgelegt war, nicht nur formale Methoden bei der Modellierung zum Einsatz zu bringen, sondern auch automatische Verifikation durchzuführen, wurde der Formalismus Timed Automata als geeignet befunden und für die Modellierung ausgewählt, da bereits mehrere Werkzeuge zur Verfügung standen. Praktische Untersuchungen innerhalb der Forschungsgruppe haben ergeben [Rus99], daß existierende Ansätze einige gravierende Mängel aufweisen (siehe Probleme 1 - 5), die nur durch das Entwickeln einer neuen Notation und eines neuen Verifikationswerkzeugs beseitigt werden können.

Das Ziel der vorliegenden Arbeit ist es, die notwendigen theoretischen Grundlagen und neuen Konzepte zur Lösung der angesprochenen Probleme darzulegen. Darüberhinaus wird die Wirksamkeit der entwickelten Ansätze durch eine Werkzeugimplementierung nachgewiesen. Abschließend werden die Möglichkeiten zur Verifikation an einer größeren Fallstudie demonstriert.

Es soll also weder die Definition einer formalen Methode noch eines darauf zugeschnittenen Software-Entwicklungsprozesses gegeben werden. Vielmehr sollen wichtige Grundlagen geschaffen werden, um eine formale Methode durch Formalismen und Werkzeuge zu unterstützen. Die empirischen Untersuchungen dienen ausschließlich dem Nachweis der Leistungsfähigkeit bzw. der Grenzen des Werkzeugs, nicht dem Entwicklungsprozeß. Ein solcher Prozeß wird im Kapitel 6 zur Einordnung der behandelten Verifikationsmöglichkeiten grob vorgestellt, um einen Überblick zu geben, in welcher Art und Weise das Werkzeug Unterstützung liefert und welche Dokumente wann und wie entstehen. Die für den Bereich der formalen Verifikation interessanten Phasen – Modellierung und Verifikation – motivieren den Schwerpunkt dieser Arbeit. Die Untersuchung von Software-Entwicklungsprozessen stellt derzeit noch eine Herausforderung für die Forschung im Gebiet der formalen Methoden dar. Nicht zuletzt wegen der bis jetzt unzureichenden Resultate auf diesem Gebiet erlangen die Möglichkeiten der formalen Verifikation in der industriellen Praxis noch nicht die Anerkennung, die sie verdienen.

Kapitel 2

Modellierungsformalismus für Realzeit-Systeme: Cottbus Timed Automata

Dieses Kapitel beschäftigt sich zunächst mit den mathematischen Grundlagen, der Vorstellung des Timed Automaton am Beispiel und der formalen Definition des Basisformalismus. Die beiden zentralen Themen sind die Einführung eines neuen, **modularen Modellierungskonzeptes** und einer **Diskretisierung** des kontinuierlichen Zustandsraumes.

Damit die Modellierung von Realzeit-Systemen besser unterstützt und insbesondere die im ersten Kapitel genannten Forderungen an einen Modellierungsformalismus erfüllt werden können, wurde ein neuer, modularer Formalismus zur Modell-Beschreibung entwickelt: **Cottbus Timed Automata (CTA)**. Dabei wurde eine Zentralidee der Software-Technik mit einem Formalismus aus dem theoretischen Bereich der Informatik kombiniert: Große Systembeschreibungen werden durch kompositionelle Module mit festgelegten Schnittstellen hierarchisch strukturiert; das Verhalten des Systems wird mit dem mathematisch gut fundierten Konzept der Timed Automata definiert.

In Vorbereitung auf die Verifikation werden verschiedene Semantiken eingeführt. Ausgehend von der Standardsemantik werden die Erreichbarkeitssemantik, die ganzzahlige Semantik und die Spursemantik in einer einheitlichen Notation präsentiert. Die Äquivalenz der neu eingeführten, ganzzahligen Diskretisierung zur kontinuierlichen Standardsemantik bezüglich der erreichten Zustände wird formal bewiesen.

Der Schwerpunkt dieser Arbeit ist die Modellierung und Verifikation von *Realzeit-Systemen*. Die zur Modellierung verwendeten Konzepte, insbesondere die Modularität, lassen sich jedoch problemlos auf hybride Systeme übertragen. Auf die Modellierung und Verifikation hybrider Systeme wird im Kapitel 4 eingegangen.

2.1 Mathematische Notationen und Begriffe

Im folgenden werden einige Begriffe und Notationen eingeführt. Eine Uhr ist eine kontinuierliche Variable, deren Wert sich mit der Ableitung nach der Zeit eins erhöht. Uh-

renbedingungen werden benutzt zur Definition von Invarianten der Zustände sowie von Wächtern der Transitionen eines Timed Automaton.

Definition 2.1 *Sei $X = \{x_1, \ldots, x_n\}$ eine endliche Menge von Uhren. Ein Vergleich einer Uhr aus X mit einer Konstanten aus $\mathbb{N}$, der Menge der natürlichen Zahlen (einschließlich 0), heißt atomare Uhrenbedingung. Durch Konjunktion von atomaren Uhrenbedingungen entstehen* **Uhrenbedingungen**. *Formal wird die Menge $\Phi(X)$ aller Uhrenbedingungen über X durch die folgende Grammatik erzeugt:*

$$\varphi := \; x \sim c \mid \varphi \wedge \varphi \mid \mathit{true} \mid \mathit{false}$$

mit $x_i \in X$, $c \in \mathbb{N}$ und $\sim \in \{=, \leq, \geq, <, >\}$.

Uhrenbedingungen legen die möglichen Werte der Uhren auf der syntaktischen Ebene fest; auf der semantischen Ebene wird der Begriff der Uhrenbelegung benutzt.

Definition 2.2 *Eine* **Uhrenbelegung** v **von** X *ist eine totale Funktion $v : X \to \mathbb{R}_+$ von X in die Menge der nichtnegativen reellen Zahlen $\mathbb{R}_+$. $Val(X)$ ist die Menge aller Uhrenbelegungen von X.*

Zu jeder Uhrenbedingung kann durch die Anwendung der folgenden Abbildung die entsprechende Menge von Uhrenbelegungen gefunden werden.

Definition 2.3 *Die* **Semantik einer Uhrenbedingung** *ist gegeben durch die Interpretationsfunktion $[\![.]\!] : \Phi(X) \to 2^{Val(X)}$, die wie folgt induktiv definiert ist:*

$$\begin{array}{rcll} [\![x \sim c]\!] & := & \{v \in Val(X) \mid v(x) \sim c\} & \textit{(Ungleichung)} \\ [\![\varphi \wedge \varphi']\!] & := & [\![\varphi]\!] \cap [\![\varphi']\!] & \textit{(Durchschnitt)} \\ [\![true]\!] & := & Val(X) & \textit{(Volle Menge)} \\ [\![false]\!] & := & \emptyset & \textit{(Leere Menge)} \end{array}$$

mit $x \in X$, $c \in \mathbb{N}$ und $\sim \in \{=, \leq, \geq, <, >\}$. φ wird interpretiert als die Menge aller Uhrenbelegungen über X, die das logische Prädikat φ erfüllen. Bei festgelegter Ordnung der Uhren in X wird für $v \in Val(X)$ auch die Vektor-Schreibweise $\bar{v} = (v(x_1), v(x_2), \ldots, v(x_n))$ verwendet.

Die Uhrenbelegung, die allen Uhren den Wert 0 zuweist, wird mit v^0 bezeichnet. Für $v \in Val(X)$ und $\delta \in \mathbb{R}_+$ bezeichnet $v + \delta$ die Uhrenbelegung von X, die jeder Uhr x den Wert $v(x) + \delta$ zuordnet. Für $v \in Val(X)$ und $Y \subseteq X$ bezeichnet $v[Y := 0]$ die Uhrenbelegung von X, die jeder Uhr in Y den Wert 0 zuweist und alle anderen Uhren unverändert läßt.

Anmerkung: **Repräsentation von Belegungsmengen.** Belegungsmengen lassen sich durch mehrere verschiedene Repräsentationsformen darstellen. Einige davon sind in Abbildung 2.1 auf der nächsten Seite aufgeführt. Für jede Repräsentationsform kann jeweils eine Interpretationsfunktion definiert werden. So kann eine Menge von Belegungen

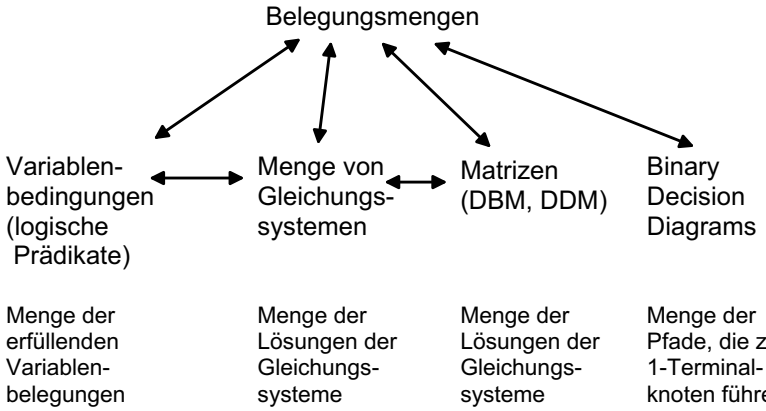


Abbildung 2.1: Semantisch isomorphe Repräsentationen von Belegungsmengen.

$V \subseteq Val(X)$ die Semantik sein für eine Bedingung (logisches Prädikat[1]) $\varphi \in \Phi(X)$ mit $[\![\varphi]\!] = V$, für eine Menge von Gleichungssystemen (Matrizen mit einer zusätzlichen Spalte für die Konstanten auf der rechten Seite können wiederum je ein Gleichungssystem repräsentieren) $\mathcal{G}$ mit $[\![\mathcal{G}]\!] = V$ oder für ein Binary Decision Diagram (Definition in Kapitel 3) $\mathcal{B}$ über X mit $[\![\mathcal{B}]\!] = V$.

Für ein S-Tupel der Länge n ist $\pi_i : S_1 \times \ldots \times S_n \rightarrow S_i$, mit $i \in \{1, \ldots, n\}$, der Operator für die **Projektion**: Es wird das i-te Element ausgewählt: $\pi_i((s_1, \ldots, s_n)) = s_i$. Für eine partielle Funktion $f : D \rightarrow\!\!\!\!\circ\; R$, wird $dom(f)$ für ihren **Definitionsbereich** $dom(f) =_{def} \{x \mid \exists y : y = f(x)\}$ und $ran(f)$ für ihren **Wertebereich** $ran(f) =_{def} \{y \mid \exists x : y = f(x)\}$ geschrieben. Für eine partielle Funktion $f : D \rightarrow\!\!\!\!\circ\; R$ und eine Menge D' ist die **Einschränkung** von f auf D' die Funktion $f|_{D'} : D' \rightarrow\!\!\!\!\circ\; R$ mit $\forall x \in D' \cap \mathrm{dom}(f) : f|_{D'}(x) = f(x)$ und undefiniert für alle anderen x. Für eine Menge von Funktionen F wird $F|_{D'}$ für das Resultat der elementweisen Anwendung von $\cdot|_{D'}$ geschrieben. Für eine partielle Funktion $f : D \rightarrow\!\!\!\!\circ\; R$ wird die totale Funktion $f^{-1} : R \rightarrow 2^D$ definiert mit $\forall y \in R : f^{-1}(y) =_{def} \{x \mid y = f(x)\}$.

Für eine totale Funktion $f : D \rightarrow R$ und eine partielle Funktion $g : D \rightarrow\!\!\!\!\circ\; R$ wird der **Überschreibe-Operator** $\lhd : ((D \rightarrow R) \times (D \rightarrow\!\!\!\!\circ\; R)) \rightarrow (D \rightarrow R)$ wie folgt definiert:

$$(f \lhd g)(x) =_{def} \begin{cases} g(x) \; falls \; x \in dom(g) \\ f(x) \; sonst \end{cases}$$

Nun wird die **Expansion** einer Uhrenbelegung auf eine Obermenge von Uhren definiert, wobei die Werte der hinzugekommenen Uhren nicht eingeschränkt werden. Sei $V \subseteq Val(X)$ eine Menge von Uhrenbelegungen von X, und sei X' eine Obermenge von X. Dann ist die Expansion von V zu X', geschrieben als $extend(V, X')$, wie folgt definiert:

$$extend(V, X') \;\; =_{def} \;\; \{v \in Val(X') \;\; | \;\; v|_X \in V\}$$

[1]Die hier verwendete Interpretation von Prädikaten entspricht nicht ganz der traditionellen Definition aus der Mathematik. In der formalen Logik ergibt die Interpretation eines Prädikats für eine bestimmte Belegung einen Wert aus $\{true, false\}$ (vgl. [Bac92] und [LSS87]).

2.2 Timed Automata

In diesem Abschnitt wird der Basisformalismus Timed Automata vorgestellt. Dazu wird zunächst eine Intuition für die Timed Automata vermittelt, indem ein kleines Beispiel erklärt wird. Danach erfolgt eine formale Definition der Timed Automata in Anlehnung an Alur [Alu99], da diese Definition weithin akzeptiert ist und somit eine gute Grundlage bildet. Um die Bedeutung der Modelle festzulegen, wird die kontinuierliche Standard-Semantik für Timed Automata aufgeführt.

2.2.1 Beispiel: nMOS-Transistor

Abbildung 2.2 auf der nächsten Seite zeigt einen Timed Automaton, der das Verhalten eines nMOS-Transistors modelliert. Der Automat hat vier Zustände (*Off*, *Rising*, *On*, *Falling*), eine Uhr (c) und zwei diskrete Variablen[2] ($gate$, out). Die Variable out repräsentiert den Leitfähigkeitszustand des Transistors. Zustand *Off* ist der Startzustand des Automaten, der Initialwert für out ist 0. Er modelliert die Situation, daß der Transistor nicht-leitend ist. Der Transistor kann in dieser Situation verharren, solange die Eingabe-Variable $gate$ den Wert 0 hat. Wenn das Gate auf 1 umgeschaltet wird, nimmt der Automat einen Zustandsübergang zum Zustand *Rising* und setzt die Uhr c zurück. In diesem Moment beginnt der Transistor mit dem Öffnen seines Kanals. Die Zustände *Rising* und *Falling* werden instabil genannt, weil der Ausgang des Transistors in diesen Zuständen seinen Wert verändert. Ist das Gate nach wenigstens zwei Zeiteinheiten noch auf 1 gesetzt, dann kann der Kanal des Transistors leitend werden und der Automat in den Zustand *On* übergehen, wobei die Variable out auf 1 gesetzt wird. Nach höchstens drei Zeiteinheiten muß der Automat den Zustand *Rising* verlassen. Falls das Gate inzwischen auf 0 gesetzt wird, muß der Automat sofort zum Zustand *Off* übergehen. Der Automat hat ein entsprechendes Verhalten für den Übergang von leitend ($out = 1$) zu nicht-leitend ($out = 0$).

2.2.2 Definition

Mittels Uhrenbedingungen können auf der syntaktischen Ebene die Mengen der erlaubten Uhrenbelegungen für Invarianten und Wächter eines Timed Automaton festgelegt werden. Die hier vorgestellte Definition der Timed Automata entspricht inhaltlich der von Alur [Alu99]. Die Einschränkung auf Konstanten aus der Menge der natürlichen Zahlen (inklusive 0) stellt keine echte Einschränkung gegenüber der Definition, die rationale Konstanten erlaubt, dar. Existieren in einem Modell rationale Konstanten, so kann durch Multiplikation aller Konstanten mit dem kleinsten gemeinsamen Vielfachen aller Nen-

[2] Für eine konzise Darstellung werden oft **diskrete Variablen** benutzt, die als Wertebereich eine endliche Teilmenge der natürlichen Zahlen haben und ihren Wert ausschließlich beim Schalten von Zustandsübergängen ändern. Diskrete Variablen werden im Formalismus nicht explizit betrachtet, da sie nur eine abkürzende Schreibweise für Zustände sind: Statt einer diskreten Variable kann ein Automat benutzt werden, der für jeden Wert der diskreten Variable einen Zustand hat, und der für jedes Lesen oder Setzen der diskreten Variable eine Synchronisationsmarke und einen entsprechenden Zustandsübergang hat.

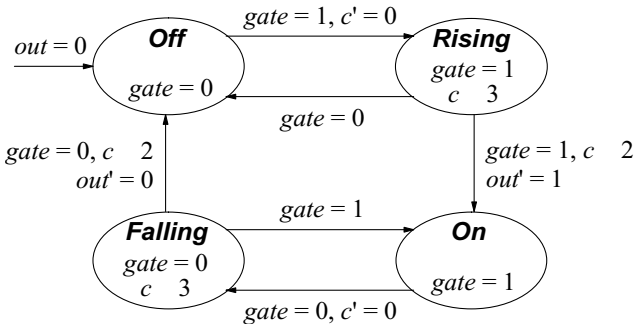


Abbildung 2.2: Timed Automaton für einen nMOS-Transistor.

ner der rationalen Konstanten ein äquivalentes Modell erzeugt werden, das ausschließlich natürliche Konstanten enthält.

Definition 2.4 *Ein* **Timed Automaton** *ist ein Tupel* $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ *mit:*

- L *ist eine endliche Menge von* **Zuständen**.
- L^0 *ist eine Menge von* **Initialzuständen**.
- X *ist eine endliche Menge von* **Uhren**.
- Σ*, mit* $\Sigma \cap \mathbb{R}_+ = \emptyset$*, ist eine endliche Menge von* **Synchronisationsmarken** *(auch Alphabet genannt).*
- $I : L \to \Phi(X)$ *ist eine totale Funktion, die jedem Zustand in* L *eine* **Invariante** *aus* $\Phi(X)$ *zuordnet.*
- $E \subseteq L \times \Sigma \times \Phi(X) \times 2^X \times L$ *ist eine Menge von* **Zustandsübergängen**.

Anmerkung: Ein Zustandsübergang $(l, \sigma, \varphi, Y, l') \in \mathcal{A}.E$ repräsentiert einen mit der Synchronisationsmarke σ markierten Übergang von Startzustand l zu Zielzustand l'. Der Wächter φ muß erfüllt sein, damit der Übergang möglich ist. Der Übergang setzt alle Uhren aus Y auf den Wert 0 zurück.

Die Punkt-Schreibweise $A.x$ wird benutzt, um die Komponente x von A zu bezeichnen, z. B. ist $\mathcal{A}.E$ die Menge der Zustandsübergänge des Automaten $\mathcal{A}$. In den Abbildungen von Timed Automata werden Synchronisationsmarken, die kein anderer Automat im Alphabet hat, meist weggelassen.

Zusätzlich werden für einen Timed Automaton abkürzende Schreibweisen eingeführt.

Definition 2.5 *Sei* $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ *ein Timed Automaton und* $e = (l, \sigma, \varphi, Y, l') \in E$ *ein Zustandsübergang. Dann gilt:*

- $\mathcal{A}.guard : E \to 2^{Val(X)}$ *ist die totale Funktion, die jedem Zustandsübergang die Menge von Uhrenbelegungen von* X *für seinen* **Wächter** *zuordnet:*
  $\mathcal{A}.guard(e) = [\![\varphi]\!]$.

- $\mathcal{A}.sync : E \to \Sigma$ *ist die totale Funktion, die jedem Zustandsübergang seine* **Synchronisationsmarke** *zuordnet:*
  $\mathcal{A}.sync(e) = \sigma$.

- $\mathcal{A}.reset : E \to 2^X$ *ist die totale Funktion, die jedem Übergang die Menge der* **rückzusetzenden Uhren** *zuordnet:*
  $\mathcal{A}.reset(e) = Y$.

Anmerkung: In der Literatur gibt es verschiedene Definitionen von Timed Automata. Einige erlauben Wächter der Form $x \sim y + c$ mit $x, y \in X$. Nach der Definition von Alur sind solche Uhrenbedingungen jedoch nicht erlaubt [Alu99]. Diese Definition vermeidet auch Probleme mit der ganzzahligen Diskretisierung (siehe Abschnitt 2.4.3).

2.2.3 Parallele Komposition

Komplexe Systeme können als **parallele Komposition** mehrerer Timed Automata beschrieben werden, die mittels Synchronisationsmarken miteinander kommunizieren. Eine Komposition zweier Timed Automata mit disjunkten Uhrenmengen kann in einen einzelnen Automaten transformiert werden, indem der Produktautomat konstruiert wird. Die Zustände des Produktautomaten sind Paare der Komponentenzustände, die Invariante eines zusammengesetzten Zustands ist die Konjunktion der Invarianten der Komponentenzustände. Zwei Zustandsübergänge der Komponenten mit der gleichen Synchronisationsmarke sind synchronisiert.

Definition 2.6 *Seien zwei Timed Automata* $\mathcal{A}_1 = (L_1, L_1^0, X_1, \Sigma_1, I_1, E_1)$ *und* $\mathcal{A}_2 = (L_2, L_2^0, X_2, \Sigma_2, I_2, E_2)$ *gegeben, und sei (ohne Beschränkung der Allgemeinheit)* $X_1 \cap X_2 = \emptyset$. *Dann ist der* **Produktautomat** $\mathcal{A}_1 \| \mathcal{A}_2$ *durch den Timed Automaton* $(L_1 \times L_2, L_1^0 \times L_2^0, X_1 \cup X_2, \Sigma_1 \cup \Sigma_2, I, E)$ *gegeben, wobei* $I((l_1, l_2)) = I_1(l_1) \wedge I_2(l_2)$ *und* $((l_1, l_2), \sigma, \varphi, Y, (l_1', l_2')) \in E$ *gdw.*

- $\sigma \in \Sigma_1 \cap \Sigma_2$ *und es existieren Zustandsübergänge* $(l_1, \sigma, \varphi_1, Y_1, l_1') \in E_1$ *und* $(l_2, \sigma, \varphi_2, Y_2, l_2') \in E_2$*, so daß* $\varphi = \varphi_1 \wedge \varphi_2$ *und* $Y = Y_1 \cup Y_2$ *gilt, oder*

- $\sigma \in \Sigma_1 \backslash \Sigma_2$*,* $l_2 = l_2'$ *und es existiert ein Zustandsübergang* $(l_1, \sigma, \varphi, Y, l_1') \in E_1$*, oder*

- $\sigma \in \Sigma_2 \setminus \Sigma_1$*,* $l_1 = l_1'$ *und es existiert ein Zustandsübergang* $(l_2, \sigma, \varphi, Y, l_2') \in E_2$.

Anmerkung: Die Bedingung $X_1 \cap X_2 = \emptyset$ ist keine echte Einschränkung, da bei nicht leerem Schnitt ein äquivalentes Automatenmodell konstruiert werden kann, bei dem ein zusätzlicher Automat die gemeinsamen Uhren kapselt und die Kommunikation über Synchronisationsmarken erfolgt.

Sei A eine nicht leere, endliche Menge von Timed Automata. Dann bezeichnet $\prod_{\mathcal{A} \in A} \mathcal{A}$ die **parallele Komposition aller Elemente von** A in einer bestimmten Ordnung.

Anmerkung: Verschiedene Ordnungen der Automaten aus A führen zwar zu unterschiedlichen resultierenden Automaten, jedoch sind diese isomorph.

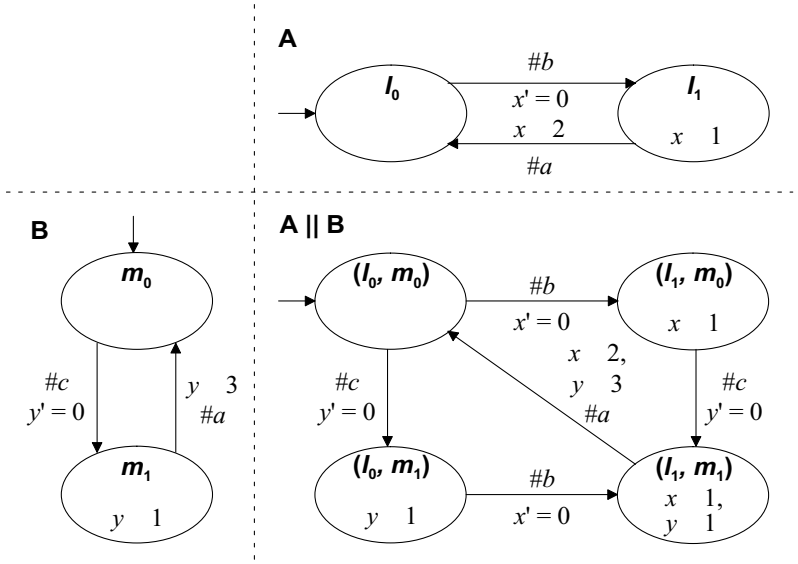


Abbildung 2.3: Zwei Timed Automata und ihr Produktautomat.

In Abbildung 2.3 wird ein Beispiel für die Konstruktion eines Produktautomaten gezeigt. Es werden drei Synchronisationsmarken (a, b, c) und zwei Uhren (x, y) verwendet ('#' ist ein Präfix für Synchronisationsmarken).

2.3 Modularität – Erster Schritt zur Bewältigung großer Systeme

In dem neuen CTA-Formalismus sind folgende Konzepte integriert worden, die durch Ausnutzung der Struktur zu einer übersichtlicheren Beschreibung und zu leichterem Verständnis des Modells beizutragen:

- Hierarchie: Im CTA-Modell können einzelne Komponenten zu einem Subsystem zusammengefaßt und als Einheit betrachtet werden. Systeme können somit durch eine hierarchische Enthaltenseinsstruktur modelliert werden. Schnittstellensignale und -variablen werden von lokalen Signalen und Variablen getrennt gehandhabt und mit Gültigkeitsbereichen versehen. In den Werkzeugen Kronos, Uppaal und HyTech ist dies nicht möglich.

- Kapselung und explizite Schnittstellen: Durch die Modellierung von Subsystemen als Module kann das innere Verhalten verkapselt werden. Dadurch, daß das innere Verhalten eines Moduls komplexer ist als das von außen sichtbare, kann mit der Modularisierung ein großes Abstraktionspotential genutzt und damit die Komposition ideal unterstützt werden (hohe Kohäsion innerhalb des Moduls, niedrige Kopplung nach außen).

- Wiederverwendbare Komponenten: Im CTA-Formalismus können durch das Konzept der generischen Modellierung wiederverwendbare Subsysteme durch Schnittstellenparameter flexibel gestaltet und mittels Schablonenmechanismus beliebig oft von ein und demselben Schablonenmodul instanziiert werden.

- Explizite Zugriffstypen für Synchronisationsmarken: Die Notation gestattet es, explizit auszudrücken, daß ein Signal für einen Automaten ein Eingabe-, Ausgabe- oder mehrfach einschränkbares Signal ist. Diese Unterscheidung ermöglicht ein komfortableres Modellieren und ein besseres Verstehen eines Modells, da in der Syntax mehr Information enthalten ist (”syntaktischer Zucker”).

- Explizite Zugriffstypen für Variablen: Analog erlaubt die Notation explizite Zugriffstypen für Variablen, welche als Eingabe-, Ausgabe- und mehrfach einschränkbare Variablen deklariert werden können. In den bestehenden Werkzeugen für Timed Automata sind alle Signale und Variablen global gültig mit uneingeschränktem Zugriff.

- Automatische Vervollständigung von Automaten mit Eingabe-Signalen: Eingabe-Signale sind Ereignisse, auf die der Automat jederzeit reagieren können muß. Falls er nicht auf ein Eingabe-Signal reagieren kann, so wird dies als Eintritt in einen speziellen Fehlerzustand verstanden. Durch diese Vorgehensweise kann der Modellierer per Analyse überprüfen, ob solch ein Fehlerzustand erreichbar ist und somit eine Fehl-Synchronisation auftritt.

Diese Modellierungskonzepte wurden in einer nicht-formalen Fassung bereits vor einigen Jahren veröffentlicht [BR98]; die formalen Definitionen und die Semantiken werden im folgenden in Anlehnung an [BR99b] bzw. an die überarbeitete Version [BR01] dargestellt.

Das Konzept, für eine bessere Strukturierung einer Beschreibung Module einzuführen, ist ein altes und generelles Grundprinzip in der Softwaretechnik [PCW84]. Ziel ist dabei immer, die Struktur der Systeme so explizit wie möglich auszudrücken.

Die Differenzierung zwischen unterschiedlichen Rollen der Signale in Automaten wurde auch in der Definition der IO-Automaten von Lynch und Tuttle benutzt [LT87] und für hybride Systeme erweitert [LSVW96]. Ein anderer Ansatz zur Beschreibung von hybriden Systemen wurde von Alur und Henzinger vorgestellt [AH97]; er basiert auf Reactive Modules [AH96], wobei eine Erweiterung um kontinuierlich veränderliche Variablen vorgenommen wurde. Diese Ansätze sind jedoch nur teilweise im Werkzeug Mocha implementiert worden [AdAG+01].

Die Ansätze von Mocha, Charon und Masaccio beschäftigen sich mit der Lösung der oben genannten Probleme, basieren jedoch auf Modellierungsformalismen, die sich von Timed Automata unterscheiden. Mocha basiert auf *Reactive Modules* (und teilweise auf *Timed Modules*). Es wird auch eine Verfeinerungsanalyse angeboten, jedoch nur für Reactive Modules (ohne Zeitaspekte) [AHM+98]. Charon ist eine Modellierungssprache, die auf hierarchischen hybriden Modulen basiert [AGH+00]. Derzeit ist kein Werkzeug für Model-Checking von Charon-Modellen verfügbar. Masaccio ist ein Formalismus (derzeit ebenfalls ohne Werkzeugimplementierung) für dynamische hybride Systeme, der zur

Modellierung der kontinuierlichen Komponenten eines Systems Differentialgleichungen verwendet [Hen00]. Auch der Formalismus Statecharts unterstützt hierarchische Modelle [Har87]. Durch die Aufnahme in den UML-Standard ist die Notation populär geworden und es existiert eine kommerzielle Werkzeugunterstützung [HLN+90]. Mit dieser Notation lassen sich jedoch kontinuierliche Systeme nur begrenzt behandeln und die synchrone Zeittakt-Steuerung erfordert eine komplizierte Semantik.

Die Vorteile der hierarchischen Verifikation von modular gegliederten Modellen – ohne vorheriges Auflösen der hierarchischen Struktur – scheinen wegen der impliziten Synchronisation durch das Vergehen der Zeit bei Timed Automata nicht zu Effizienzverbesserung zu führen (siehe auch [ABB+01]). Deshalb wird die Semantik des gründlich erforschten Basisformalismus der Timed Automata genutzt und die modulare Struktur auf eine "flache" Menge von Timed Automata abgebildet. Für die Effizienzverbesserung der Verifikationsalgorithmen wird die modulare Struktur hier in anderer Weise genutzt (siehe Abschnitt 3.3 über das statische Variablenordnen). Der Prozeß des Auflösens der hierarchischen Struktur wird *Flattening* genannt und entspricht dem allgemeinen Konzept der Auflösung von Hierarchie, wie es auch in anderen Bereichen der Softwaretechnik Verwendung findet [Mey88, BLS01, BLS00].

2.3.1 Module

CTA-Module bestehen aus einem Automaten und einer Partition der Variablen und Signale in disjunkte Mengen für verschiedene Zugriffstypen: nur lesen, exklusives einschränken, mehrfach einschränkbar und lokal. Damit beinhaltet das Konzept der CTA-Module diejenigen der neuen Konzepte, die mit der Schnittstellenspezifikation eines CTA-Moduls zu tun haben.

Definition 2.7 *Ein* **CTA-Modul** $\mathcal{M} = (\mathcal{A}, \Sigma, X, par_\Sigma, par_X)$ *ist ein Tupel mit:*

- $\mathcal{A}$ *ist ein Timed Automaton.*
- Σ *ist die Menge der Signale*[3].
- X *ist die Menge der Variablen*[4].
- $par_\Sigma : \Sigma \to \{\mathtt{I}, \mathtt{O}, \mathtt{M}, \mathtt{L}\}$ *ist eine Partitionierungsfunktion für Signale.* $par_\Sigma^{-1}(\mathtt{I})$ *ist die Menge der Eingabe-Signale,* $par_\Sigma^{-1}(\mathtt{O})$ *die Menge der Ausgabe-Signale,* $par_\Sigma^{-1}(\mathtt{M})$ *die Menge der mehrfach einschränkbaren Signale und* $par_\Sigma^{-1}(\mathtt{L})$ *die Menge der lokal definierten Signale.*
- $par_X : X \to \{\mathtt{I}, \mathtt{O}, \mathtt{M}, \mathtt{L}\}$ *ist eine Partitionierungsfunktion für Variablen.* $par_X^{-1}(\mathtt{I})$ *ist die Menge der Eingabe-Variablen,* $par_X^{-1}(\mathtt{O})$ *die Menge der Ausgabe-Variablen,*

[3] Im Zusammenhang mit der modularen Struktur und der Unterscheidung zwischen Eingabe-, Ausgabe-, mehrfach einschränkbaren und lokalen Synchronisationsmarken wird auch der Begriff **Signal** verwendet, da dieser Begriff eher die gewünschte Intuition des Konzepts hervorruft.

[4] Die einzige bisher eingeführte Art von Variablen sind Uhren. Später werden weitere Arten von **Variablen** eingeführt.

$par_X^{-1}(\mathtt{M})$ die Menge der mehrfach einschränkbaren Variablen und $par_X^{-1}(\mathtt{L})$ die Menge der lokal definierten Variablen.

Diese Komponenten haben die folgenden Axiome zu erfüllen:

1. $\mathcal{A}.\Sigma = \Sigma$ *und* par_Σ *ist total.*

2. $\sigma \in \Sigma \wedge par_\Sigma(\sigma) = \mathtt{I}$
   $\Rightarrow \quad \forall l \in \mathcal{A}.L : \left(\bigcup_{e \in \{d \in \mathcal{A}.E \mid \mathcal{A}.sync(d) = \sigma \wedge \pi_1(d) = l\}} \mathcal{A}.guard(e) \right) = Val(X)$

3. $\mathcal{A}.X = X$ *und* par_X *ist total.*

4. $x \in X \wedge par_X(x) = \mathtt{I}$
   $\Rightarrow \quad \forall e \in \mathcal{A}.E : \quad x \notin \mathcal{A}.reset(e)$

Anmerkung: Die Axiome in Definition 2.7 legen die Unterscheidung der Eingabe-Signale und -Variablen von den anderen Signalen und Variablen fest. Zu beachten ist, daß Ausgabe-, mehrfach einschränkbare und lokale Signale und Variablen durch diese Definitionen nicht voneinander unterschieden werden. Die Unterscheidung dieser Konzepte wird erst bei der Betrachtung von CTA-Kompositionen relevant.

Die Definition für die Eingabe-*Signale* (Axiom 2) kann wie folgt interpretiert werden: Für jedes Eingabe-Signal ist in jedem Zustand des Automaten jederzeit ein mit dem Eingabe-Signal markierter Zustandsübergang erlaubt. In dieser Weise kann der Automat das Eingabe-Signal nicht einschränken und somit nicht für einen Deadlock verantwortlich gemacht werden.

Die mehrfach einschränkbaren Signale und Variablen sind für jeden Zugriffsmodus verfügbar. Sowohl das Modul als auch seine Umgebung kann ein als mehrfach einschränkbar deklariertes Signal oder eine als mehrfach einschränkbar deklarierte Variable in beliebiger Weise einschränken.

Für Eingabe-*Variablen* (Axiom 4) kann die Definition wie folgt interpretiert werden: Der Wert einer Eingabe-Variable darf in keiner der Zustandsübergänge durch ein Rücksetzen verändert werden.

2.3.2 Beispiel: AND-Schaltkreis

Die Strukturierungsmöglichkeiten werden an einem Beispielsystem verdeutlicht: eine auf MOS[5]-Transistoren basierende integrierte Schaltung für die AND-Funktion mit 4 Eingängen. Ein Modell für diesen Schaltkreis, das aus einer Menge einzelner, unstrukturierter Automaten besteht, wurde für das Werkzeug Kronos benutzt [BMPY97]. Im folgenden wird ein modular strukturiertes CTA-Modell für diesen Schaltkreis vorgestellt. Im Abschnitt 5.4.2 wird dieses Beispiel für einen Performance-Vergleich mit Kronos benutzt.

[5]'MOS' steht für Metall-Oxid-Silizium, der Bauart in der Halbleitertechnik für (spannungsgesteuerte) Feldeffekt-Transistoren.

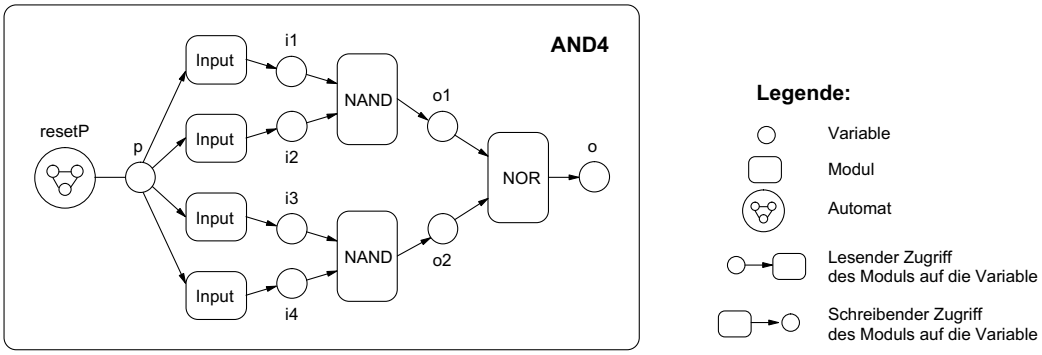


Abbildung 2.4: Modell der AND-Schaltung.

Abbildung 2.4 illustriert, in welcher Weise das Modell aus verschiedenen Modulen und Automaten zusammengesetzt ist. In der Abbildung werden die Kommunikationsverbindungen zwischen verschiedenen Komponenten in einer Datenfluß-ähnlichen Diagrammart dargestellt. Das Hauptmodul, welches das Verhalten des logischen AND-Gatters mit 4 Eingängen modelliert, besteht aus zwei Modul-Instanzen eines NAND-Gatters und einer Modul-Instanz eines NOR-Gatters. Die Umgebung des AND-Gatters wurde durch die Uhr p für den Zeit-Takt und 4 Variablen für die Eingangsbelegungen modelliert. Die Uhr erhält den Startwert 0. Wenn der Wert 15 erreicht ist, setzt der Automat resetP diese Uhr zurück auf 0. Ein Modul Input besteht aus einem Automaten, der während der ersten fünf Zeittakte von p einmal den Wert einer Eingabe-Variablen (i1, i2, i3, i4) ändern kann. Während der letzten 10 Zeit-Takte bleiben die Eingabe-Variablen unverändert (sie sind 'stabil'). Die binären Variablen o1 und o2 repräsentieren die Ausgabewerte der zwei NAND-Gatter und o modelliert den Ausgabewert des NOR-Gatters und somit den Ausgabewert der gesamten Schaltung.

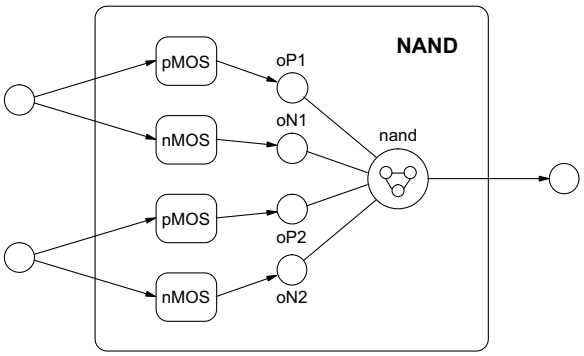


Abbildung 2.5: Modell des logischen NAND.

Abbildung 2.5 zeigt die Struktur des Moduls für ein NAND-Gatter (NAND). Es besteht aus zwei pMOS-Transistoren und zwei nMOS-Transistoren. Der Automat (nand) liest die Ausgabe-Variablen out (leitend bzw. nicht-leitend) der Transistoren (mit diesem

Modul verbunden als oP1, oN1, oP2 und oN2) und entscheidet damit den Ausgabewert des NAND-Gatters. Das Modul für das NOR-Gatter verhält sich analog.

Ein Modul für einen nMOS-Transistor enthält eine Uhr c und einen Automaten. Ein solcher nMOS-Transistor (siehe Abbildung 2.2 auf Seite 27) braucht zwischen 2 und 3 Zeiteinheiten, um seinen Ausgabewert nach einer Änderung des Potentials am Gate neu zu belegen. Der Unterschied zum pMOS-Transistors besteht im inversen Ausgabewert und darin, daß der pMOS-Transistor spätestens 4 Zeiteinheiten nach der Änderung am Gate mit einer Veränderung seines Ausgabewertes reagiert.

Interessante, zu analysierende Fragen zum Verhalten der AND-Schaltung sind: Wieviele Transistoren können ihren Zustand gleichzeitig verändern? (Die maximale Stromaufnahme des Schaltkreises verhält sich proportional zur Anzahl der gleichzeitigen Umschaltvorgänge der Transistoren.) Ist in der AND-Schaltung ein Kurzschluß möglich? Zur Beantwortung dieser Fragen müssen die erreichbaren Konfigurationen des Modells berechnet und überprüft werden. Im Abschnitt 5.4.2 wird über die Performanceresultate für die Berechnung aller erreichbaren Konfigurationen berichtet, weil diese der Engpaß bei der Verifikation der Schaltung ist.

2.3.3 Komposition

Um aus einer Menge von CTA-Modulen eine Komposition erzeugen zu können, sind einige Einschränkungen notwendig, die dafür sorgen, daß die Komposition wiederum ein CTA-Modul sein kann. Die CTA-Komposition wird in der folgenden Weise als parallele Komposition von CTA-Modulen definiert.

Definition 2.8 *Eine* **CTA-Komposition** $\mathcal{C} = (M, \Sigma, X, par_\Sigma, par_X)$ *ist ein Tupel mit:*

- M *ist eine endliche, nicht leere Menge von CTA-Modulen.*
- Σ *ist eine endliche Menge von Signalen.*
- X *ist eine endliche Menge von Variablen.*
- $par_\Sigma : \Sigma \to \{\mathtt{I}, \mathtt{O}, \mathtt{M}, \mathtt{L}\}$ *ist eine Partitionierungsfunktion für Signale.*
- $par_X : X \to \{\mathtt{I}, \mathtt{O}, \mathtt{M}, \mathtt{L}\}$ *ist eine Partitionierungsfunktion für Variablen.*

Die Komponenten haben die folgenden Axiome zu erfüllen:

1. *Explizite Zugriffstypen für Signale:*

$$par_\Sigma \text{ ist total.}$$

2. *Kommunikation über gemeinsame Signale:*

$$\forall \mathcal{M}, \mathcal{M}' \in M : \mathcal{M} \neq \mathcal{M}' \Rightarrow \mathcal{M}.\Sigma \cap \mathcal{M}'.\Sigma \subseteq \Sigma$$

3. *Unsichtbarkeit der lokalen Signale der Module:*

$$\forall \mathcal{M} \in M : \mathcal{M}.par_{\Sigma}^{-1}(\mathtt{L}) \cap \Sigma = \emptyset$$

4. *Benutzung von Eingabe-Signalen der Komposition durch Module:*

$$\forall \mathcal{M} \in M : par_{\Sigma}^{-1}(\mathtt{I}) \cap \mathcal{M}.\Sigma \subseteq \mathcal{M}.par_{\Sigma}^{-1}(\mathtt{I})$$

5. *Benutzung von Ausgabe-Signalen der Module:*

$$\forall \mathcal{M}, \mathcal{M}' \in M : \mathcal{M} \neq \mathcal{M}' \Rightarrow \left(\begin{array}{ll} & \mathcal{M}.par_{\Sigma}^{-1}(\mathtt{O}) \cap \mathcal{M}'.\Sigma \subseteq \mathcal{M}'.par_{\Sigma}^{-1}(\mathtt{I}) \\ \wedge & \mathcal{M}.par_{\Sigma}^{-1}(\mathtt{O}) \cap (par_{\Sigma}^{-1}(\mathtt{I}) \cup par_{\Sigma}^{-1}(\mathtt{M})) = \emptyset \end{array} \right)$$

Analoge Axiome gelten für X und par_X.

Anmerkung: Im folgenden werden die Axiome verbal beschrieben:

1. Die Mengen $par_{\Sigma}^{-1}(\mathtt{I})$, $par_{\Sigma}^{-1}(\mathtt{O})$, $par_{\Sigma}^{-1}(\mathtt{M})$ und $par_{\Sigma}^{-1}(\mathtt{L})$ sind eine Partition der Menge Σ.
   Die Mengen $par_{X}^{-1}(\mathtt{I})$, $par_{X}^{-1}(\mathtt{O})$, $par_{X}^{-1}(\mathtt{M})$ und $par_{X}^{-1}(\mathtt{L})$ sind eine Partition der Menge X.

2. Elemente von M dürfen nur über Elemente von Σ und X kommunizieren.

3. Die lokalen Signale und Variablen verschiedener Module sind verschieden. Die lokalen Signale und Variablen eines CTA-Moduls können außerhalb des CTA-Moduls nicht benutzt werden.

4. Eingabe-Signale und -Variablen der CTA-Komposition werden in den Modulen aus M nur als Eingabe-Signale bzw. -Variablen benutzt.

5. Ausgabe-Signale eines CTA-Moduls dürfen in der Umgebung des CTA-Moduls nur als Eingabe-Signale verwendet werden. Diese Ausgabe-Signale müssen in der enthaltenden CTA-Komposition entweder als lokal oder als Ausgabe-Signal deklariert sein. Analoge Aussagen gelten für die Ausgabe-Variablen.

Diese Definition für die CTA-Kompositionen enthält einige der wichtigsten Konzepte der vorliegenden Arbeit. Sie erlauben es, ein System in Subsysteme zu unterteilen und so die modulare Struktur eines Systems explizit zu machen, und sie enthalten den Kern der Konzepte zur Differenzierung unterschiedlicher Zugriffsarten von Signalen und Variablen.

Bisher fehlt noch die Möglichkeit zur Definition von hierarchischen Systemen. Dies erlaubt die Definition eines CTA-Moduls, das äquivalent zu einer gegebenen CTA-Komposition ist. Die Automaten der Module einer Komposition können semantisch zu einem Automaten kombiniert werden. Dies führt zu einem einzigen CTA-Modul, das der CTA-Komposition entspricht. Zunächst wird eine Konstruktionsfunktion definiert, die aus der CTA-Komposition ein Tupel definiert, von dem anschließend behauptet wird, daß es ein CTA-Modul ist.

Definition 2.9 *Die Funktion $ctamod$ bildet jede CTA-Komposition auf eine "flache", d. h. von Hierarchie freie, Struktur ab. Sei $\mathcal{C}$ eine CTA-Komposition. Dann ist $ctamod(\mathcal{C})$ das Tupel $(\mathcal{A}, \Sigma, X, par_\Sigma, par_X)$ mit:*

- $\mathcal{A} =_{def} \prod_{\mathcal{M} \in \mathcal{C}.M} \mathcal{M}.\mathcal{A}$
- $\Sigma =_{def} \bigcup_{\mathcal{M} \in \mathcal{C}.M} \mathcal{M}.\Sigma$
- $par_\Sigma : \Sigma \rightarrow \{\mathtt{I}, \mathtt{O}, \mathtt{M}, \mathtt{L}\}$ *ist in der folgenden Weise definiert:*
  - $par_\Sigma(\sigma) = \mathtt{I} \Leftrightarrow \mathcal{C}.par_\Sigma(\sigma) = \mathtt{I}$
  - $par_\Sigma(\sigma) = \mathtt{O} \Leftrightarrow \mathcal{C}.par_\Sigma(\sigma) = \mathtt{O}$
  - $par_\Sigma(\sigma) = \mathtt{M} \Leftrightarrow \mathcal{C}.par_\Sigma(\sigma) = \mathtt{M}$
  - $par_\Sigma(\sigma) = \mathtt{L} \Leftrightarrow \mathcal{C}.par_\Sigma(\sigma) = \mathtt{L} \vee \sigma \notin dom(\mathcal{C}.par_\Sigma)$
- X *und* par_X *werden analog zu* Σ *und* par_Σ *konstruiert.*

Anmerkung: Die Menge der lokalen Signale und Variablen des der gegebenen CTA-Komposition entsprechenden Tupels besteht aus den lokalen Signalen und Variablen der CTA-Komposition selbst und aus allen Signalen und Variablen der enthaltenen Module, die nicht in der Schnittstellendefinition der CTA-Komposition auftreten (z. B. lokale Signale und Variablen der enthaltenen Module).

Um den Produktautomaten für die gesamte Komposition zu erhalten, muß das Produkt aller Automaten aus den enthaltenen Modulen gebildet werden.

Das durch die Funktion $ctamod$ zu einer CTA-Komposition definierte Tupel ist ein CTA-Modul:

Satz 2.1 *Sei $\mathcal{C}$ eine CTA-Komposition. Dann ist $ctamod(\mathcal{C}) = (\mathcal{A}, \Sigma, X, par_\Sigma, par_X)$ ein CTA-Modul.*

Beweis: Zu zeigen ist, daß das durch die Funktion $ctamod(\mathcal{C})$ erzeugte Tupel die vier Axiome der Definition 2.7 erfüllt und somit ein CTA-Modul ist. Im folgenden werden die einzelnen Axiome für CTA-Module nacheinander bewiesen:

1. Die von der Definition 2.7 (Axiom 1) eines CTA-Moduls geforderte Bedingung $\Sigma = \mathcal{A}.\Sigma$ ist erfüllt, da für jedes Modul $\mathcal{M} \in \mathcal{C}.M$ nach der Modul-Definition $\mathcal{M}.\Sigma = \mathcal{M}.\mathcal{A}.\Sigma$ gilt, nach der Definition der Funktion $ctamod(\mathcal{C})$ (Punkt 2) alle Signale in Σ enthalten sind und der Produktautomat zweier Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ alle Signale aus $\mathcal{A}_1.\Sigma \cup \mathcal{A}_2.\Sigma$ im Alphabet hat.

   par_Σ ist total, da nach Axiom 1 der Definition 2.8 für CTA-Kompositionen $\mathcal{C}.par_\Sigma$ total ist und bei der Konstruktion alle weiteren Signale dem Wert $\mathtt{L}$ zugeordnet werden.

2. Das zweite Axiom fordert, daß für $\mathcal{A}$ gilt:

$$\forall \sigma \in par_{\Sigma}^{-1}(\mathtt{I}): \quad \forall l \in \mathcal{A}.L:$$

$$\left(\bigcup_{e \in \{d \in \mathcal{A}.E \mid \mathcal{A}.sync(d)=\sigma \wedge \pi_1(d)=l\}} \mathcal{A}.guard(e) \right) = Val(\mathcal{A}.X)$$

Nach Axiom 4 und 5 aus Definition 2.8 können Eingabe-Signale der Komposition innerhalb der Module nur als Eingabe-Signale benutzt werden.

Nach Axiom 2 von Definition 2.7 gilt für jedes Modul $\mathcal{M} \in \mathcal{C}.M$:

$$\forall \sigma \in \mathcal{M}.par_{\Sigma}^{-1}(\mathtt{I}): \quad \forall l \in \mathcal{M}.\mathcal{A}.L:$$

$$\left(\bigcup_{e \in \{d \in \mathcal{M}.\mathcal{A}.E \mid \mathcal{M}.\mathcal{A}.sync(d)=\sigma \wedge \pi_1(d)=l\}} \mathcal{M}.\mathcal{A}.guard(e) \right) = Val(\mathcal{M}.X)$$

Durch die Funktion $ctamod$ wird der Produktautomat der einzelnen Automaten der Module von $\mathcal{C}$ konstruiert. Sei $\mathcal{C}.M = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ die Menge der Module in der Komposition $\mathcal{C}$. Die Konstruktion des Produktes einer Menge von Automaten erfolgt durch die wiederholte Produktbildung zweier Automaten. Deshalb wird die Produktbildung zweier Automaten betrachtet und induktiv auf das Produkt mehrerer Automaten geschlossen. Sei $\mathcal{A}_k$ der bisher konstruierte Produktautomat. Entsprechend Definition 2.6 sind für den Produktautomaten $\mathcal{A}_{k+1} = \mathcal{A}_k \mid\mid \mathcal{M}_{k+1}.\mathcal{A}$ zwei Fälle zu betrachten. Sei $\sigma \in \mathcal{C}.par_{\Sigma}^{-1}(\mathtt{I})$ das betrachtete Eingabe-Signal, $(l_1, l_2) \in \mathcal{A}_{k+1}.L$ der betrachtete Zustand und o. B. d. A. $\sigma \in \mathcal{A}_k.\Sigma$.

Fall 1: $\sigma \notin \mathcal{M}_{k+1}.\mathcal{A}.\Sigma$. Nach Definition 2.6 existiert für jedes $(l_1, \sigma, \varphi_1, Y_1, l_1') \in \mathcal{A}_k.E$ ein Übergang $((l_1, l_2), \sigma, \varphi_1, Y_1, (l_1', l_2)) \in \mathcal{A}_{k+1}.E$. Damit bleibt die Eigenschaft erhalten und die Disjunktion der Wächter ergibt die volle Menge:
Sei $V = \{V_1, \ldots, V_n\}$ eine Menge von Belegungsmengen mit:
$V = \{extend([\![\varphi]\!], \mathcal{A}_{k+1}.X) \mid \exists((l_1, l_2), \sigma, \varphi, Y, (l_1', l_2')) \in \mathcal{A}_{k+1}.E\}$.
Dann gilt: $\bigcup_{i \in \{1,\ldots,n\}} V_i = Val(\mathcal{A}_{k+1}.X)$.

Fall 2: $\sigma \in \mathcal{M}_{k+1}.\mathcal{A}.\Sigma$. Dann existiert nach Definition 2.6 für jedes $(l_1, \sigma, \varphi_1, Y_1, l_1') \in \mathcal{A}_k.E$ und jedes $(l_2, \sigma, \varphi_2, Y_2, l_2') \in \mathcal{M}_{k+1}.\mathcal{A}.E$ ein Übergang $((l_1, l_2), \sigma, \varphi_1 \wedge \varphi_2, Y_1 \cup Y_2, (l_1', l_2')) \in \mathcal{A}_{k+1}.E$. Seien $V^1 = \{V_1^1, \ldots, V_{n_1}^1\}, V^2 = \{V_1^2, \ldots, V_{n_2}^2\}$ und $V = \{V_1, \ldots, V_n\}$ Mengen von Belegungsmengen mit:
$V^1 = \{extend([\![\varphi_1]\!], \mathcal{A}_{k+1}.X) \mid \exists(l_1, \sigma, \varphi_1, Y_1, l_1') \in \mathcal{A}_k.E\}$,
$V^2 = \{extend([\![\varphi_2]\!], \mathcal{A}_{k+1}.X) \mid \exists(l_2, \sigma, \varphi_2, Y_2, l_2') \in \mathcal{M}_{k+1}.\mathcal{A}.E\}$ und
$V = \{extend([\![\varphi\ ]\!], \mathcal{A}_{k+1}.X) \mid \exists((l_1, l_2), \sigma, \varphi, Y, (l_1', l_2')) \in \mathcal{A}_{k+1}.E\}$.

Es gelten

$\bigcup_{i\in\{1,\ldots,n_1\}} V_i^1 = Val(\mathcal{A}_{k+1}.X)$ (wegen Induktionsanfang) und

$\bigcup_{i\in\{1,\ldots,n_2\}} V_i^2 = Val(\mathcal{A}_{k+1}.X)$ (wegen Axiom 2 von Definition 2.7 und

da σ Eingabe-Signal von Modul $\mathcal{M}_{k+1}$ ist). Aus

$$\bigcup_{i\in\{1,\ldots,n\}} V_i = \bigcup_{i\in\{1,\ldots,n_1\},j\in\{1,\ldots,n_2\}} V_i^1 \cap V_j^2$$

$$= \bigcup_{i\in\{1,\ldots,n_1\}} V_i^1 \cap \bigcup_{i\in\{1,\ldots,n_2\}} V_i^2.$$

folgt die Behauptung $\bigcup_{i\in\{1,\ldots,n\}} V_i = Val(\mathcal{A}_{k+1}.X)$.

3. Die dritte Bedingung ergibt sich analog zur 1. Bedingung auch für Variablen.

4. Es wird gefordert, daß der Wert einer Eingabe-Variable nicht verändert werden darf, d. h. in den Mengen der rückzusetzenden Uhren darf keine Uhr aus $par_X^{-1}(\mathtt{I})$ vorkommen. Bei jedem Modul $\mathcal{M} \in \mathcal{C}.M$ dürfen dem vierten Axiom von CTA-Modul zufolge in $\mathcal{M}.\mathcal{A}.reset(e)$ keine Uhren aus $\mathcal{M}.par_X^{-1}(\mathtt{I})$ vorkommen. Da die Menge $\mathcal{C}.par_X^{-1}(\mathtt{I})$ nach Axiom 4 und 5 der Komposition nur Eingabe-Variablen von $\mathcal{M}$ enthält, ist diese Forderung auch für $ctamod(\mathcal{C})$ erfüllt.

Damit wurden alle Forderungen der Definition 2.7 erfüllt und es folgt die Behauptung von Satz 2.1. □

2.3.4 Instanziierung

Viele Systeme enthalten mehrere ähnliche Komponenten. Für die Modellierung von Systemen ist es daher oft hilfreich, Module von einer bereits modellierten Komponente, einem "Schablonenmodul", abzuleiten. Dafür wird das Konzept der Instanziierung angeboten, das es ermöglicht, die Komponenten einer CTA-Komposition von bereits existierenden, sogenannten Schablonenmodulen abzuleiten.

Definition 2.10 *Eine* **CTA-Instanziierung** *ist ein Tupel* $\mathcal{I} = (\mathcal{M}, \Sigma, X, ident_\Sigma, ident_X)$ *mit:*

- $\mathcal{M}$ *ist das instanziierte CTA-Modul.*
- Σ *ist die Menge der Signale der Instanziierung.*
- X *ist die Menge der Variablen der Instanziierung.*

- $ident_\Sigma : \mathcal{M}.\Sigma \rightarrow \Sigma$ *ist eine totale Identifikationsfunktion, die jedem Signal von* $\mathcal{M}$ *ein Signal aus* Σ *zuordnet.* $ident_\Sigma$ *muß verschiedenen Signalen des instanziierten CTA-Moduls* $\mathcal{M}$ *verschiedene Signale aus* Σ *zuordnen (*$ident$ *ist injektiv):*
  $\forall a, a' \in dom(ident) : a \neq a' \rightarrow ident(a) \neq ident(a')$
- $ident_X : \mathcal{M}.X \rightarrow X$ *ist analog zu* $ident_\Sigma$ *definiert.*

Sei $\mathcal{A}$ ein Automat und seien f_Σ und f_X zwei Identifikationsfunktionen mit $dom(f_\Sigma) = \mathcal{A}.\Sigma$ und $dom(f_X) = \mathcal{A}.X$. Die **Ersetzung von Signalen bzw. Variablen in** $\mathcal{A}$ **durch** f_Σ **bzw.** f_X wird als $replace(\mathcal{A}, f_\Sigma, f_X)$ bezeichnet. Das Ergebnis ist der Automat, der aus $\mathcal{A}$ entsteht durch das Ersetzen aller Signale bzw. Variablen des Definitionsbereiches von f_Σ bzw. f_X durch den jeweils von der Funktion f_Σ bzw. f_X zugeordneten Wert. Diese Ersetzung wird in allen Prädikaten für die Invarianten und Wächter des Automaten sowie für die Mengen der rückzusetzenden Uhren vorgenommen.

In der gleichen Vorgehensweise wie bei Komposition wird zunächst eine $ctamod$-Funktion definiert, die aus einer Instanziierung ein Tupel erzeugt, von dem später behauptet wird, daß es ein CTA-Module ist.

Definition 2.11 *Sei* $\mathcal{I}$ *eine CTA-Instanziierung. Dann ist* $ctamod(\mathcal{I})$ *das Tupel* $(\mathcal{A}, \Sigma, X, par_\Sigma, par_X)$ *mit:*

- $\mathcal{A} =_{def} replace(\mathcal{I}.\mathcal{M}.\mathcal{A}, \mathcal{I}.ident_\Sigma, \mathcal{I}.ident_X)$
- $\Sigma =_{def} ran(\mathcal{I}.ident_\Sigma)$
- *Für* $par_\Sigma : \Sigma \rightarrow \{\mathtt{I}, \mathtt{O}, \mathtt{M}, \mathtt{L}\}$ *gilt:*
  $\forall i \in \{\mathtt{I}, \mathtt{O}, \mathtt{M}, \mathtt{L}\} : par_\Sigma^{-1}(i) = ran(\mathcal{I}.ident_\Sigma|_{\mathcal{I}.\mathcal{M}.par^{-1}(i)})$
- X *und* par_X *werden analog zu* Σ *und* par_Σ *konstruiert.*

Anmerkung: Die Funktion $ctamod$ interpretiert eine CTA-Instanziierung als ein CTA-Modul. Mit Unterstützung dieser Interpretationsfunktion können kompatible Mengen von CTA-Instanziierungen als Komponenten einer CTA-Komposition benutzt werden.

Die Funktion $ctamod$ definiert ein CTA-Modul zu einer gegebenen CTA-Instanziierung:

Satz 2.2 *Sei* $\mathcal{I}$ *eine CTA-Instanziierung. Dann ist* $ctamod(\mathcal{I}) = (\mathcal{A}, \Sigma, X, par_\Sigma, par_X)$ *ein CTA-Modul.*

Beweis: Da $\mathcal{I}.ident_\Sigma$ und $\mathcal{I}.ident_X$ injektiv sind, sind par_Σ und par_X Funktionen und somit sind die Mengen $par_\Sigma^{-1}(\mathtt{I})$, $par_\Sigma^{-1}(\mathtt{O})$, $par_\Sigma^{-1}(\mathtt{M})$, $par_\Sigma^{-1}(\mathtt{L})$ $par_X^{-1}(\mathtt{I})$, $par_X^{-1}(\mathtt{O})$, $par_X^{-1}(\mathtt{M})$ und $par_X^{-1}(\mathtt{L})$ als disjunkte Mengen konstruiert worden. Die Umbenennung von Automaten-Komponenten verletzt keine der in Definition 2.7 aufgeführten Bedingungen. □

Der Name $ctamod$ benennt zwei verschiedene Funktionen, die beide CTA-Module erzeugen. Die erste Funktionen liefert ein CTA-Modul zu einer gegebenen Komposition, die

zweite liefert ein CTA-Modul zu einer Instanziierung. Durch diese Funktionen können – wo immer es bei der Modellierung sinnvoll und hilfreich erscheint – CTA-Instanziierungen oder CTA-Kompositionen anstatt explizit definierter CTA-Module benutzt werden.

Eine andere Anwendung der $ctamod$-Funktionen ist die Festlegung der Semantik. CTA-Kompositionen und CTA-Instanziierungen werden als CTA-Module interpretiert. Der Automat eines Moduls wird mittels einer Funktion $[\![.]\!]$ als markiertes Transitionssystem interpretiert; die **Semantik eines CTA-Moduls** $\mathcal{M}$ ist gegeben durch die Semantik des enthaltenden Automaten: $[\![\mathcal{M}]\!] = [\![\mathcal{M}.\mathcal{A}]\!]$. Somit ist das einzige semantisch nicht reduzierbare Konzept, das hier eingeführt wurde, die Unterteilung der Signale und Variablen eines Automaten in Eingabe-, Ausgabe-, mehrfach einschränkbare und lokale Komponenten. Im folgenden Abschnitt wird daher unter Semantik immer die Semantik eines Automaten verstanden. Es werden verschiedene Interpretationen für Timed Automata ausführlich dargestellt.

2.4 Interpretationen von Timed Automata

Verschiedene Anforderungen der algorithmischen Verifikation machen die Definition verschiedener Semantiken notwendig. Spur-Semantik und Erreichbarkeitssemantik bilden als letzte Voraussetzung für die algorithmische Verifikation den Abschluß des Abschnitts über Semantiken.

Ein Algorithmus für die Erreichbarkeitsanalyse bleibt korrekt, solange er eine Semantik benutzt, die bzgl. der erreichbaren Zustände *äquivalent* zur kontinuierlichen Semantik ist. Da es bei der Erreichbarkeitsanalyse um die Erreichbarkeit von Zuständen der Automaten geht, kann die Bedeutung eines Timed Automaton auch als Menge der erreichbaren Zustände betrachtet werden. Von einer Transitionssystem-Semantik eines Timed Automaton kann die Erreichbarkeitssemantik abgeleitet werden. Wird zum Zweck einer effizienteren algorithmischen Analyse eine neue Transitionssystem-Semantik für Timed Automata benutzt, so darf diese neue Semantik für einen Timed Automaton nicht zu einer anderen Erreichbarkeitssemantik führen.

In den nächsten Abschnitten wird eine Erreichbarkeitssemantik formalisiert, die Klasse der abgeschlossenen Timed Automata wird eingeführt und eine darauf basierende ganzzahlige, endliche Transitionssystem-Semantik definiert. Darüberhinaus wird bewiesen, daß dieses neue Transitionssystem zur gleichen Erreichbarkeitssemantik wie die kontinuierliche Semantik führt.

2.4.1 Semantik I: Kontinuierliches Transitionssystem

Die Bedeutung eines Timed Automaton wird dadurch definiert, daß ihm ein markiertes Transitionssystem zugeordnet wird. Die Konfigurationsmenge eines solchen Transitionssystems ist im allgemeinen nicht endlich.

Im folgenden werden Transitionssysteme in allgemeiner Form eingeführt und darauf aufbauend die traditionelle Transitionssystem-Semantik der Timed Automata vorgestellt.

Definition 2.12 *Ein* **markiertes Transitionssystem** *(kurz: Transitionssystem) ist ein Tupel* $(Q, Q^0, \Sigma, \rightarrow)$ *mit den folgenden Komponenten:*

- Q *ist eine (möglicherweise unendliche) Menge von* **Konfigurationen**.
- $Q^0 \subseteq Q$ *ist eine Menge von* **Initialkonfigurationen**.
- Σ *ist eine Menge von* **Marken**.
- $\rightarrow \subseteq Q \times \Sigma \times Q$ *ist eine Menge von* **Transitionen**.

Das System startet in einer der Initialkonfigurationen und kann seine Konfiguration von q nach q' mit der Marke a wechseln, falls $(q, a, q') \in \rightarrow$ (auch als $q \xrightarrow{a} q'$ geschrieben). $q \rightarrow q'$ wird geschrieben genau dann, wenn $q \xrightarrow{a} q'$ für eine Marke a gilt.

Definition 2.13 *Die* **kontinuierliche Transitionssystem-Semantik** $[\![\mathcal{A}]\!]_C$ *eines Timed Automaton* $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ *ist das markierte Transitionssystem* $(L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow)$*, wobei* $\rightarrow$ *zwei Arten von Transitionen enthält:*

- **Zeit-Transitionen:**
  Für $(l, v), (l', v') \in L \times Val(X)$ *und* $\delta \in \mathbb{R}_+$ *gilt* $(l, v) \xrightarrow{\delta} (l', v')$*,*
  gdw. $l' = l$*,* $v' = v + \delta$*,* $v \in [\![I(l)]\!]$ *und* $v' \in [\![I(l)]\!]$.
- **Diskrete Transitionen:**
  Für $(l, v), (l', v') \in L \times Val(X)$ *und* $a \in \Sigma$ *gilt* $(l, v) \xrightarrow{a} (l', v')$*,*
  gdw. ein $(l, a, \varphi, Y, l') \in E$ *existiert mit* $v \in [\![\varphi]\!]$ *und* $v' = v[Y := 0]$.

Anmerkung: Für alle Uhrenbedingungen $\varphi \in \Phi(X)$ gilt die Äquivalenz der folgenden zwei Aussagen:

- "$v \in [\![\varphi]\!]$ und $v + \delta \in [\![\varphi]\!]$" und
- "für alle $\delta' \in \mathbb{R}$ mit $0 \leq \delta' \leq \delta$ gilt $v + \delta' \in [\![\varphi]\!]$".

Dies ist der Fall, weil als Uhrenbedingungen nur Konjunktionen erlaubt sind.

2.4.2 Semantik II: Erreichbare Zustände

Beim Nachweis von Sicherheitseigenschaften per Erreichbarkeitsanalyse wird überprüft, ob eine bestimmte Menge von Kontrollzuständen von den Initialzuständen aus erreicht werden kann. Die Erreichbarkeitssemantik ist die Menge aller erreichbaren Kontrollzustände. Sie sagt aus, daß zwei Modelle dann die gleichen Erreichbarkeitseigenschaften erfüllen, wenn die Erreichbarkeitssemantiken äquivalent sind. So haben z. B. das Transitionssystem der kontinuierlichen Semantik (siehe Abschnitt 2.4.1) und das Transitionssystem der ganzzahligen Semantik (siehe folgenden Abschnitt 2.4.3) für einen bestimmten Automaten dieselbe Erreichbarkeitssemantik. Es wird also für einen Timed Automaton

$\mathcal{A}$ und eine zur algorithmischen Analyse verwendete Semantik $[\![.]\!]_I$ gefordert, daß die Erreichbarkeitssemantiken $[\![.]\!]_R$ übereinstimmen: $[\![[\![\mathcal{A}]\!]_C]\!]_R = [\![[\![\mathcal{A}]\!]_I]\!]_R$. Damit ist dann eine Semantik (wie z. B. $[\![.]\!]_I$), die eine effizientere algorithmische Verifikation ermöglicht, für alle Analysen gerechtfertigt, welche sich auf die Menge der erreichbaren Zustände[6] stützen.

Im folgenden werden Konfigurationsfolgen und erreichbare Konfigurationen für ein Transitionssystem definiert.

Definition 2.14 *Sei $\mathcal{S} = (Q, Q^0, \Sigma, \rightarrow)$ ein markiertes Transitionssystem, $(q_0, q_1, ..., q_k)$ eine endliche Folge von Konfigurationen, $\sigma_0, \sigma_1, ..., \sigma_{k-1} \in \Sigma$, $q_0 \in Q^0$ und $q_i \xrightarrow{\sigma_i} q_{i+1}$ für alle $i \in \{0, 1, ..., k-1\}$. Dann ist $(q_0, q_1, ..., q_k)$ (kurz: $(q_i)_{0 \leq i \leq k}$) eine* **Konfigurationsfolge** *von $\mathcal{S}$. $Run(\mathcal{S})$ bezeichnet die Menge aller Konfigurationsfolgen von $\mathcal{S}$. Die Konfiguration q_k ist* **erreichbar**. *$Reach(\mathcal{S})$ bezeichnet die Menge der erreichbaren Konfigurationen von $\mathcal{S}$. Für $t_1, t_2 \in \mathbb{N}$ wird die Konfiguration q_k erreichbar zwischen Zeit t_1 und Zeit t_2 genannt, falls $t_1 \leq \sum_{i \in I} \sigma_i \leq t_2$, wobei $I = \{j \mid 0 \leq j < k, \sigma_j \in (\Sigma \cap \mathbb{R}_+)\}$ die Menge aller Indizes der Zeitmarken von $\{\sigma_0, \sigma_1, ..., \sigma_{k-1}\}$ ist. $Reach(\mathcal{S})(t_1, t_2)$ bezeichnet die Menge der Konfigurationen, die zwischen Zeit t_1 und t_2 erreichbar sind.*

Da für die Verifikation von Sicherheitseigenschaften die Menge der erreichbaren Zustände von entscheidender Bedeutung ist, bzw. die Bedeutung eines Timed Automaton gerade in der Menge der erreichbaren Zustände liegt, kann für eine Transitionssystem-Semantik zu einem Timed Automaton eine abstraktere Semantik angegeben werden: die Erreichbarkeitssemantik.

Definition 2.15 *Sei $\mathcal{A}$ ein Timed Automaton und $[\![\mathcal{A}]\!]_C$ seine kontinuierliche Transitionssystem-Semantik. Ein (Kontroll-) Zustand $l \in \mathcal{A}.L$ ist* **erreichbar**, *falls eine Uhrenbelegung $v \in Val(\mathcal{A}.X)$ mit $(l, v) \in Reach([\![\mathcal{A}]\!]_C)$ existiert. Die dem Automaten $\mathcal{A}$ und der Semantik $[\![.]\!]_C$ zugeordnete* **Erreichbarkeitssemantik** *$[\![[\![\mathcal{A}]\!]_C]\!]_R$ ist die Menge der erreichbaren Zustände von $\mathcal{A}$ bezüglich des Transitionssystems $[\![\mathcal{A}]\!]_C$. Formal: $[\![[\![\mathcal{A}]\!]_C]\!]_R = \{l \mid \exists v : (l, v) \in Reach([\![\mathcal{A}]\!]_C)\}$.*

Definition 2.16 *Das* **Erreichbarkeitsproblem** *für Timed Automata besteht darin, für einen gegebenen Timed Automaton $\mathcal{A}$ und einen Zustand l_F zu entscheiden, ob $l_F \in [\![[\![\mathcal{A}]\!]_C]\!]_R$ gilt.*

Definition 2.17 *Zwei Transitionssystem-Semantiken $[\![.]\!]_1$ und $[\![.]\!]_2$ sind* **zustandsäquivalent** *(erreichbarkeitsäquivalent) für einen Timed Automaton $\mathcal{A}$ genau dann, wenn $[\![[\![\mathcal{A}]\!]_1]\!]_R = [\![[\![\mathcal{A}]\!]_2]\!]_R$ gilt.*

Die reellwertigen Uhrenbelegungen führen beim kontinuierlichen Transitionssystem immer zu einer unendlichen Konfigurationsmenge. Deshalb wird im folgenden eine Diskretisierung und Begrenzung dieses kontinuierlichen Zustandsraumes benutzt; es wird eine ganzzahlige Transitionssystem-Semantik definiert. Begonnen wird mit der Einführung der abgeschlossenen Timed Automata, einer Klasse von Timed Automata, für die die ganzzahlige Semantik zustandsäquivalent zur herkömmlichen Semantik ist. Abschließend wird diese Äquivalenz bewiesen.

[6] Darüberhinaus erhält die ganzzahlige Semantik auch abgeschlossene Uhrenbedingungen.

2.4.3 Semantik III: Ganzzahliges Transitionssystem

Abgeschlossene Timed Automata. Für die angestrebte BDD-basierte Verifikation ist eine Diskretisierung der Zeit erforderlich. Das kontinuierliche Vergehen von Zeit wird ersetzt durch diskrete Zeitschritte, wobei nur endlich viele diskrete Repräsentanten einer unendlichen, dichten Menge von Uhrenbelegungen betrachtet werden. Dieses Vorgehen wurde von Göllü et al. durch Äquivalenz-Sätze gerechtfertigt [GPV94]. Weiterhin wurde beobachtet [AMP98], daß es bei der Erreichbarkeitsanalyse der sogenannten abgeschlossenen Timed Automata genügt, nur ganzzahlige Uhrenbelegungen zu betrachten. Die Uhrenbedingungen der abgeschlossenen Timed Automata sind voneinander unabhängig und enthalten keine strikten Relationsoperatoren ($<$ und $>$).

Diese Arbeiten werden im folgenden erweitert, indem eine *ganzzahlige Semantik* für abgeschlossene Timed Automata formal definiert und die Äquivalenz zur herkömmlichen, kontinuierlichen Semantik bewiesen wird [BN01, BN00]. Die Äquivalenz bezieht sich darauf, daß in dieser Semantik die gleichen diskreten Zustände des Automaten erreicht werden wie in der herkömmlichen, kontinuierlichen Semantik. Deshalb kann bei der Erreichbarkeitsanalyse auf diese Repräsentation zurückgegriffen werden.

Eine zur kontinuierlichen Semantik zustandsäquivalente Diskretisierung der Zeit existiert für alle Timed Automata [GPV94]. Jedoch wird hier eine Einschränkung auf die Teilklasse der abgeschlossenen Timed Automata vorgenommen, um eine Diskretisierung anwenden zu können, die einerseits besonders einfach ist und andererseits eine effiziente Erreichbarkeitsanalyse ermöglicht. Für DBMs und ähnliche Datenstrukturen führt dies nicht zu signifikanten Performance-Verbesserungen. Diese Einschränkung ist technischer Natur, und es sind noch keine Beispiele innerhalb des vorgesehenen Anwendungsgebietes der Fertigungsanlagen und Realzeit-Algorithmen bekannt, für die es kompliziert wäre, Modelle mit rein abgeschlossenen Uhrenbedingungen und ganzzahligen Konstanten zu konstruieren.

Definition 2.18 Abgeschlossene Timed Automata *sind Timed Automata nach Definition 2.4, die nur solche Uhrenbedingungen φ beinhalten, die von der Grammatik*

$$\varphi := x \leq c \mid x \geq c \mid \varphi \wedge \varphi \mid true \mid false$$

erzeugt werden, wobei $x \in X$ und $c \in \mathbb{N}$, d. h. die Relationen $<$ und $>$ sind nicht erlaubt.

Der Produktautomat zweier abgeschlossener Timed Automata ist wieder abgeschlossen. Für abgeschlossene Timed Automata ist es ausreichend, bei der Berechnung der erreichbaren Zustände nur ganzzahlige Uhrenwerte zu berücksichtigen. Für eine Menge von Uhren X ist die Menge aller ganzzahligen Uhrenbelegungen $Val_I(X)$ definiert als die Menge der totalen Funktionen von X nach $\mathbb{N}$.

Für einen Timed Automaton $\mathcal{A}$ mit einer Uhr x bezeichnet $C_{\mathcal{A}}(x)$ die größte Konstante, mit der x in einer Uhrenbedingung von $\mathcal{A}$ verglichen wird. Für $v \in Val_I(X)$ und $\delta \in \mathbb{N}$ ist $v \oplus \delta$ diejenige Uhrenbelegung von X, die jeder Uhr x den Wert $min\,(v(x) + \delta, C_{\mathcal{A}}(x) + 1)$ zuordnet. Die Definition der ganzzahligen Semantik kann analog zur Definition der kontinuierlichen Semantik erfolgen.

Im folgenden wird nicht nur die Definition einer ganzzahligen Semantik angegeben, sondern auch ein formaler Beweis für die Äquivalenz bzgl. der Erreichbarkeitssemantik erbracht[7].

Definition 2.19 *Sei $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ ein abgeschlossener Timed Automaton. Die* **ganzzahlige Semantik (1-diskrete Semantik)** *$[\![\mathcal{A}]\!]_I$ von $\mathcal{A}$ ist das Transitionssystem $(L \times \mathit{Val}_I(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow_I)$ mit den folgenden Transitionen:*

- **Zeit-Transitionen:**
  Für $(l, v), (m, w) \in L \times \mathit{Val}_I(X)$ und $\delta \in \mathbb{N}$ gilt $(l, v) \xrightarrow{\delta}_I (m, w)$
  gdw. $l = m$, $w = v \oplus \delta$, $v \in [\![I(l)]\!]$ und $w \in [\![I(l)]\!]$.

- **Diskrete Transitionen:**
  Für $(l, v), (m, w) \in L \times \mathit{Val}_I(X)$ und $a \in \Sigma$ gilt $(l, v) \xrightarrow{a}_I (m, w)$
  gdw. ein $(l, a, \varphi, Y, m) \in E$ existiert mit $v \in [\![\varphi]\!]$ und $w = v[Y := 0]$.

Die Begriffe aus Definition 2.15 können analog auf die ganzzahlige Semantik angewendet werden.

Um die *Zustandsäquivalenz* der ganzzahligen und der kontinuierlichen Semantik zu beweisen, wird für einen Timed Automaton $\mathcal{A}$ mit der Menge von Uhren X die Relation $\succ \subseteq \mathit{Val}(X) \times \mathit{Val}_I(X)$ definiert, die jeder kontinuierlichen Uhrenbelegung ihre möglichen ganzzahligen **Repräsentanten** zuordnet. Für $v \in \mathit{Val}(X)$ und $v' \in \mathit{Val}_I(X)$ gilt $v \succ v'$, gdw. ein $\gamma \in \mathbb{R}$ existiert mit $0 \leq \gamma < 1$, so daß für jede Uhr $x \in X$ gilt:

a) $v'(x) - 1 + \gamma < v(x) \leq v'(x) + \gamma$ oder

b) $v'(x) - 1 + \gamma < v(x)$ und $v'(x) = C_{\mathcal{A}}(x) + 1$.

Somit ist v' ein Repräsentant von v, falls v' aus v durch ein Abrunden aller Uhrenwerte mit einem gebrochenen Teil unterhalb oder gleich einer bestimmten Grenze und ein Aufrunden aller Uhrenwerte mit einem gebrochenen Teil oberhalb dieser Grenze entsteht. Die zweite Bedingung der Definition schränkt den Wertebereich der Repräsentanten auf den Maximalwert $C_{\mathcal{A}}(x) + 1$ ein. Dies genügt, um die entscheidende Eigenschaft der Repräsentantenrelation sicherzustellen: Wenn eine kontinuierliche Uhrenbelegung v eine Uhrenbedingung von $\mathcal{A}$ erfüllt, dann erfüllt auch der ganzzahlige Repräsentant diese Uhrenbedingung. Zur Illustration der Repräsentantenrelation ist in Abbildung 2.6 auf der nächsten Seite die Menge der reellen Uhrenbelegungen dargestellt, für die der ganzzahlige Punkt $(2, 2)$ ein Repräsentant ist.

Beweise für die *Zustandsäquivalenz* der ganzzahligen Semantik zur kontinuierlichen Semantik für andere Formalismen als Timed Automata sind in der Literatur zu finden [Pop91, HMP92, AMP98]. Der Beweis für Timed Automata benutzt das folgende Lemma.

[7] Auch wenn es dabei um technische Details geht, ist dieser Beweis wichtig. So wurde z. B. in einer bekannten Publikation eine Diskretisierung ohne Beweis der Korrektheit vorgeschlagen [BMPY97], die sich bei genauerer Betrachtung als nicht zulässig erweist (siehe Abbildung 2.8 auf Seite 47 für das Gegenbeispiel).

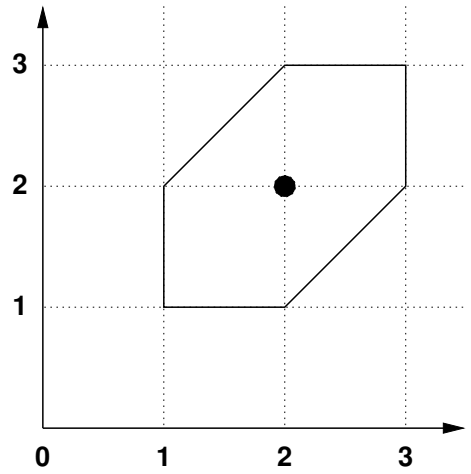


Abbildung 2.6: Menge der kontinuierlichen Uhrenbelegungen, die von der ganzzahligen Uhrenbelegung (2, 2) repräsentiert wird.

Lemma 2.3 *Sei $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ ein abgeschlossener Timed Automaton mit der kontinuierlichen Semantik $[\![\mathcal{A}]\!]_C = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow_C)$ und der ganzzahligen Semantik $[\![\mathcal{A}]\!]_I = (L \times Val_I(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow_I)$. Dann gilt:*

1. *Seien $(l, v), (l, w) \in L \times Val(X), \delta \in \mathbb{R}_+$ und es gelte $(l, v) \xrightarrow{\delta}_C (l, w)$. Dann existiert für alle $v' \in Val_I(X)$ mit $v \succ v'$ ein $\delta' \in \mathbb{N}$ und ein $w' \in Val_I(X)$, so daß $(l, v') \xrightarrow{\delta'}_I (l, w')$ und $w \succ w'$ gilt.*

2. *Seien $(l, v), (m, w) \in L \times Val(X), a \in \Sigma$ und es gelte $(l, v) \xrightarrow{a}_C (m, w)$. Dann existiert für alle $v' \in Val_I(X)$ mit $v \succ v'$ ein $w' \in Val_I(X)$ so daß $(l, v') \xrightarrow{a}_I (m, w')$ und $w \succ w'$ gilt.*

Bevor der formale Beweis des Lemmas gegeben wird, erfolgt zunächst eine intuitive Beschreibung der ersten Aussage des Lemmas. Dafür wird das Beispiel von Abbildung 2.7 auf der nächsten Seite betrachtet, welches eine Transition der Zeit $\delta = 0.8$ von $(0.8, 1.3)$ zu $(1.6, 2.1)$ zeigt. Der ganzzahlige Punkt $(1, 1)$ ist ein Repräsentant des Startpunktes $(0.8, 1.3)$ für z. B. $\gamma = 0.6$.

Wird von $(0.8, 1.3)$ zu $(1.6, 2.1)$ übergegangen, dann ist der Punkt $(1.5, 2.0)$ der erste kontinuierliche Punkt, der nicht von $(1, 1)$ repräsentiert wird. Die Punkte ab unmittelbar nach $(1, 1.5)$ bis zum Endpunkt $(1.6, 2.1)$ sind durch $(2, 2)$ repräsentiert. Wird $\delta' = 1$ gewählt, resultiert dies in einer Transition von $(1, 1)$, einem Repräsentanten des Startpunktes $(0.8, 1.3)$, zu einem Repräsentanten des Endpunktes $(1.6, 2.1)$.

Allgemein gilt, daß der Endpunkt der δ'-Transition ein Repräsentant des Endpunktes der δ-Transition ist, falls $\delta' = \lfloor \delta + \gamma \rfloor$ gewählt wird.

Beweis: Aussage 1 des Lemmas: Es müssen die beiden Fälle der Definition von $\succ$ unterschieden werden. Sei $v' \in Val_I(X)$, $v \succ v'$. Dann existiert entsprechend der Definition der Relation $\succ$ ein $\gamma \in \mathbb{R}$ mit $0 \leq \gamma < 1$, so daß für alle $x \in X$ gilt:

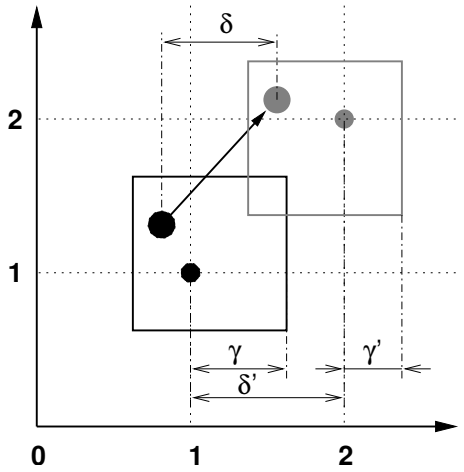


Abbildung 2.7: Diskretisierung und Repräsentanten.

Fall a) $v'(x) + \gamma + \delta < C_{\mathcal{A}}(x) + 1$:

$$v'(x) - 1 + \gamma \quad < \quad v(x) \quad \leq \quad v'(x) + \gamma.$$

Weil $w = v + \delta$, gilt das folgende für alle $x \in X$:

$$v'(x) - 1 + \gamma + \delta \quad < \quad w(x) \quad \leq \quad v'(x) + \gamma + \delta.$$

Sei $\delta' = \lfloor \delta + \gamma \rfloor$ und $w' = v' + \delta'$. Dann gilt für alle $x \in X$:

$$w'(x) - \delta' - 1 + \gamma + \delta \quad < \quad w(x) \quad \leq \quad w'(x) - \delta' + \gamma + \delta.$$

Da $0 \leq \gamma + \delta - \delta' < 1$, impliziert dies $w \succ w'$.

Fall b) $v'(x) + \gamma + \delta \geq C_{\mathcal{A}}(x) + 1$:
Aus $\delta' = \lfloor \delta + \gamma \rfloor$ und $w' = v' \oplus \delta'$ folgt analog zu Fall a)

$$w'(x) - \delta' - 1 + \gamma + \delta \quad < \quad w(x),$$

und damit $w \succ w'$.

Weil v und w die Invariante $I(l)$ erfüllen, und $v \succ v'$ und $w \succ w'$ gelten, kann geschlußfolgert werden, daß v' und w' die Invariante $I(l)$ erfüllen. Somit gilt $(l, v') \xrightarrow{\delta'}_I (l, w')$.

Aussage 2 des Lemmas: Aufgrund der Definition der kontinuierlichen Semantik existiert ein $(l, a, \varphi, Y, m) \in E$ mit $v \in [\![\varphi]\!]$ und $w = v[Y := 0]$. Sei $v' \in Val_I(X)$, $v \succ v'$. Dann gilt $v' \in [\![\varphi]\!]$, weil $v \in [\![\varphi]\!]$. $\quad v \succ v'$ impliziert $v[Y := 0] \succ v'[Y := 0]$. Wird w' auf $v'[Y := 0]$ gesetzt, folgt Aussage 2. □

Satz 2.4 *Für jeden abgeschlossenen Timed Automaton $\mathcal{A}$ gilt $[\![[\![\mathcal{A}]\!]_C]\!]_R = [\![[\![\mathcal{A}]\!]_I]\!]_R$.*

Beweis: Sei $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ ein abgeschlossener Timed Automaton mit der kontinuierlichen Semantik $[\![\mathcal{A}]\!]_C = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow_C)$ und der ganzzahligen Semantik $[\![\mathcal{A}]\!]_I = (L \times Val_I(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow_I)$.

Als erstes wird $[\![[\![\mathcal{A}]\!]_C]\!]_R \subseteq [\![[\![\mathcal{A}]\!]_I]\!]_R$ bewiesen. Es wird per Induktion über k gezeigt, daß für jede Konfigurationsfolge $((l_0, v_0), (l_1, v_1), ..., (l_k, v_k))$ in $Run([\![\mathcal{A}]\!]_C)$ eine Konfigurationsfolge $((l_0, v'_0), (l_1, v'_1), ..., (l_k, v'_k))$ in $Run([\![\mathcal{A}]\!]_I)$ existiert, so daß $v_i \succ v'_i$ für alle $i \in \{0, 1, ..., k\}$ gilt.

Induktionsanfang: Nach der Definition einer Konfigurationsfolge gilt $l_0 \in L^0$ und $v_0 = v^0$. $((l_0, v^0))$ ist ebenfalls in $Run([\![\mathcal{A}]\!]_I)$, und $v_0 \succ v^0$ gilt.

Induktionsschritt: Es ist zu zeigen, daß ein $a' \in \Sigma \cup \mathbb{N}$ und ein v'_{i+1} mit $v_{i+1} \succ v'_{i+1}$ existiert, so daß $(l_i, v'_i) \xrightarrow{a'}_I (l_{i+1}, v'_{i+1})$. Die Induktionsbehauptung stellt $v_i \succ v'_i$ sicher und daß ein $a \in \Sigma \cup \mathbb{R}_+$ mit $(l_i, v_i) \xrightarrow{a}_R (l_{i+1}, v_{i+1})$ existiert. Die Behauptung des Satzes folgt aus Lemma 2.3, Aussage 1, falls $a \in \mathbb{R}_+$, und aus Aussage 2, falls $a \in \Sigma$. Damit ist der Induktionsbeweis abgeschlossen.

$[\![[\![\mathcal{A}]\!]_I]\!]_R \subseteq [\![[\![\mathcal{A}]\!]_C]\!]_R$ folgt direkt aus der Definition der Semantik. □

Unzulässige Diskretisierungen. Bei der Diskretisierung von allgemeinen Timed Automata ist die Schrittweite der Diskretisierung immer abhängig von der Anzahl der Uhren, die vom Automaten benutzt werden. Um dies zu zeigen, wird im folgenden eine diskrete Semantik definiert, die auf einer Schrittweite von $\frac{1}{k}$ für einen festen Wert k beruht. Im Anschluß daran wird nachgewiesen, daß diese Semantik für die Erreichbarkeitsanalyse nicht zulässig ist.

Definition 2.20 *Sei $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ ein Timed Automaton. Bezeichne $\mathbb{N}_k$ für alle $k \in \mathbb{N}$, $k \geq 1$ die Menge $\{\frac{i}{k} \mid i \in \mathbb{N}\}$. Für eine Uhrenmenge X sei die Menge der $\frac{1}{k}$-diskreten Uhrenbelegungen $Val_{D_k}(X)$ definiert als die Menge der totalen Abbildungen von X nach $\mathbb{N}_k$.*
*Für eine Schrittweite $\frac{1}{k}$ mit $k \in \mathbb{N}$, $k \geq 1$ ist die $\frac{1}{k}$-**diskrete Semantik** $[\![\mathcal{A}]\!]_{D_k}$ von $\mathcal{A}$ das Transitionssystem $(L \times Val_{D_k}(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}_k, \rightarrow_{D_k})$ mit folgenden Transitionen:*

- **Zeit-Transitionen:**
  Für $(l, v) \in L \times Val_{D_k}(X)$, $\delta \in \mathbb{N}_k$ gilt $(l, v) \xrightarrow{\delta}_{D_k} (l, v + \delta)$,
  gdw. $v \in [\![I(l)]\!]$ und $v + \delta \in [\![I(l)]\!]$.
- **Diskrete Transitionen:**
  Für $(l, v) \in L \times Val_{D_k}(X)$, $(l, a, \varphi, Y, l') \in E$ gilt $(l, v) \xrightarrow{a}_{D_k} (l', v[Y := 0])$,
  gdw. $v \in [\![\varphi]\!]$.

Satz 2.5 *Für jede Schrittweite $\frac{1}{k}$ mit $k \in \mathbb{N}$, $k \geq 1$ existiert ein Timed Automaton $\mathcal{A}$ mit zwei Uhren, so daß gilt $[\![[\![\mathcal{A}]\!]_C]\!]_R \neq [\![[\![\mathcal{A}]\!]_{D_k}]\!]_R$.*

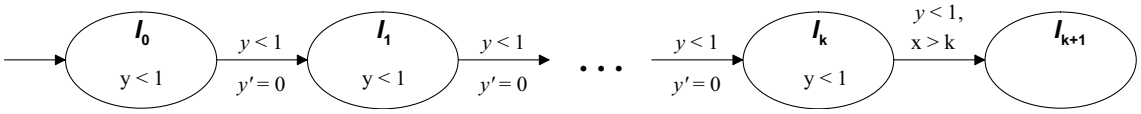


Abbildung 2.8: Gegenbeispiel für die $\frac{1}{k}$-diskrete Semantik.

Beweis: Für den Timed Automaton in Abbildung 2.8 ist der Zustand l_{k+1} bei kontinuierlicher Semantik erreichbar, zum Beispiel durch die Konfigurationsfolge $\Big((l_0, \{(x,0),(y,0)\}), (l_0, \{(x, \frac{k+1}{k+2}), (y, \frac{k+1}{k+2})\}), (l_1, \{(x, \frac{k+1}{k+2}), (y,0)\})$, ..., $(l_k, \{(x, \frac{(k+1)(k+1)}{k+2}), (y, \frac{k+1}{k+2})\}), (l_{k+1}, \{(x, \frac{(k+1)(k+1)}{k+2}), (y, \frac{k+1}{k+2})\})\Big)$.

Bei der $\frac{1}{k}$-diskreten Semantik ist in jeder erreichbaren Konfiguration mit dem Zustand l_k der Wert der Uhr x höchstens $\frac{(k+1)(k-1)}{k}$, also kleiner als k. Der Zustand l_{k+1} ist damit nicht erreichbar. □

Die Konfigurationsmenge $L \times Val_{D_k}(X)$ der $\frac{1}{k}$-diskreten Semantik enthält nur dann mindestens einen Repräsentanten jeder Region, wenn $k \geq |X| + 1$ gilt. Es sind also für Timed Automata mit vielen Uhren besonders kleine Schrittweiten $\frac{1}{k}$ erforderlich, um die Zustandsäquivalenz von kontinuierlicher und $\frac{1}{k}$-diskreter Semantik zu gewährleisten. Besonders interessant ist deshalb die Aussage des Satzes 2.5, daß schon Timed Automata *mit nur zwei Uhren beliebig kleine Schrittweiten* erfordern können.

In [BMPY97] wurde die Teilklasse der **halboffenen Timed Automata** definiert, deren Uhrenbedingungen nur die Relationszeichen $\geq$ und $<$ enthalten dürfen. Die Behauptung in dieser Publikation lautete, daß es eine Schrittweite $\frac{1}{k}$ gibt, für die die kontinuierliche und die $\frac{1}{k}$-diskrete Semantik zustandsäquivalent sind. Diese Behauptung kann widerlegt werden, indem in Abbildung 2.8 auf der vorherigen Seite die Bedingung $x > k$ des Zustandsübergangs von l_k nach l_{k+1} durch $x \geq k$ ersetzt wird. Dann handelt es sich um einen halboffenen Timed Automata, was jedoch ohne Auswirkungen auf den Beweis von Satz 2.5 bleibt. Auf den letzten Seiten wurde gezeigt, daß die 1-diskrete Semantik für abgeschlossene Timed Automata zustandsäquivalent zur kontinuierlichen Semantik ist.

2.4.4 Semantik IV: Sichtbare Spuren

Vielfach besteht die Anforderung, das Verhalten eines Modells nach außen in einer abstrakteren Weise zu verstehen und zu definieren als dies durch eine der Transitionssystem-Semantiken geschieht, da in diesen auch Informationen über die nur lokal sichtbaren Komponenten (Zustände der Automaten, lokale Uhrenbelegungen) enthalten sind.

Die Spur-Semantik definiert das Verhalten eines Moduls als Menge aller möglichen Sequenzen von Synchronisationsmarken, die durch Schalten der Übergänge des Transitionssystems entstehen. Diese Semantik wird insbesondere für die Verfeinerungsanalyse genutzt: Ein Modul verfeinert ein anderes, wenn es spezialisierteres Verhalten besitzt, d. h. die Spur-Semantik des verfeinerten Moduls ist eine Teilmenge der Spur-Semantik des abstrakteren Moduls.

Definition 2.21 *Sei $(Q, Q^0, \Sigma, \rightarrow)$ ein markiertes Transitionssystem. Ein* **Lauf** *ist eine unendliche Folge $(q_0, \sigma_0, q_1, \sigma_1, q_2, \sigma_2, \dots)$ (kurz: $(q_i, \sigma_i)_{i \in \mathbb{N}}$) mit $q_0 \in Q^0$ und $\forall i \in \mathbb{N} : q_i \in Q, \sigma_i \in \Sigma, q_i \xrightarrow{\sigma_i} q_{i+1}$.*

Sei $\mathcal{M}$ ein CTA-Modul und $[\![\mathcal{M}.\mathcal{A}]\!] = (Q, Q^0, \Sigma, \rightarrow)$ sein markiertes Transitionssystem. Ein **Sichtlauf** *ist eine unendliche Folge $(q_i, \sigma_i)_{i \in \mathbb{N}}$ mit $q_0 \in Q^0$ und $\forall i \in \mathbb{N} : q_i \in Q$, $\sigma_i \in \Sigma \setminus \mathcal{M}.par_\Sigma^{-1}(\mathtt{L}), q_i \overset{\sigma_i}{\rightarrowtail} q_{i+1}$. $\rightarrowtail$ ist die Hülle bezüglich der internen Transitionen; formal wird $\rightarrowtail$ induktiv definiert, mit $\sigma \in \Sigma \setminus \mathcal{M}.par_\Sigma^{-1}(\mathtt{L}), \tau \in \mathcal{M}.par_\Sigma^{-1}(\mathtt{L})$ und:*

- $q \xrightarrow{\sigma} q' \;\Rightarrow\; q \overset{\sigma}{\rightarrowtail} q'$
- $q \overset{\sigma}{\rightarrowtail} q'' \wedge q'' \xrightarrow{\tau} q' \;\Rightarrow\; q \overset{\sigma}{\rightarrowtail} q'$

- $q \xrightarrow{\tau} q'' \wedge q'' \overset{\sigma}{\rightarrowtail} q' \;\Rightarrow\; q \overset{\sigma}{\rightarrowtail} q'$

Definition 2.22 *Sei $\mathcal{T}$ ein markiertes Transitionssystem. Dann ist die unendliche Folge $(\sigma_i)_{i\in\mathbb{N}}$ eine* **Spur** *genau dann, wenn ein Sichtlauf $(q_i, \sigma_i)_{i\in\mathbb{N}}$ für $\mathcal{T}$ existiert.*

Anmerkung: Die Transitionsrelation $\rightarrow$ enthält Transitionen mit lokalen Marken, sogenannte stille Transitionen[8]. Diese Marken werden in der Spur-Semantik nicht berücksichtigt, da sie für die Umgebung nicht von Bedeutung sind.

Definition 2.23 *Sei $\mathcal{T}$ ein markiertes Transitionssystem. Dann ist die* **Spur-Semantik** *$[\![\mathcal{T}]\!]_{\mathcal{L}}$ definiert als die Menge aller Spuren des Transitionssystems $\mathcal{T}$. Für zwei Spur-Semantiken $[\![\mathcal{T}]\!]_{\mathcal{L}}$ und $[\![\mathcal{T}']\!]_{\mathcal{L}}$ wird die Komposition als Durchschnitt definiert: $[\![\mathcal{T}]\!]_{\mathcal{L}} \,||\, [\![\mathcal{T}']\!]_{\mathcal{L}} =_{def} [\![\mathcal{T}]\!]_{\mathcal{L}} \cap [\![\mathcal{T}']\!]_{\mathcal{L}}$.*

2.4.5 Beziehung der verschiedenen Semantiken zueinander

Unter Semantik wird in dieser Arbeit eine abstrahierende Repräsentation verstanden, die die interessierenden Eigenschaften besonders hervorhebt bzw. eine Analyse technisch ermöglicht. In Abbildung 2.9 auf der nächsten Seite werden die verschiedenen Semantiken entsprechend dem Grad ihrer Abstraktheit in einer Hierarchie angeordnet. Der Automat stellt die Ausgangsstruktur dar. Aus diesem Automaten wird zunächst das kontinuierliche Transitionssystem als Standardsemantik erzeugt[9]. Dieses bildet die herkömmliche semantische Basis für weitere Betrachtungen. Für abgeschlossene Timed Automata kann eine ganzzahlige, endliche Semantik definiert werden: das ganzzahlige Transitionssystem. Betrachtet man die Transitionssysteme auf der Ebene der erzeugten Sprachen, dann entsteht zunächst die Menge der Läufe eines Automaten. Für die Erreichbarkeitsanalyse, insbesondere bei Timed Automata, ist die maximale Abstraktion von der ursprünglichen Automatenstruktur durch die Erreichbarkeitssemantik (erreichbare Zustände) gegeben. Für die Verfeinerungsanalyse ist es erforderlich, Signalfolgen genau zu betrachten. Deshalb ist für diesen Zweck die Spur-Semantik diejenige mit der maximalen Abstraktion.

2.5 Zusammenfassung

Die Modellierung von Realzeit-Systemen mit Timed Automata ist in den letzten Jahren im wissenschaftlichen Bereich populär geworden. Der Formalismus bietet eine klare Semantik und die theoretischen Grundlagen sind gut untersucht [AD94]. Durch die graphische Darstellung kann mit Timed Automata anschaulich modelliert werden und die Modelle sind gut verständlich.

Die in der Literatur vorgeschlagenen, auf Timed Automata basierenden Formalismen zur Modellierung von Realzeit-Systemen bieten jedoch unzureichende Unterstützung für

[8] In der Literatur werden diese lokalen Signale *silent action* genannt (z. B. in [WL97]).

[9] Für hybride Automaten ist die kompositionelle Update-Semantik als zusätzliche Schicht zwischen Automat und kontinuierlichem Transitionssystem sinnvoll. Für Timed Automata ist diese Semantik trivial und wird deswegen erst in Kapitel 4 auf hybride Automaten bezogen definiert.

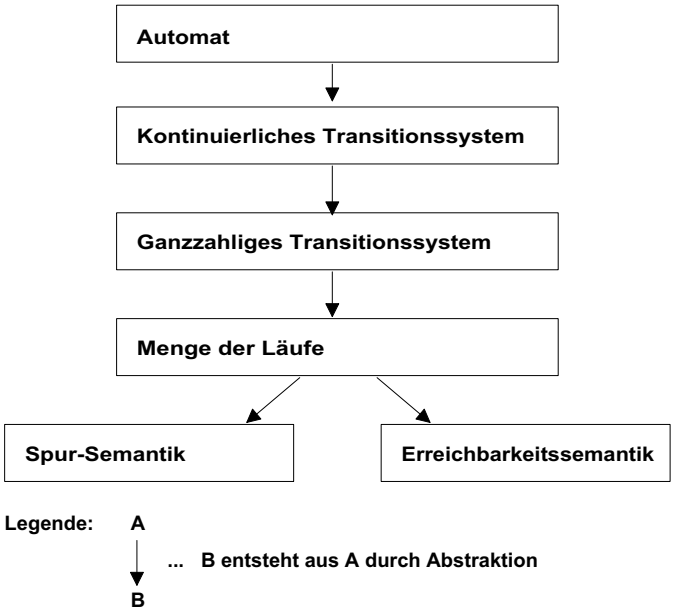


Abbildung 2.9: Abstraktionshierarchie der verschiedenen Semantiken.

die Konstruktion *größerer* Modelle. Bereits bei Systemen, die aus wenigen Automaten bestehen, ist es schwierig, die dem Modell zugrunde liegende Struktur zu erkennen und zu verstehen.

Aus diesem Grund wurde eine Erweiterung vorgenommen, aus der ein neuer Formalismus entstanden ist: Cottbus Timed Automata [BR98, BR01]. Die Stärke dieser Beschreibungsnotation liegt im Modulkonzept, das eine modulare, hierarchische Strukturierung des Modells sowie Wiederverwendbarkeit und Austauschbarkeit von System-Komponenten gut unterstützt. Damit wurde der *erste Schritt zur Bewältigung großer Systeme* realisiert.

Ein weiterer wesentlicher Beitrag besteht in der Definition einer ganzzahligen Diskretisierung des reellwertigen Zustandsraumes der Timed Automata (in Erweiterung von [ABK$^+$97, AMP98]). Die Korrektheit dieser ganzzahligen Semantik für die Erreichbarkeitsanalyse wurde formal bewiesen [Bey01b, BN01]. Die vier wichtigsten Semantiken für Timed Automata wurden in einer einheitlichen Darstellung formal beschrieben: kontinuierliches Transitionssystem, Menge erreichbarer Zustände, ganzzahliges Transitionssystem, Menge von Spuren.

Damit erfüllen die Cottbus Timed Automata die im Kapitel 1 aufgestellte Forderung 1, und es stehen präzise Semantiken zur Verfügung, auf deren Grundlagen eine effiziente Verifikation erfolgen kann. Eine Beschreibung der Modellierungssyntax und mehrere Modellierungsbeispiele folgen im Kapitel 5.

Kapitel 3

Effiziente Verifikation von Cottbus Timed Automata

In diesem Kapitel werden die Konzepte für eine effiziente BDD-basierte Implementierung der Verifikation von CTA-Modellen vorgestellt. Nach einer kurzen Diskussion des Algorithmus für die Erreichbarkeitsanalyse werden die grundsätzlichen Belange einer auf Binary Decision Diagrams basierenden Repräsentation erörtert: Die Datenstruktur BDD wird definiert und erklärt, auf die Darstellung von Belegungsmengen diskreter Variablen und das Problem des Variablenordnens wird eingegangen. Am Ende des zweiten Abschnitts wird erklärt, wie Transitionsrelationen und Erreichbarkeitsmengen der ganzzahligen Semantik konkret repräsentiert werden.

Der Kern des Kapitels besteht in einem Beitrag zur Lösung des Problems, eine gute Variablenordnung zu finden. Der Kommunikationsgraph des Modells ist die Grundlage der Überlegungen. Er gibt Auskunft über die Modellstruktur und die Abhängigkeiten innerhalb des Modells. Eine obere Schranke für die Repräsentationsgröße der Transitionsrelation, die durch einen formalen Beweis abgesichert ist, bildet die Basis einer Schätzfunktion für die Repräsentationsgröße der Erreichbarkeitsmenge. Dadurch kann aus einer Menge möglicher Variablenordnungen die nach der Bewertung beste Ordnung ausgewählt werden. An einem Beispiel wird die Wirkungsweise der Schätzfunktion genau erläutert [Bey01b].

Immer populärer auf dem Gebiet der formalen Verifikation werden Techniken des modularen Beweisens und der Kompositionalität [dR98, dRLP98]. Die Effizienz der vorgeschlagenen Implementierung der Erreichbarkeitsanalyse löst zwar das Problem des exponentiellen Aufwands der Verifikation in vielen Fällen (vgl. Abschnitt 5.4.2), jedoch erreicht auch hier das reine Model-Checking bei sehr großen Modellen mit komplexer Kommunikationsstruktur seine Grenzen. Daher wird eine Verfeinerungsanalyse zur Kombination mit der Erreichbarkeitsanalyse vorgeschlagen, mit der durch modulares Beweisen in mehreren Schritten auch Modelle verifiziert werden können, die mit reiner Erreichbarkeitsanalyse nicht handhabbar wären.

Die Effizienz der BDD-Repräsentation ist von mehreren Parametern abhängig, die im letzten Abschnitt des Kapitels beschrieben werden: Es wird auf die besondere Rolle der Zeit-Transition eingegangen, verschiedene Alternativen für die Repräsentation der einzel-

nen diskreten Transitionen diskutiert, sowie eine On-the-fly-Analyse definiert und deren Korrektheit bewiesen. Weitere Parameter wie die Ordnung der diskreten Transitionen, die Wahl der Hash-Funktion für die Hash-Tabelle und die Behandlung der inaktiven Uhren können erheblich zur Performancesteigerung beitragen. Die Effektivität der in diesem Kapitel vorgeschlagenen Konzepte wird anhand einiger Benchmark-Beispiele im Kapitel 5 und einer großen Fallstudie in Kapitel 6 verdeutlicht.

Die Hauptaufgabe dieses Kapitels besteht in der Bereitstellung aller erforderlichen Konzepte für die Implementierung eines effizienten Werkzeugs. Es sollen nicht nur stark vereinfachte Aufsatz-Beispiele, sondern auch praxisrelevante, komplizierte Systeme, wie etwa die Modell-Fertigungsanlage, verifiziert werden können.

3.1 Erreichbarkeitsanalyse

Eingabe: Timed Automaton $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$
mit der kontinuierlichen Semantik $[\![\mathcal{A}]\!]_C = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow)$,
Menge von Zuständen L^F
Ausgabe: $true$ gdw. $L^F \cap [\![[\![\mathcal{A}]\!]_C]\!]_R \neq \emptyset$

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

Abbildung 3.1: Algorithmus für die Erreichbarkeitsanalyse.

In diesem Abschnitt werden die Ziele, Möglichkeiten und Algorithmen der Erreichbarkeitsanalyse vorgestellt. Ein generischer Algorithmus für die Erreichbarkeitsanalyse wird in Abbildung 3.1 gezeigt. Der Algorithmus prüft für einen gegebenen Timed Automaton $\mathcal{A}$, ob er einen Zustand aus einer gegebenen Menge L^F erreichen kann. Während der Ausführung ist die Variable R die Menge aller bereits erreichten Konfigurationen. Jede Iteration des Algorithmus fügt die Menge $\{q' \in L \times Val(X) \mid \exists q : q \in R \wedge q \rightarrow q'\}$ der Konfigurationen, die von R aus mit einer Transition erreichbar sind, zu R hinzu. Der Algorithmus bricht ab, falls eine der folgenden Bedingungen erfüllt ist:

1. Es wurde eine Konfiguration erreicht, deren Zustand in der Menge der (verbotenen) Zustände L^F enthalten ist. Dann stoppt der Algorithmus und meldet eine Verletzung der Sicherheitseigenschaft.

2. Der Fixpunkt $R = R_{prev}$ wurde erreicht. Dann enthält R alle erreichbaren Konfigurationen. Falls in R keine Konfiguration mit einer Zustandskomponente aus L^F enthalten ist (Fall 1), dann ist die Sicherheitseigenschaft erfüllt.

Anmerkung: Hier wird davon ausgegangen, daß die Erreichbarkeitsanalyse zum Nachweis von Sicherheitseigenschaften und somit des Nichterreichens von ”Fehlerzuständen” angewendet wird (Negativtest). Es ist genauso sinnvoll, mittels Erreichbarkeitsanalyse zu überprüfen, ob bestimmte Konfigurationen erreicht werden können (Positivtest).

3.2 BDD-Repräsentation – Zweiter Schritt zur Bewältigung großer Systeme

Im folgenden werden die für die Erfüllung von Forderung 2 auf Seite 14 notwendigen Überlegungen genauer beschrieben. Der erste Abschnitt führt die Binary Decision Diagrams (BDDs) ein. Es wird beschrieben, wie Belegungsmengen von Booleschen Variablen als BDDs repräsentiert werden können. Im zweiten Abschnitt werden einige Notationen für diskrete Variablen eingeführt, die eine Generalisierung der Booleschen Variablen darstellen. Der letzte Abschnitt zeigt, daß Mengen von Konfigurationen und Transitionen der ganzzahligen Semantik als Belegungsmengen diskreter Variablen betrachtet werden können und wie die BDD-Repräsentationen dieser Belegungsmengen berechnet werden.

3.2.1 Binary Decision Diagrams – Einführung

Ein Binary Decision Diagram (BDD) [Bry86] repräsentiert eine Menge von Belegungen für eine Menge von Booleschen Variablen. BDDs ermöglichen eine kanonische und kompakte Repräsentation von Mengen und erlauben eine effiziente Implementierung von Operationen wie Durchschnitt, Vereinigung und Existenzquantifikation.

Ein BDD ist ein gerichteter, azyklischer Graph, der durch Reduktion von einem binären Entscheidungsbaum abgeleitet wird. Ein binärer Entscheidungsbaum besteht aus Entscheidungsknoten, 0-Terminalknoten und 1-Terminalknoten. Jeder Entscheidungsknoten ist einer Booleschen Variable zugeordnet und hat zwei Kinder, genannt Low-Kind und High-Kind.

Die Belegungen, die vom Entscheidungsbaum repräsentiert werden, entsprechen den Pfaden vom Wurzelknoten zu den 1-Terminalknoten. Die Variable eines Knotens hat den Wert 0, falls der Pfad zu einem Low-Kind führt, und den Wert 1, falls der Pfad zu einem High-Kind führt.

Die Bottom-Up-Anwendung der folgenden zwei Reduktionsregeln transformiert einen binären Entscheidungsbaum in einen BDD:

1. **Isomorphieregel:** Verschmelzen aller isomorphen Teilbäume.
2. **Eliminationsregel:** Eliminieren aller Knoten, deren Kinder isomorph sind.

In dieser Arbeit werden nur geordnete BDDs (auch OBDD genannt) verwendet, bei denen die Variablen in jedem Pfad von der Wurzel zu einem Terminalknoten in der gleichen Reihenfolge auftreten. Für eine gegebene Variablenordnung ist die Repräsentation einer Menge von Belegungen eindeutig.

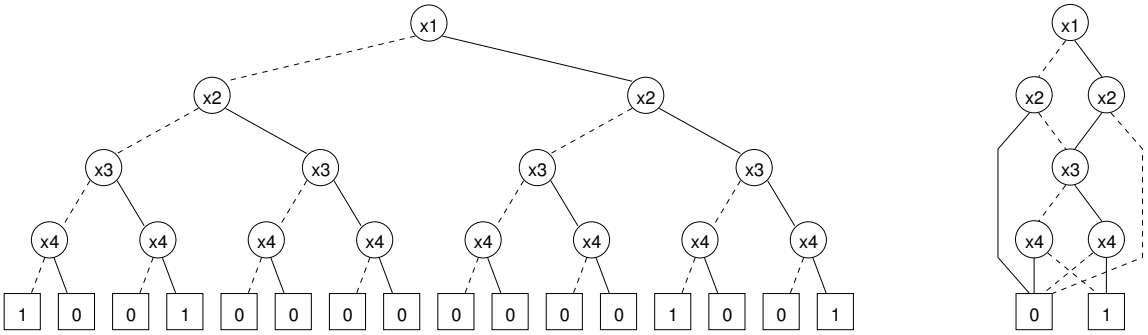


Abbildung 3.2: Entscheidungsbaum und BDD für eine Menge von Belegungen.

Abbildung 3.2 zeigt den Entscheidungsbaum und die BDD-Repräsentation der Menge aller Belegungen der Booleschen Variablen $\{x_1, x_2, x_3, x_4\}$ mit $v(x_1) = v(x_2)$ und $v(x_3) = v(x_4)$. Eine gestrichelte Kante führt zu einem Low-Kind, eine durchgehende zu einem High-Kind.

Anmerkung: Die grundsätzliche Idee, aus der später die BDDs entwickelt wurden, war die Shannon-Entwicklung, bei der eine Schaltfunktion durch die Belegung einer Variablen in die Kombination zweier Unterfunktionen (Kofaktoren) zerlegt werden kann[1]. Werden diese Unterfunktionen als Teilbäume betrachtet, entsteht ein binärer Entscheidungsbaum. Die Binary Decision Diagrams wurden 1959 von Lee eingeführt [Lee59]. Bekannt gemacht und genauer studiert wurden die BDDs von Akers [Ake78]. Dabei handelte es sich um allgemeine binäre Entscheidungsgraphen. Das volle Potential für die algorithmische Arbeit mit dieser Datenstruktur wurde erst 1986 von Bryant untersucht: es liegt in der Benutzung einer feststehenden Variablenordnung (die zu einer kanonischen Darstellung führt) und einer komprimierten Darstellung, wodurch erstaunlich effiziente Algorithmen ermöglicht werden [Bry86, Bry92]. Durch das Ausdehnen der Isomorphieregel auf verschiedene BDDs, d. h. das Benutzen eines Teilgraphen in mehreren verschiedenen BDDs, wird die Datenstruktur "Shared Reduced Ordered Binary Decision Diagram" definiert [BRB90]. Mit der Abkürzung BDD ist im allgemeinen diese Datenstruktur gemeint.

3.2.2 Diskrete Variablen

Im folgenden werden einige Notationen zu Booleschen und den allgemeineren diskreten Variablen sowie ihrer Belegungen angegeben, um formal definieren zu können, wie ein BDD eine Menge von Belegungen repräsentiert.

Diskrete Variablen sind Variablen mit einem endlichen und nicht leeren Wertebereich. Ohne Beschränkung der Allgemeinheit werden nur Wertebereiche zugelassen, deren Anzahl von Elementen eine Zweierpotenz ist; Wertebereiche, für die das nicht gilt, werden

[1] Shannon wandte die grundlegenden Regeln der klassischen Logik und der elementaren Mengenlehre bereits 1938 zur Beschreibung von digitalen Schaltungen an [Sha38].

mit ungenutzten Elementen aufgefüllt. Boolesche Variablen sind spezielle diskrete Variablen mit dem Wertebereich $\{0, 1\}$.

Ein BDD kann nicht nur Belegungsmengen Boolescher Variablen repräsentieren, sondern auch Belegungsmengen diskreter Variablen. Dafür wird jede diskrete Variable x mit mehr als zwei Werten im Wertebereich durch mehrere Boolesche Variablen kodiert.

Analog zu Abschnitt 2.1 werden die Begriffe Bedingung und Belegung für die folgenden Abschnitte auf diskrete Variablen übertragen. Sei X eine Menge diskreter Variablen, und bezeichne $Range(x)$ den Wertebereich einer Variable $x \in X$. Die Menge $\Phi(X)$ von **Bedingungen** über X wird erzeugt von der Grammatik $\varphi := x_1 \sim c \mid x_1 \sim x_2 \mid \varphi \wedge \varphi$, mit $x_1, x_2 \in X$, $\sim \in \{\leq, \geq, <, >, =, \neq\}$, $Range(x_1) = Range(x_2)$ und $c \in Range(x_1)$. Falls $Range(x_1)$ keine total geordnete Menge ist, ist nur $\sim \in \{=, \neq\}$ erlaubt.

Eine **Belegung** v **von** X ist eine totale Funktion, die jeder Variablen $x \in X$ ein Element aus $Range(x)$ zuordnet. Wie in Abschnitt 2.1 bezeichnet $Val(X)$ die Menge aller Belegungen von X, und für eine Bedingung $\varphi \in \Phi(X)$ bezeichnet $[\![\varphi]\!]$ die Menge aller Belegungen von X, die φ erfüllen. Diese Festlegung ist mehrdeutig, da für zwei Mengen von Variablen X und Y gilt: $\varphi \in \Phi(X)$ impliziert $\varphi \in \Phi(X \cup Y)$. ($\varphi$ bezieht sich auf verschiedene Mengen von Variablen, weil z. B. $x > 5$ eine Bedingung für die beiden Mengen $\{x\}$ und $\{x, y\}$ ist.) Somit könnte $[\![\varphi]\!]$ ein Element von $Val(X)$ genauso gut wie von $Val(X \cup Y)$ sein. Diese Mehrdeutigkeit wird durch die Identifizierung der korrespondierenden Belegungsmengen aufgelöst. Formal: $V \subseteq Val(X)$ und $W \subseteq Val(X \cup Y)$ werden identifiziert, gdw. $W = \{w \in Val(X \cup Y) \mid \exists v \in V : \forall x \in X : v(x) = w(x)\}$. Dies ist gerechtfertigt durch die Tatsache, daß V und W die gleiche BDD-Repräsentation besitzen.

Im folgenden werden einige Begriffe für eine Belegungsmenge $V \subseteq Val(X)$ definiert.

Definition 3.1 *Für eine Variable $x \in X$ ist die* **Existenzquantifizierung**[2] *$\exists x.V$ definiert als die Menge aller Belegungen von $X \setminus \{x\}$, die in allen Variablen außer x mit einer Belegung in V übereinstimmen. Formal: $\exists x.V = \{w \in Val(X \setminus \{x\}) \mid \exists v \in V : \forall y \in X \setminus \{x\} : v(y) = w(y)\}$.*

Definition 3.2 *Für zwei Variablen $x \in X$ und $y \notin X$ ist $V[x \leftarrow y]$ die Belegungsmenge, die durch* **Umbenennung** *von x in y entsteht. Formal: $V[x \leftarrow y] = \{w \in Val((X \setminus \{x\}) \cup \{y\}) \mid \exists v \in V : v(x) = w(y) \wedge \forall z \in X \setminus \{x\} : v(z) = w(z)\}$.*

Definition 3.3 *Für eine Variable $x \in X$ und eine Konstante $c \in Range(x)$ ist der* **Kofaktor** *$V|_{x=c}$ definiert als $\exists x.(V \cap [\![x = c]\!])$. Sei $Y = \{y_1, y_2, ..., y_n\}$ eine Teilmenge von X. Dann bezeichnet $V|_Y$ die Menge der Kofaktoren von V bezüglich der Variablen in Y. Formal: $V|_Y = \{V|_{y_1=c_1,...,y_n=c_n} \mid \forall i \in \{1, 2, ..., n\} : c_i \in Range(y_i)\}$.*

Es existieren effiziente BDD-Operationen für die Existenzquantifizierung und die Umbenennung von Variablen, sowie für Durchschnitt, Vereinigung und Vergleich von Belegungsmengen.

[2] Die prädikatenlogischen Quantoren $\exists$ und $\forall$ werden hier statt nur auf Prädikate mit analoger Bedeutung auch auf Belegungsmengen angewendet.

3.2.3 Binary Decision Diagrams – formale Definition

Für den Beweis der oberen Schranke im Abschnitt 3.3.2 und um einen exakteren Einblick in die Operationen liefern zu können, wird eine formale Definition für die BDDs benötigt. Die folgende Definition ist der von McMillan ähnlich [McM93]. Ein BDD wird mit seinem Wurzelknoten identifiziert.

Definition 3.4 *Sei $\vec{x}$ ein Vektor $(x_1, x_2, ..., x_n)$ von Booleschen Variablen. Falls $n = 0$, dann ist $\mathcal{B}$ ein* **Binary Decision Diagram** *über $\vec{x}$ gdw. $\mathcal{B}$ der 0-Terminalknoten ($\mathcal{B} = 0$) oder der 1-Terminalknoten ($\mathcal{B} = 1$) ist. Falls $n > 0$, dann ist $\mathcal{B}$ ein BDD über $\vec{x}$ gdw.*

- *$\mathcal{B}$ ein BDD über $(x_2, ..., x_n)$ ist oder*
- *$\mathcal{B} = (x_1, \mathcal{B}_0, \mathcal{B}_1)$, wobei $\mathcal{B}_0$ und $\mathcal{B}_1$ verschiedene BDDs über $(x_2, ..., x_n)$ sind. $\mathcal{B}$ wird x_1-Knoten genannt, $\mathcal{B}_0$ ist das Low-Kind von $\mathcal{B}$ und $\mathcal{B}_1$ ist das High-Kind von $\mathcal{B}$.*

Definition 3.5 *Sei $\mathcal{B}$ ein BDD über $(x_1, x_2, ..., x_n)$. Dann ist die $\mathcal{B}$ zugeordnete* **BDD-Semantik** *$[\![\mathcal{B}]\!]$ definiert als die folgende Belegungsmenge von $\{x_1, x_2, ..., x_n\}$:*

$$[\![\mathcal{B}]\!] = \begin{cases} \emptyset, & falls\ \mathcal{B} = 0 \\ Val(\{x_1, x_2, ..., x_n\}), & falls\ \mathcal{B} = 1 \\ ([\![\mathcal{B}_0]\!] \cap [\![x_i = 0]\!]) \cup ([\![\mathcal{B}_1]\!] \cap [\![x_i = 1]\!]), & falls\ \mathcal{B} = (x_i, \mathcal{B}_0, \mathcal{B}_1) \end{cases}$$

Für die Anwendung von BDDs ist es von zentraler Bedeutung, daß sie Mengen von Belegungen Boolescher Variablen kanonisch darstellen.

Satz 3.1 (Bryant, 1986) *Sei $\vec{x} = (x_1, x_2, ..., x_n)$ ein Vektor von Booleschen Variablen. Dann existiert für jede Belegungsmenge $V \subseteq Val(\{x_1, x_2, ..., x_n\})$ genau ein BDD $\mathcal{B}$ über $\vec{x}$ mit $V = [\![\mathcal{B}]\!]$ (Kanonizität).*

Beweis: Der Beweis zu dieser Kernaussage der BDD-Repräsentation von Belegungsmengen wurde 1986 von Bryant erbracht [Bry86].

□

3.2.4 Operationen

Um die Komplexität der Operationen einschätzen zu können, sind einige Erklärungen zur Implementierung eines BDD-Pakets erforderlich. Ausführlicher werden die Implementierungsdetails in einem Aufsatz von Brace et al. erklärt [BRB90]. Die Knoten aller verwendeten BDDs werden in einem Knotenvektor gespeichert. Ein Knoten hat als Attribute (unter anderen) den Variablenindex und die Indizes seines High- und Low-Kindes im Knotenvektor. Im Knotenvektor gibt es keine gleichen Knoten. Dadurch und durch die Kanonizität (Satz 3.1) ist garantiert, daß zwei die gleiche Menge darstellende BDDs gleiche Indizes im Knotenvektor haben. Erzeugt eine Operation einen Knoten für ihren Ergebnis-BDD, wird dieser nur dann neu in den Knotenvektor aufgenommen, wenn er noch nicht

Prozedur $Union$
Eingabe: BDDs $\mathcal{B}_1$, $\mathcal{B}_2$ über $(x_1, x_2, ..., x_n)$
Ausgabe: BDD über $(x_1, x_2, ..., x_n)$, der $[\![\mathcal{B}_1]\!] \cup [\![\mathcal{B}_2]\!]$ darstellt

if $\mathcal{B}_1 = 1$ **or** $\mathcal{B}_2 = 0$ **or** $\mathcal{B}_1 = \mathcal{B}_2$ **then return** $\mathcal{B}_1$
if $\mathcal{B}_2 = 1$ **or** $\mathcal{B}_1 = 0$ **then return** $\mathcal{B}_2$
Seien $\mathcal{B}_1 = (x_{i_1}, \mathcal{B}_{1,0}, \mathcal{B}_{1,1})$, $\mathcal{B}_2 = (x_{i_2}, \mathcal{B}_{2,0}, \mathcal{B}_{2,1})$
if $i_1 = i_2$ **then**
  $\mathcal{B}_{r,0} := Union(\mathcal{B}_{1,0}, \mathcal{B}_{2,0})$; $\mathcal{B}_{r,1} := Union(\mathcal{B}_{1,1}, \mathcal{B}_{2,1})$
  if $\mathcal{B}_{r,0} = \mathcal{B}_{r,1}$ **then return** $\mathcal{B}_{r,0}$ **else return** $(x_{i_1}, \mathcal{B}_{r,0}, \mathcal{B}_{r,1})$
if $i_1 < i_2$ **then**
  $\mathcal{B}_{r,0} := Union(\mathcal{B}_{1,0}, \mathcal{B}_2)$; $\mathcal{B}_{r,1} := Union(\mathcal{B}_{1,1}, \mathcal{B}_2)$
  if $\mathcal{B}_{r,0} = \mathcal{B}_{r,1}$ **then return** $\mathcal{B}_{r,0}$ **else return** $(x_{i_1}, \mathcal{B}_{r,0}, \mathcal{B}_{r,1})$
if $i_1 > i_2$ **then**
  $\mathcal{B}_{r,0} := Union(\mathcal{B}_1, \mathcal{B}_{2,0})$; $\mathcal{B}_{r,1} := Union(\mathcal{B}_1, \mathcal{B}_{2,1})$
  if $\mathcal{B}_{r,0} = \mathcal{B}_{r,1}$ **then return** $\mathcal{B}_{r,0}$ **else return** $(x_{i_2}, \mathcal{B}_{r,0}, \mathcal{B}_{r,1})$

Abbildung 3.3: Algorithmus für die Operation Vereinigung von BDDs (nach [Bry86]).

darin enthalten ist. Um dies in konstanter Zeit prüfen zu können, werden alle Knoten in eine Hash-Tabelle eingetragen. Der Index eines Knotens wird durch eine Hash-Funktion aus dem Variablenindex und den Indizes seiner Kinder berechnet.

Das Freigeben nicht mehr benötigter Knoten erfolgt durch Garbage Collection. Dazu wird eine Liste aller vom Nutzer referenzierten Knoten geführt, und bei der Garbage Collection werden alle Knoten gelöscht, die nicht von diesen aus erreichbar sind. Ein sofortiges Löschen nicht mehr benötigter Knoten ist nicht möglich, weil dadurch Einträge in den Caches der Operationsergebnisse ungültig würden.

Es folgt eine Diskussion der für die Erreichbarkeitsanalyse benötigten Operationen für BDDs über dem Vektor von Booleschen Variablen $(x_1, x_2, ..., x_n)$ mit Anmerkungen zur Komplexität der Operationen (vgl. [BRB90]).

Elementare BDDs: Die BDD-Darstellung von $[\![x_i = 0]\!]$ ist $(x_i, 1, 0)$ und die BDD-Darstellung von $[\![x_i = 1]\!]$ ist $(x_i, 0, 1)$ (für alle $i \in \{1, 2, ..., n\}$).

Vergleich: Wie oben bereits beschrieben, kann für zwei BDDs $\mathcal{B}_1$ und $\mathcal{B}_2$ durch Vergleich ihrer Indizes im Knotenvektor in konstanter Zeit festgestellt werden, ob $[\![\mathcal{B}_1]\!] = [\![\mathcal{B}_2]\!]$.

Vereinigung und Durchschnitt: Abbildung 3.3 zeigt eine rekursive Prozedur, die für zwei BDDs $\mathcal{B}_1$ und $\mathcal{B}_2$ den BDD berechnet, der $[\![\mathcal{B}_1]\!] \cup [\![\mathcal{B}_2]\!]$ darstellt. Der Algorithmus zur Berechnung der Schnittmenge ist analog. Die Prozedur wird durch Anwendung von dynamischer Programmierung verfeinert. Werden alle BDDs $\mathcal{B}_1$ und $\mathcal{B}_2$, für die die Vereinigung bereits berechnet wurde, zusammen mit dem Ergebnis in einer Hash-Tabelle gespeichert, ist die Anzahl der rekursiven Aufrufe auf $(|\mathcal{B}_1| + 2) \cdot (|\mathcal{B}_2| + 2)$ begrenzt. Folglich ist auch die Größe des entstehenden BDDs

höchstens $(|\mathcal{B}_1| + 2) \cdot (|\mathcal{B}_2| + 2)$. Dabei handelt es sich um Worst-Case-Angaben, die meist weit unterschritten werden. Praktisch hat es sich als effizienter erwiesen, statt Hash-Tabellen Caches zu benutzen, in denen beim Eintragen eines neuen Datensatzes ein eventuell an der entsprechenden Position gespeicherter alter Datensatz einfach überschrieben wird.

Bedingungen: Sei x eine diskrete Variable, die durch $x_{i_1}, x_{i_2}, ..., x_{i_{|x|}}$ ($i_1, i_2, ..., i_{|x|} \in \{1, 2, ..., n\}$) kodiert wird. Dann kann der Vergleich $[\![x = c]\!]$ ($c \in Range(x)$) als Konjunktion von Bedingungen $[\![x_{i_k} = c_k]\!]$ ($k \in \{1, 2, ..., |x|\}$, $c_k \in \{0, 1\}$) ausgedrückt werden. Die anderen Vergleiche von diskreten Variablen mit Konstanten können als Vereinigung ausgedrückt werden, zum Beispiel $[\![x \leq c]\!] = \bigcup_{d \in \{e \in Range(x) \mid e \leq c\}} [\![x = d]\!]$.

Kofaktor: Sei V eine Menge von Belegungen der Variablenmenge $\{x_1, x_2, ..., x_n\}$ und $\mathcal{B}$ der BDD mit $[\![\mathcal{B}]\!] = V$. Dann kann der BDD für den Kofaktor $V|_{x_i=c}$ ($i \in \{1, 2, ..., n\}$, $c \in \{0, 1\}$) berechnet werden, indem jede Kante zu einem x_i-Knoten zu dessen c-Kind umgelenkt wird und alle x_i-Knoten entfernt werden. Dies ist durch Traversieren von $\mathcal{B}$ möglich. Der entstehende BDD ist nicht größer als $\mathcal{B}$.

Existenzquantifizierung: Sei V eine Menge von Belegungen der Variablenmenge $\{x_1, x_2, ..., x_n\}$ und $\mathcal{B}$ ein BDD mit $[\![\mathcal{B}]\!] = V$. Dann kann $\exists x_i.V$ ($i \in \{1, 2, ..., n\}$) berechnet werden als $V|_{x_i=0} \cup V|_{x_i=1}$. Folglich ist die Knotenzahl des entstehenden BDDs höchstens $(|\mathcal{B}| + 2)^2$. Eine Existenzquantifizierung einer durch mehrere Boolesche Variablen kodierten Variable kann durch Nacheinanderausführung der Existenzquantifizierungen der einzelnen Booleschen Variablen implementiert werden.

Umbenennen von Variablen: Sei V eine Menge von Belegungen der Variablenmenge $Y = \{x_{i_1}, x_{i_2}, ..., x_{i_j}\}$ mit $1 \leq i_1 < i_2 < ... < i_j \leq n$ und $\mathcal{B}$ ein BDD mit $[\![\mathcal{B}]\!] = V$. Es soll der BDD für $V[x_{i_k} \leftarrow x_i]$ berechnet werden, wobei $\mathcal{B}$ keine Knoten mit Variablen zwischen x_{i_k} und x_i enthält. Handelt es sich bei $V[x_{i_k} \leftarrow x_i]$ zum Beispiel um die im Erreichbarkeitsalgorithmus angewendete Basisoperation zum Umbenennen der gestrichenen in ungestrichene Variablen, so muß $i < i_k$ und $i_{k-1} < i$ (bzw. $1 \leq i$, falls $k = 1$) gelten. Für diese Operation muß der BDD lediglich traversiert und jeder x_{i_k}-Knoten in einen x_i-Knoten mit den gleichen Kindern umgewandelt werden. Die Knotenzahl des entstehenden BDDs entspricht der von $\mathcal{B}$. Es ist möglich, in einem Durchlauf mehrere Umbenennungen vorzunehmen.

3.2.5 Variablenordnung

Für eine feste Variablenordnung ist die BDD-Darstellung einer Menge von Belegungen eindeutig. Die Größe der BDDs kann sich bei verschiedenen Variablenordnungen erheblich unterscheiden. Ein Beispiel dafür ist die Menge V der Belegungen v der Booleschen Variablen $\{x_1, x_2, ..., x_{2n}\}$, für die gilt $v(x_i) = v(x_{n+i})$ für alle $i \in \{1, 2, ..., n\}$. Bei der Variablenordnung $(x_1, x_2, ..., x_{2n})$ wächst die BDD-Repräsentation exponentiell in n,

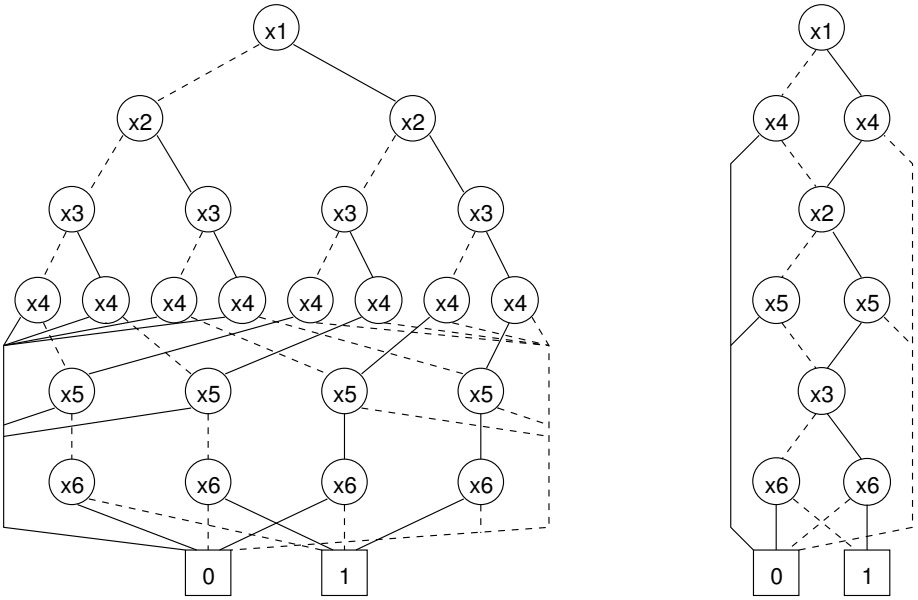


Abbildung 3.4: BDD-Darstellung einer Menge bei verschiedenen Variablenordnungen.

bei der Variablenordnung $(x_1, x_{n+1}, x_2, x_{n+2}, ..., x_n, x_{2n})$ linear. Abbildung 3.4 zeigt die BDDs für $n = 3$. Die BDD-Größen vieler praktisch auftretender Mengen reagieren sensibel auf die Variablenordnung, folglich ist die Konstruktion guter Ordnungen von großer Bedeutung.

Die Berechnung einer optimalen Variablenordnung ist NP-vollständig [BW96, THY93]. Daher ist der Zeitaufwand zur Berechnung einer optimalen Variablenordnung für den praktischen Gebrauch unakzeptabel. Es wurden jedoch mehrere Heuristiken untersucht, die sich in zwei Richtungen unterscheiden lassen:

Statisches Variablenordnen: Noch vor dem Erstellen eines BDDs werden abhängig vom Anwendungsbereich gute Variablenordnungen berechnet, die sich während der gesamten Laufzeit nicht mehr verändern. Für kommunizierende endliche Automaten können gute Variablenordnungen aus der Kommunikationsstruktur abgeleitet werden [TSL$^+$90, JPHS91, ATB94]. Im folgenden werden mit diesem Ansatz Heuristiken für den Bereich der Timed Automata entwickelt (in den Abschnitten 3.3.1 und 3.3.2, im Abschnitt 3.3.5 durch eine Beispielrechnung gerechtfertigt) und spezielle Erweiterungen für modulare CTA-Modelle angegeben. Der Vorteil des statischen Ordnens ist, daß zur Berechnungszeit der Variablenordnung alle Informationen über die Modellstruktur vorhanden sind und zur Berechnung verwendet werden können. Außerdem wird die Berechnung der statischen Variablenordnung nur einmal durchgeführt, wodurch mehr Zeit für die Berechnung der Ordnung zur Verfügung steht.

Dynamisches Variablenordnen: Während der Durchführung von Berechnungen mit den BDDs werden die Variablen nach dem Überschreiten von bestimmten Triggerwerten (z. B. BDD-Größe) anhand zur Laufzeit vorhandener Kriterien neu geord-

net [Rud93]. Diese Strategie ist vom Anwendungsbereich unabhängig und die Wirkung ist sofort an der BDD-Größe erkennbar. Vorteilhaft an dieser Strategie ist, daß alle Laufzeit-Informationen verfügbar sind und statt Schätzungen exakte Werte zur Beurteilung der Ordnung zur Anwendung kommen können. Dieser Ansatz kann jedoch keine Strukturinformationen des Modells berücksichtigen.

Wird durch statisches Variablenordnen eine gute Anfangsordnung der Variablen erzielt, so erhöht das dynamische Variablenordnen die Laufzeit der Algorithmen erheblich, obwohl keine deutliche Verminderung des Platzbedarfs erreicht wird. Ein empirischer Beleg für diese Behauptung erfolgte von Yang et al. [YBO+98]. Zur Vermeidung dieses unnötigen Mehraufwands wird bei der Implementierung der in dieser Arbeit vorgestellten Konzepte auf dynamisches Variablenordnen verzichtet und stattdessen an der Berechnung einer guten Anfangsordnung gearbeitet.

3.2.6 BDDs für Transitionsrelationen und Erreichbarkeitsmengen

In welcher Weise die Transitionen des Produktautomaten und die Menge der erreichbaren Konfigurationen in der BDD-Darstellung kodiert werden, behandelt dieser Abschnitt (siehe auch [BN01, BN00]). Verschiedene erweiterte Repräsentationen der Transitionsrelation werden in Abschnitt 3.5 diskutiert.

Für eine BDD-basierte Implementierung des in Abbildung 3.1 auf Seite 52 angegebenen Algorithmus für die Erreichbarkeitsanalyse muß die Transitionsrelation als BDD repräsentiert werden. Eine endliche Relation kann durch eine Belegungsmenge dargestellt werden, indem den Argumenten der Relation diskrete Variablen zugeordnet werden. Sei $P \subseteq P_1 \times P_2 \times ... \times P_n$ eine Relation und $X = \{x_1, x_2, ..., x_n\}$ eine Menge diskreter Variablen mit $Range(x_i) = P_i$ für alle $i \in \{1, 2, ..., n\}$. Dann sei $V_{P,(x_1,x_2,...,x_n)}$ der Name der Belegungsmenge von X mit $v \in V_{P,(x_1,x_2,...,x_n)}$ gdw. $(v(x_1), v(x_2), ..., v(x_n)) \in P$.

Die Kernoperation der Erreichbarkeitsanalyse ist die sogenannte "Image-Computation", die für ein gegebenes Transitionssystem $(Q, Q^0, \Sigma, \rightarrow)$ und eine Konfigurationsmenge $R \subseteq Q$ die Menge aller Folgekonfigurationen $R' = \{q' \in Q \mid \exists q : q \in R \wedge q \rightarrow q'\}$ berechnet (vgl. Abbildung 3.1 auf Seite 52). Für zwei Variablen q, q' mit $Range(q) = Range(q') = Q$ kann die Konfigurationsmenge R durch die Menge $V_{R,(q)}$ von Belegungen mit dem Definitionsbereich $\{q\}$ und dem Wertebereich Q repräsentiert werden; die Transitionsrelation $\rightarrow$ wird durch eine Belegungsmenge $V_{\rightarrow,(q,q')}$ von $\{q, q'\}$ repräsentiert. Dann ist $V_{R,(q)} \cap V_{\rightarrow,(q,q')}$ eine Belegungsmenge, die jedem Element von R seine Folgekonfigurationen zuordnet: $v \in V_{R,(q)} \cap V_{\rightarrow,(q,q')}$ gdw. $v(q) \in R$ und $v(q) \rightarrow v(q')$. Die Existenzquantifizierung von q führt zur gewünschten Menge von Folgekonfigurationen als Belegungsmenge von q': $v \in \exists q : (V_{R,(q)} \cap V_{\rightarrow,(q,q')})$ gdw. $v(q') \in R'$, oder in anderer Weise geschrieben, $V_{R',(q')} = \exists q : (V_{R,(q)} \cap V_{\rightarrow,(q,q')})$. Nun kann diese Menge durch die Umbenennung von q' durch q in eine Belegungsmenge von q transformiert werden: $V_{R',(q)} = (\exists q : (V_{R,(q)} \cap V_{\rightarrow,(q,q')}))[q' \leftarrow q]$. In dieser Arbeit wird die Abkürzung $\rightarrow(q, q')$ für $V_{\rightarrow,(q,q')}$ und $\xrightarrow{a}(q, q')$ für $V_{\xrightarrow{a},(q,q')}$ mit $a \in \Sigma$ verwendet.

Die verbleibende Aufgabe ist, die BDD-Repräsentation der Transitionsrelation für einen gegebenen abgeschlossenen Timed Automaton $\mathcal{A} = (L, L^0, \{x_1, ..., x_n\}, \Sigma, I, E)$ mit der ganzzahligen Semantik $[\![\mathcal{A}]\!]_I = (Q, Q^0, \Sigma \cup \mathbb{N}, \rightarrow)$ zu berechnen.

Die Transitionsrelation wird durch mehrere BDDs repräsentiert: ein BDD für die Zeit-Transitionsrelation $\xrightarrow{1}$ und je ein BDD für die diskrete Transitionsrelation $\xrightarrow{a}$ jeder Synchronisationsmarke $a \in \Sigma$. Auf diese Weise gespaltene Transitionsrelationen werden im folgenden **geteilte Transitionsrelation** genannt, die einzelnen Teil-Relationen sind **partielle Transitionsrelationen**. In diesem Kapitel ist eine Konfiguration von $\mathcal{A}$ repräsentiert durch den Zustand und die ganzzahligen Uhrenwerte. Die Variable l mit $Range(l) = L$ enthält den Zustand, und die Uhren x_i ($1 \leq i \leq n$) werden betrachtet als diskrete Variablen mit $Range(x_i) = \{0, 1, ..., C_{\mathcal{A}}(x_i) + 1\}$. Die gestrichene Version einer jeden Variablen enthält den Wert der Variablen in der Folgekonfiguration und hat den gleichen Wertebereich wie die ungestrichene Version. Die partiellen Transitionsrelationen können als Belegungsmengen der Variablen $\{l, l', x_1, x'_1, ..., x_n, x'_n\}$ repräsentiert werden. Nach dieser Notation kann die Repräsentation der Transitionsrelation in der folgenden Weise berechnet werden:

$$\begin{aligned} \xrightarrow{1}(l, l', x_1, x'_1, ..., x_n, x'_n) &= [\![l' = l]\!] \cap [\![I(l)]\!] \cap [\![I(l)]\!][x_1 \leftarrow x'_1, ..., x_n \leftarrow x'_n] \\ &\cap \bigcap_{i \in \{1,2,...,n\}} [\![x'_i = min(x_i + 1, C_{\mathcal{A}}(x_i) + 1)]\!] \end{aligned}$$

und für alle $a \in \Sigma$ gilt

$$\xrightarrow{a}(l, l', x_1, x'_1, ..., x_n, x'_n) =$$

$$\bigcup_{(m,a,\varphi,Y,m') \in E} \left([\![l = m]\!] \cap [\![l' = m']\!] \cap [\![\varphi]\!] \cap \bigcap_{i \in \{1,2,...,n\}} \begin{cases} [\![x'_i = 0]\!], & falls\ x_i \in Y \\ [\![x'_i = x_i]\!], & sonst \end{cases} \right)$$

Um die Transitionsrelation für den Produktautomaten $\mathcal{A}$ mehrerer Timed Automata $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n$ mit paarweise disjunkten Uhrenmengen zu berechnen, ist es nicht notwendig, den Produktautomaten explizit zu berechnen. Sei Σ_i die Menge der Synchronisationsmarken und $\rightarrow_i$ die Transitionsrelation von $\mathcal{A}_i$ ($i \in \{1, 2, ..., n\}$). Eine diskrete Transition von $\mathcal{A}$ mit Synchronisationsmarke a bedeutet, daß alle Komponenten, die a in ihrem Alphabet haben, eine a-Transition nehmen, wobei alle anderen Komponenten ihre Konfiguration nicht verändern. Seien q_i und q'_i entsprechend die Variablen für die Konfiguration und die Folgekonfiguration von $\mathcal{A}_i$. Dann gilt für jede Synchronisationsmarke a in der Menge der Synchronisationsmarken $\bigcup_{i \in \{1,2,...,n\}} \Sigma_i$ des Produktautomaten:

$$\xrightarrow{a}(q_1, q'_1, ..., q_n, q'_n) = \bigcap_{i \in \{1,2,...,n\}} \begin{cases} \xrightarrow{a}_i(q_i, q'_i), & falls\ a \in \Sigma_i \\ [\![q'_i = q_i]\!], & sonst \end{cases}$$

Für die Zeit-Transitionsrelation von $\mathcal{A}$ gilt:

$$\xrightarrow{1}(q_1, q'_1, ..., q_n, q'_n) = \bigcap_{i \in \{1,2,...,n\}} \xrightarrow{1}_i(q_i, q'_i).$$

Eingabe: parallele Komposition $\mathcal{A} = \{L, L^0, X, \Sigma, I, E\}$
mit der diskreten Semantik $[\![\mathcal{A}]\!]_I = (Q, Q^0, \Sigma \cup \mathbb{N}, \rightarrow)$
der abgeschlossenen Timed Automata $\mathcal{A}_i = (L_i, L_i^0, X_i, \Sigma_i, I_i, E_i)$
mit den diskreten Semantiken $[\![\mathcal{A}_i]\!]_I = (Q_i, Q_i^0, \Sigma_i \cup \mathbb{N}, \rightarrow_i), i \in \{1, ..., n\}$
und disjunkten Uhrenmengen: $X_j \cap X_k = \emptyset$ für alle $j, k \in \{1, ..., n\}$ mit $j \neq k$
Ausgabe: $Reach([\![\mathcal{A}]\!]_I)(q_1, ..., q_n)$,
mit Variable q_i entspricht der Konfiguration von $\mathcal{A}_i$ ($Range(q_i) = Q_i$)

$R := Q^0(q_1, \ldots, q_n)$
do
$R_{prev} := R$
forall $a \in (\Sigma \cup \{1\})$
$R := R \cup \big(\exists q_1 ... \exists q_n (R \cap \xrightarrow{a}(q_1, q_1', ..., q_n, q_n')) \big) [q_1' \leftarrow q_1] ... [q_n' \leftarrow q_n]$
until $R = R_{prev}$
return R

Abbildung 3.5: Berechnung der Menge erreichbarer Konfigurationen.

Zeit-Transitionen von mehr als einer Zeiteinheit können als Folge von Zeit-Transitionen einer Zeiteinheit repräsentiert werden. Eine monolithische Transitionsrelation könnte erzeugt werden durch die Vereinigung der partiellen Transitionsrelationen, aber in der Praxis ist es effizienter, diese partiellen Transitionsrelationen sequentiell anzuwenden (vgl. Abschnitt 3.5.2).

Durch die in diesem Abschnitt eingeführte BDD-Repräsentation von Mengen und Relationen kann ein BDD-basierter Algorithmus für die Erreichbarkeitsanalyse angegeben werden. In Abbildung 3.5 ist ein solcher Algorithmus für die Berechnung aller erreichbaren Konfigurationen einer parallelen Komposition von n Timed Automata aufgeführt. Die in der Abbildung eingeführten Bezeichnungen werden in den folgenden Abschnitten verwendet. Es werden ausschließlich abgeschlossene Timed Automata mit ihrer ganzzahligen Semantik (wie im Abschnitt 2.4.3 eingeführt) benutzt. Eine Überführung der Resultate auf andere Diskretisierungen ist möglich.

3.3 Effizienz durch statisches Variablenordnen

Die Anzahl der Zustände eines Produktautomaten wächst exponentiell mit der Anzahl der parallel laufenden Komponenten (bzw. Prozesse). Dieses Problem der ”Zustandsraumexplosion” erzwingt typischerweise die Anwendung von symbolischen Repräsentationstechniken. Die Technik der Repräsentation von Mengen von Zuständen als Binary Decision Diagrams ist weitverbreitet und wird für die Verifikation von finiten Automaten eingesetzt. Der nächste Schritt hin zur effizienten Verifikation ist somit die Benutzung einer endlichen Menge von Konfigurationen für die Erreichbarkeitsanalyse durch

das Anwenden einer diskreten Semantik. Die Diskretisierung erlaubt eine einheitliche Repräsentation von Mengen von Konfigurationen, die aus je einem Zustand des Automaten zusammen mit der diskretisierten (ursprünglich kontinuierlichen) Uhrenbelegung besteht. Diese Technik wurde auch von Bozga et al. in einer BDD-basierten Version von Kronos untersucht [BMPY97], jedoch konnte nur eine geringe Performance-Verbesserung gegenüber der herkömmlichen Matrix-basierten Datenstrukturen erzielt werden.

Basierend auf den in den letzten Abschnitten vorgestellten Konzepten wurde eine Werkzeugimplementierung für die BDD-basierte Erreichbarkeitsanalyse von abgeschlossenen Timed Automata geschaffen [Bey01c]. Erfahrungen mit der Analyse von finiten Automaten haben gezeigt, daß die Effizienz einer solchen Implementierung drastisch von der geeigneten Wahl mehrerer Parameter abhängt [BCL$^+$94, RAB$^+$95]. In diesem Abschnitt wird kurz skizziert, in welcher Weise diese Probleme in der entstandenen Implementierung gelöst worden sind.

Wie bereits in Abschnitt 3.2.1 erwähnt, ist die BDD-Repräsentation einer Konfigurationsmenge eindeutig bestimmt für eine gegebene Variablenordnung. Jedoch können Veränderungen der **Variablenordnung** große Auswirkungen auf die Größe des repräsentierenden BDDs haben. Der in dieser Arbeit verfolgte Ansatz zum Finden guter Variablenordnungen wird in Abschnitt 3.3.5 vorgestellt. Dabei wird eine aus der Kommunikationsstruktur des Systems resultierende Variablenordnung verwendet. Es wird eine obere Schranke für die Repräsentationsgröße der Transitionsrelation formal bewiesen. Nach dieser Schranke wächst die Größe der Transitionsrelation z. B. bei dem Protokoll für gegenseitigen Ausschluß von Fischer polynomiell mit der Anzahl der Prozesse. Aus der oberen Schranke für die Transitionsrelation wird eine Größenschätzung für die Repräsentation der Menge der erreichbaren Konfigurationen abgeleitet. Empirische Studien belegen, daß diese Schätzfunktion für gute Variablenordnungen kleine Werte liefert. Deshalb kann diese Schätzung als qualitative Bewertung verschiedener Variablenordungen verwendet werden.

3.3.1 Kommunikationsgraph und Variablenordnung

Aziz et al. bewiesen eine obere Schranke für die Größe des BDDs von Transitionsrelationen kommunizierender endlicher Automaten [ATB94]. Auf der Basis dieser oberen Schranke werden dort gute Variablenordnungen für die Menge erreichbarer Zustände festgelegt. In diesem Abschnitt werden die Resultate dieser Arbeit benutzt, um die Charakteristiken guter Variablenordnungen für Timed Automata zu erklären.

Die Aufgabe besteht darin, eine Variablenordnung für eine gegebene parallele Komposition von Timed Automata $\mathcal{A}$ zu finden, so daß die Anzahl der Knoten der BDD-Repräsentation von $Reach([\![\mathcal{A}]\!]_I)$ so klein wie möglich ist. Für diesen Zweck wird die Kommunikation zwischen den Komponenten untersucht. Zwei Komponenten $\mathcal{A}_j$ und $\mathcal{A}_k$ mit $j, k \in \{1, ..., n\}$ **kommunizieren** miteinander, kurz $\mathcal{A}_j \rightleftharpoons \mathcal{A}_k$, gdw. $\Sigma_j \cap \Sigma_k \neq \emptyset$ und $j \neq k$. Werden die Komponenten als Knoten betrachtet und die Kommunikationsrelation $\rightleftharpoons$ als Kantenmenge, entsteht der **Kommunikationsgraph**.

Zwei generelle Eigenschaften guter Variablenordungen sind:

1. Kommunizierende Komponenten haben nahe beieinander liegende Positionen innerhalb der Variablenordnung.

2. Komponenten, die mit vielen anderen Komponenten kommunizieren, haben Positionen am Anfang der Variablenordnung.

An zwei Beispielen werden diese Eigenschaften illustriert: Betrachtet werden drei endliche Automaten (d. h. Timed Automata ohne Uhren) $\mathcal{A}_1$, $\mathcal{A}_2$ und $\mathcal{A}_3$, mit den Zuständen l_1, l_2 und l_3.

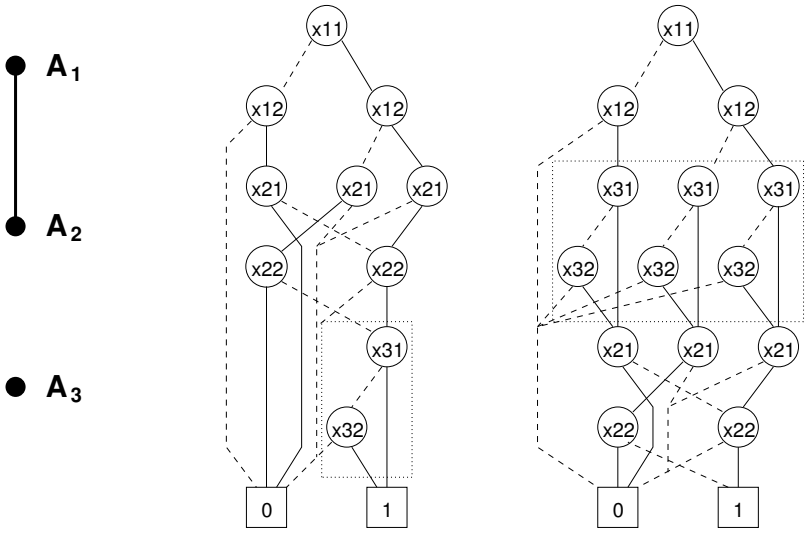


Abbildung 3.6: Kommunikationsgraph und Variablenordnung, Beispiel 1.

Im ersten Beispiel kommuniziert $\mathcal{A}_3$ weder mit $\mathcal{A}_1$ noch mit $\mathcal{A}_2$. $\mathcal{A}_1$ und $\mathcal{A}_2$ stellen über die Kommunikation sicher, daß sie sich jederzeit im gleichen Zustand befinden. Seien x_{i1} und x_{i2} die Variablen, die den Zustand von $\mathcal{A}_i$ kodieren. Abbildung 3.6 zeigt den Kommunikationsgraphen und die BDDs der erreichbaren Zustände für die Variablenordnung $(x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32})$ auf der linken Seite und für die Variablenordnung $(x_{11}, x_{12}, x_{31}, x_{32}, x_{21}, x_{22})$ auf der rechten Seite. Dieses Beispiel verdeutlicht, daß das Berücksichtigen der ersten Eigenschaft zu einer besseren Variablenordnung führt.

Im zweiten Beispiel kommuniziert $\mathcal{A}_1$ mit $\mathcal{A}_2$ und $\mathcal{A}_3$, und $\mathcal{A}_2$ kommuniziert nicht mit $\mathcal{A}_3$. Über die Kommunikation ist sichergestellt, daß $\mathcal{A}_1$ und $\mathcal{A}_2$ sowie $\mathcal{A}_1$ und $\mathcal{A}_3$ sich jeweils in verschiedenen Zuständen befinden. Abbildung 3.7 auf der nächsten Seite zeigt wiederum den Kommunikationsgraphen und die BDDs der erreichbaren Konfigurationen für die Variablenordnung $(x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32})$ und für die Variablenordnung $(x_{21}, x_{22}, x_{31}, x_{32}, x_{11}, x_{12})$. Die beiden Variablenordnungen unterscheiden sich nicht bezüglich der ersten Eigenschaft, jedoch sind die Größen ihrer BDDs verschieden. Dies zeigt, daß von diesen zwei Variablenordnungen diejenige die bessere ist, die die zweite Eigenschaft berücksichtigt.

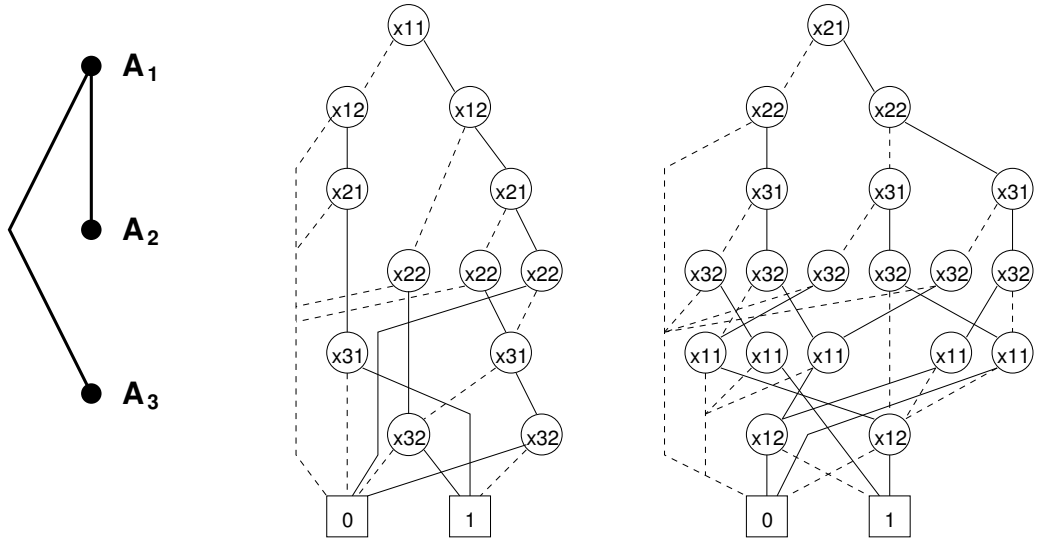


Abbildung 3.7: Kommunikationsgraph und Variablenordnung, Beispiel 2.

Um einen Algorithmus für das Finden guter Variablenordnungen zu entwickeln, wird zunächst eine obere Schranke für die BDD-Größe der Transitionsrelation eines Produktautomaten angegeben. Der einzige Engpaß im hier benutzten Erreichbarkeitsalgorithmus ist jedoch die Größe des BDDs für die Menge der erreichbaren Konfigurationen, weil die monolithische Transitionsrelation nicht berechnet wird (stattdessen werden die partiellen Transitionsrelationen angewendet, die von kleinen BDDs repräsentiert werden, siehe Abschnitt 3.5.2.1). Die obere Schranke für die Transitionsrelation wird benötigt, weil gute Variablenordnungen für die Transitionsrelation oft auch gut für die Menge der erreichbaren Konfigurationen sind und deshalb die Schätzung für die Größe der Menge erreichbarer Konfigurationen von der oberen Schranke für die Größe der Transitionsrelation abgeleitet wird. Ein Algorithmus für die Berechnung einer guten Variablenordnung sucht somit nach einer Variablenordnung, für die die Größenschätzung einen kleinen Wert ergibt. Dieser Ansatz wurde von Aziz et al. auf kommunizierende finite Automaten angewendet [ATB94].

Diese Argumentation ist nicht nur durch die eigenen empirischen Untersuchungen gerechtfertigt (siehe z. B. Abschnitt 5.4.2), sondern auch durch die Ergebnisse anderer Forschungsgruppen. Die erste Annahme, daß eine gute Korrelation zwischen der von der oberen Schranke vorhergesagten Größe und der tatsächlichen Größe der (monolithischen) Transitionsrelation existiert, wird durch die Ergebnisse von [ATB94] belegt. Die zweite Annahme ist, daß sich eine gute Variablenordnung für die Transitionsrelation günstig auf die Menge der erreichbaren Konfigurationen auswirkt. Experimente in [YBO+98] belegen dies für Mengen von Zuständen kommunizierender finiter Automaten. In dieser Arbeit (und bezogen auf finite Automaten auch in [ATB94]) wird durch empirische Untersuchungen demonstriert, daß mit dieser Strategie gute Variablenordnungen für die Menge der erreichbaren Konfigurationen von Timed Automata gefunden werden. Jedoch existieren

auch Gegenbeispiele mit linearem Wachstum des BDDs für die Transitionsrelation, aber exponentiellem Wachstum des BDDs für die erreichbaren Konfigurationen [McM93].

Um gute Variablenordnungen zu finden, ist es nicht notwendig, daß die geschätzte Größe absolut nahe an der tatsächlichen Größe liegt. Die Schätzung soll lediglich die Relation zwischen verschiedenen Variablenordnungen widerspiegeln, d. h. für gute Variablenordnungen sollten die Schätzungen der Größe kleinere Werte liefern als für schlechte Variablenordnungen. Abgesehen davon haben die obere Schranke für die BDD-Größe, die im nächsten Abschnitt definiert wird, und die auf der daraus abgeleiteten Größenschätzung basierenden Algorithmen einen weiteren Vorteil: Sie verhalten sich entsprechend der beiden in diesem Abschnitt dargelegten Eigenschaften guter Variablenordnungen.

3.3.2 Obere Schranke für die Transitionsrelation

Eine gute Variablenordnung wird durch die Abhängigkeiten der Variablen untereinander festgelegt. In den folgenden Abschnitten wird eine Heuristik zum Anordnen der Variablen vorgeschlagen, die auf einer BDD-Größenschätzung beruht [Bey01b]. Dabei wird die Kommunikationsstruktur (und damit auch die Modulstruktur) des Modells berücksichtigt. Es werden diejenigen Variablen innerhalb der Ordnung zusammengezogen, die viel miteinander kommunizieren, die also kohäsiv zueinander sind. Dazu wird eine obere Schranke für die Anzahl der Knoten des die Transitionsrelation repräsentierenden BDDs angegeben und bewiesen.

Im folgenden werden nur solche BDDs betrachtet, die durch die Anwendung der Isomorphieregel erzeugt wurden: Verschmelzen aller isomorphen Teilbäume. Die Eliminationsregel wird nicht angewendet[3]. Die Anzahl der Knoten in einem solchen BDD ist eine obere Schranke für die Anzahl der Knoten nach der Anwendung der Eliminationsregel. Die Isomorphieregel hat mehr Einfluß auf die Reduktion des BDDs, und die Zahl der durch sie eliminierten Knoten reagiert sensibler auf die Variablenordnung als dies bei der Eliminationsregel der Fall ist.

Satz 3.2 *Sei $\mathcal{B}$ ein BDD über den Vektor $(x_1, x_2, ..., x_k)$ von Booleschen Variablen und $i \in \{1, ..., k-1\}$. Dann ist die Anzahl der x_{i+1}-Knoten in $\mathcal{B}$ kleiner oder gleich der doppelten Anzahl von x_i-Knoten.*

Beweis: Alle x_{i+1}-Knoten sind Kinder von x_i-Knoten. Jeder x_i-Knoten hat maximal zwei Kinder. □

Sei $\mathcal{B}$ ein BDD über $(q_1, q'_1, ..., q_n, q'_n)$, $i \in \{1, ..., n\}$ und x die Boolesche Variable, die an erster Position in der Variablenordnung der Kodierung von q_i steht. Dann bezeichnet $|\mathcal{B}|_i$ die Anzahl von x-Knoten in $\mathcal{B}$ und $|\mathcal{B}|_{n+1}$ bezeichnet die Anzahl von Terminalknoten in $\mathcal{B}$. $|\mathcal{B}|$ ist die Anzahl aller Nicht-Terminalknoten in $\mathcal{B}$. Als abkürzende Schreibweise bezeichnet $V|_i$ die Menge der Kofaktoren $V|_{\{q_1, q'_1, ..., q_{i-1}, q'_{i-1}\}}$ für $i \in \{1, ..., n+1\}$. Für eine Menge V bezeichnet $|V|$ die Anzahl der Elemente von V.

[3] Genaugenommen entsprechen diese BDDs dann nicht der Definition 3.4 auf Seite 56. Diese Regel würde jedoch die formale Betrachtung komplizierter gestalten.

Satz 3.3 *Sei $\mathcal{B}$ ein BDD über $(q_1, q_1', ..., q_n, q_n')$ und $V \subseteq Val(\{q_1, q_1', ..., q_n, q_n'\})$ eine Belegungsmenge mit $[\![\mathcal{B}]\!] = V$. Dann gilt $|\mathcal{B}|_i = \left| V|_i \right|$ für alle $i \in \{1, ..., n+1\}$.*

Informal: Sei x_1 die erste Variable in der Variablenordnung des BDDs $\mathcal{B}$. Dann entsprechen die beiden Kinder des x_1-Knotens den beiden Kofaktoren von $\mathcal{B}$ bezüglich x_1. Die x_{i+1}-Knoten eines BDDs entsprechen den Kofaktoren des BDDs bezüglich $x_1, \ldots, x_i$.

Beweis: Seien $x_1, ..., x_s$ die Booleschen Variablen, die $q_1, q_1', ..., q_n, q_n'$ kodieren, so daß $\mathcal{B}$ ein BDD über $(x_1, ..., x_s)$ ist. Für $s = 0$ ist die Behauptung trivial, im folgenden gelte $s > 0$. Sei $s' \in \{1, ..., s\}$, so daß $x_1, ..., x_{s'}$ die Variablen $q_1, q_1', ..., q_{i-1}, q_{i-1}'$ kodieren. Schließlich sei $\mathcal{B} = (x_1, \mathcal{B}_0, \mathcal{B}_1)$. Nach der Definition der BDDs sind $\mathcal{B}_0$ und $\mathcal{B}_1$ BDDs über $(x_2, ..., x_s)$. Nach der Definition der Semantik von BDDs gilt:

$$\begin{aligned}
[\![\mathcal{B}]\!] &= ([\![\mathcal{B}_0]\!] \cap [\![x_1 = 0]\!]) \cup ([\![\mathcal{B}_1]\!] \cap [\![x_1 = 1]\!]) \\
[\![\mathcal{B}]\!] \cap [\![x_1 = 0]\!] &= [\![\mathcal{B}_0]\!] \cap [\![x_1 = 0]\!] \\
\exists x_1.([\![\mathcal{B}]\!] \cap [\![x_1 = 0]\!]) &= \exists x_1.([\![\mathcal{B}_0]\!] \cap [\![x_1 = 0]\!]) \\
[\![\mathcal{B}]\!]|_{x_1=0} &= [\![\mathcal{B}_0]\!]
\end{aligned}$$

Analog gilt $[\![\mathcal{B}]\!]|_{x_1=1} = [\![\mathcal{B}_1]\!]$. Die x_2-Knoten von $\mathcal{B}$ sind also (wegen Satz 3.1 bijektiv) den Kofaktoren von $[\![\mathcal{B}]\!]$ bezüglich x_1 zugeordnet. Induktiv entsprechen die $x_{s'+1}$-Knoten von $\mathcal{B}$ (bzw. Terminalknoten von $\mathcal{B}$, falls $s = s'$) den Kofaktoren von $[\![\mathcal{B}]\!]$ bezüglich $x_1, ..., x_{s'}$, es gilt also $|\mathcal{B}|_i = \left|[\![\mathcal{B}]\!]|_{\{x_1,...,x_{s'}\}}\right|$. Wegen der Bijektivität der Kodierung der Belegungen von $\{q_1, q_1', ..., q_{i-1}, q_{i-1}'\}$ bzw. $\{q_i, q_i', ..., q_n, q_n'\}$ durch Belegungen von $\{x_1, ..., x_{s'}\}$ bzw. $\{x_{s'+1}, ..., x_s\}$ (siehe Abschnitt 3.2.2) gilt $\left|[\![\mathcal{B}]\!]|_{\{x_1,...,x_{s'}\}}\right| = \left|[\![\mathcal{B}]\!]|_{\{q_1,q_1',...,q_{i-1},q_{i-1}'\}}\right|$. □

Anmerkung: Seien $x_1, ..., x_{s'}$ die Booleschen Variablen, die die Variablen $q_1, q_1', ..., q_{i-1}, q_{i-1}'$ kodieren. Falls es eine Belegung von $\{x_1, ..., x_{s'}\}$ in $V|_i$ gibt, die nicht eine Kodierung einer Belegung von $\{q_1, q_1', ..., q_{i-1}, q_{i-1}'\}$ ist, dann muß die leere Menge als weiterer Kofaktor betrachtet werden. Dies ist der Fall, falls die Kardinalität einer Menge Q_k ($k \in \{1, ..., i-1\}$) keine Zweierpotenz ist. Dann gilt $|\mathcal{B}|_i \leq \left| V|_i \cup \{\emptyset\} \right|$. Dieser Fall kommt jedoch aufgrund der Vereinbarung aus Abschnitt 3.2.2 nicht vor.

Aus der Anzahl der Kofaktoren einer Belegungsmenge läßt sich die Größe ihrer BDD-Repräsentation folgern. Eine obere Schranke für die Anzahl der Kofaktoren der Transitionsrelation wird durch das folgende Lemma angegeben.

Die Zeit-Transitionen werden synchron für die Uhren in allen Komponenten angewendet. Bei der Berücksichtigung im Kommunikationsgraphen ergäbe dies zusätzlichen Kanten, die alle Automaten miteinander verbinden, die eine Uhr haben. Da diese Betrachtung keinen Hinweis für die Variablenordnung gibt, werden solche Kanten hier ausgeschlossen.

Im folgenden wird nur die Transitionsrelation der diskreten Transitionen $\rightarrow' = \bigcup_{a \in \Sigma} \xrightarrow{a}$ berücksichtigt. Um die Kommunikationsstruktur nutzen zu können, wird eine Funktion definiert, die die Kommunikation zwischen den Teilen des Systems reflektiert. Diese Funktion hängt von der Ordnung der Variablen ab. Die Menge $Comm_{\mathcal{A}}(i)$ enthält die Indizes aller Komponenten von $\mathcal{A}$, die einen Index kleiner als i haben und

mit einer Komponenten mit einem Index größer als oder gleich i kommunizieren: $Comm_{\mathcal{A}}(i) = \{k \mid k < i \text{ und es existiert ein } l \geq i \text{ mit } \mathcal{A}_k \rightleftharpoons \mathcal{A}_l\}$.

Lemma 3.4 *Für die Transitionsrelation* $\rightarrow'(q_1, q_1', ..., q_n, q_n')$ *und jedes* $i \in \{1, ..., n+1\}$ *gilt:*

$$\left|\rightarrow'(q_1, q_1', ..., q_n, q_n')|_i \ \cup \ \{\emptyset\}\right| \ \leq \ 4 \cdot \prod\nolimits_{k \in Comm_{\mathcal{A}}(i)} |Q_k|^2 + 4$$

Beweis: Zunächst wird ein Lemma angegeben, das bei der Berechnung der Anzahl von Kofaktoren hilfreich ist.

Lemma 3.5 *Für alle* $V, W \subseteq Val(\{q_1, q_1', ..., q_n, q_n'\})$ *und* $i \in \{1, ..., n+1\}$ *gilt*

$$\begin{aligned}(V \cap W)|_i \ &\subseteq \ \left\{ V' \cap W' \;\middle|\; V' \in V|_i, W' \in W|_i \right\} \\ (V \cup W)|_i \ &\subseteq \ \left\{ V' \cup W' \;\middle|\; V' \in V|_i, W' \in W|_i \right\}\end{aligned}$$

Beweis: Aus der Definition 3.3 von Kofaktor folgt, daß für alle $c_1, c_1' \in Q_1, ...,$ $c_{i-1}, c_{i-1}' \in Q_{i-1}$ gilt:

$$\begin{aligned}&(V \cap W)|_{q_1=c_1, q_1'=c_1', ..., q_{i-1}=c_{i-1}, q_{i-1}'=c_{i-1}'} \\ &\qquad = V|_{q_1=c_1, q_1'=c_1', ..., q_{i-1}=c_{i-1}, q_{i-1}'=c_{i-1}'} \ \cap \ W|_{q_1=c_1, q_1'=c_1', ..., q_{i-1}=c_{i-1}, q_{i-1}'=c_{i-1}'}\end{aligned}$$

Diese Gleichung gilt analog für die Vereinigung.
Daraus folgt die Behauptung[4]. (Lemma 3.5) □

Die Transitionsrelation $\rightarrow'$ wird in drei Teilmengen partitioniert. Von den Kofaktoren dieser drei Teilmengen können unter Anwendung von Lemma 3.5 die Kofaktoren von $\rightarrow'$ gefolgert werden.

Fall 1. Diskrete Transitionen, die nur die Komponenten $\mathcal{A}_1$ bis $\mathcal{A}_{i-1}$ betreffen:

$$\begin{aligned}&\textstyle\bigcup_{a \in \Sigma \setminus (\Sigma_i \cup ... \cup \Sigma_n)} \xrightarrow{a} (q_1, q_1', ..., q_n, q_n') \\ &= \textstyle\bigcup_{a \in \Sigma \setminus (\Sigma_i \cup ... \cup \Sigma_n)} \bigcap_{k \in \{1,2,...,n\}} \begin{cases} \xrightarrow{a}_k (q_k, q_k'), & \textit{falls } a \in \Sigma_k \\ [\![q_k' = q_k]\!], & \textit{sonst} \end{cases} \\ &= \textstyle\bigcup_{a \in \Sigma \setminus (\Sigma_i \cup ... \cup \Sigma_n)} \left(\begin{array}{l} \quad \bigcap_{k \in \{1,...,i-1\}} \begin{cases} \xrightarrow{a}_k (q_k, q_k'), & \textit{falls } a \in \Sigma_k \\ [\![q_k' = q_k]\!], & \textit{sonst} \end{cases} \\ \cap \ \bigcap_{k \in \{i,...,n\}} [\![q_k' = q_k]\!] \end{array} \right) \\ &= \quad \textstyle\bigcup_{a \in \Sigma \setminus (\Sigma_i \cup ... \cup \Sigma_n)} \bigcap_{k \in \{1,...,i-1\}} \begin{cases} \xrightarrow{a}_k (q_k, q_k'), & \textit{falls } a \in \Sigma_k \\ [\![q_k' = q_k]\!], & \textit{sonst} \end{cases} \\ &\quad \cap \ \textstyle\bigcap_{k \in \{i,...,n\}} [\![q_k' = q_k]\!]\end{aligned}$$

[4]Die Gleichheit der Mengen im Lemma gilt im allgemeinen nicht: Seien $V = \{v\}$ und $W = \{w\}$ mit $v \neq w$, jedoch $\{v\}|_i = \{w\}|_i$.

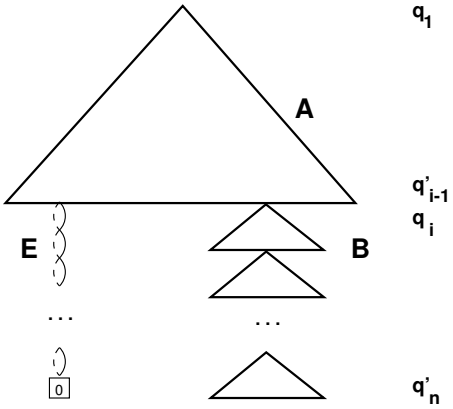


Abbildung 3.8: BDD für Fall 1.

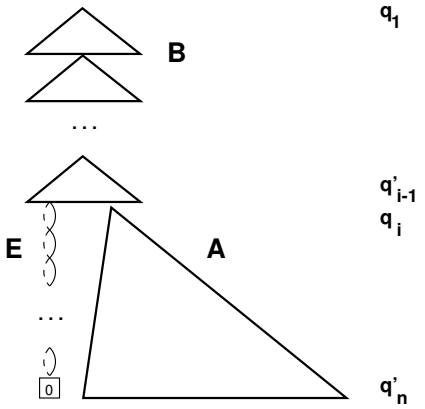


Abbildung 3.9: BDD für Fall 2.

Werden die zwei Terme des Durchschnitts mit T_1 und T_2 bezeichnet, gilt für die Kofaktoren:

$T_1|_i \subseteq \left\{\emptyset,\ Val(\{q_i, q_i', ..., q_n, q_n'\})\right\}$
$T_2|_i = \{T_2\}$

Die leere Menge muß berücksichtigt werden als Kofaktor für diejenigen Belegungen, die keiner der betrachteten diskreten Transitionen entsprechen.
Nach Lemma 3.5 folgt bzgl. der Kofaktoren:

$$\left(\bigcup_{a\in\Sigma\setminus(\Sigma_i\cup...\cup\Sigma_n)} \xrightarrow{a}\right)\Big|_i \subseteq \left\{\emptyset,\ \bigcap_{k\in\{i,...,n\}} [\![q_k = q_k']\!]\right\}.$$

Eine Skizze des BDDs für diese Kofaktoren ist in Abbildung 3.8 dargestellt. Dabei bezeichnet 'A' ein BDD für den Teil, in dem die Transitionen die Belegungen für die Variablen verändern. 'E' bezeichnet den BDD für die leere Menge und 'B' bezeichnet den BDD für die Belegungen mit $q_k = q_k'$.

Fall 2. Diskrete Transitionen, die nur die Komponenten $\mathcal{A}_i$ bis $\mathcal{A}_n$ betreffen:

$$\bigcup_{a\in\Sigma\setminus(\Sigma_1\cup...\cup\Sigma_{i-1})} \xrightarrow{a} (q_1, q_1', ..., q_n, q_n')$$

$$= \bigcap_{k\in\{1,...,i-1\}} [\![q_k' = q_k]\!]$$
$$\cap \bigcup_{a\in\Sigma\setminus(\Sigma_1\cup...\cup\Sigma_{i-1})} \bigcap_{k\in\{i,...,n\}} \begin{cases} \xrightarrow{a}_k (q_k, q_k'), & falls\ a \in \Sigma_k \\ [\![q_k' = q_k]\!], & sonst \end{cases}$$

Wird der zweite Term des Durchschnitts mit T bezeichnet, folgt nach Lemma 3.5:

$$\left(\bigcup_{a\in\Sigma\setminus(\Sigma_1\cup...\cup\Sigma_{i-1})} \xrightarrow{a}\right)\Big|_i \subseteq \{\emptyset,\ T\}.$$

Die BDD-Repräsentation ist in Abbildung 3.9 dargestellt.

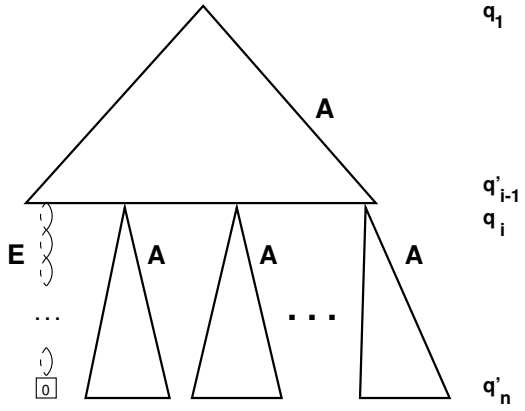


Abbildung 3.10: BDD für Fall 3.

Fall 3. Diskrete Transitionen, die sowohl Komponenten vor $\mathcal{A}_i$ als auch Komponenten nach $\mathcal{A}_i$ betreffen:

$$\bigcup_{a\in(\Sigma_1\cup\ldots\cup\Sigma_{i-1})\cap(\Sigma_i\cup\ldots\cup\Sigma_n)} \xrightarrow{a} (q_1, q_1', ..., q_n, q_n')$$

$$= \quad \bigcap_{k\in\{1,\ldots,i-1\}\setminus Comm_{\mathcal{A}}(i)} [\![q_k' = q_k]\!]$$

$$\cap \bigcup_{a\in(\Sigma_1\cup\ldots\cup\Sigma_{i-1})\cap(\Sigma_i\cup\ldots\cup\Sigma_n)} \bigcap_{k\in Comm_{\mathcal{A}}(i)\cup\{i,\ldots,n\}} \begin{cases} \xrightarrow{a}_k (q_k, q_k'), & falls\ a \in \Sigma_k \\ [\![q_k' = q_k]\!], & sonst \end{cases}$$

Sei der erste Term des Durchschnitts mit T_1 und der zweite Term mit T_2 bezeichnet, dann gilt

$T_1|_i \subseteq \{\emptyset,\ Val(\{q_i, q_i', ..., q_n, q_n'\})\}$ und

$T_2 \subseteq Val(\ \{q_k, q_k' \mid k \in Comm_{\mathcal{A}}(i)\} \cup \{q_i, q_i' ..., q_n, q_n'\}\ )$. Daraus folgt

$|\ T_2|_i\ | \leq \prod_{k\in Comm_{\mathcal{A}}(i)} |Q_k|^2$.

Nach Lemma 3.5 gilt

$$\left|\ \left(\bigcup_{a\in(\Sigma_1\cup\ldots\cup\Sigma_{i-1})\cap(\Sigma_i\cup\ldots\cup\Sigma_n)} \xrightarrow{a}\right)\Big|_i \cup \{\emptyset\}\ \right| \leq \prod_{k\in Comm_{\mathcal{A}}(i)} |Q_k|^2 + 1.$$

Die leere Menge auf der linken Seite der Ungleichung muß aus dem gleichen Grund wie bei Fall 1 mit einfließen.
Abbildung 3.10 zeigt eine Skizze dieses interessantesten Teils als BDD-Repräsentation.

Wird Lemma 3.5 auf die Vereinigung dieser drei Teilmengen der Transitionsrelation $\rightarrow'$ angewendet, ergibt sich die Behauptung. □

Von dieser oberen Schranke für die Anzahl der Kofaktoren wird eine obere Schranke für die BDD-Größe der Transitionsrelation abgeleitet. $|q_i|$ bezeichnet die Anzahl der Booleschen Variablen, die q_i kodieren.

Satz 3.6 *Sei $\mathcal{B}$ der BDD über $(q_1, q_1', ..., q_n, q_n')$ mit $[\![\mathcal{B}]\!] = \rightarrow' (q_1, q_1', ..., q_n, q_n')$. Dann gilt:*

$$|\mathcal{B}| \leq \sum_{i=1}^{n} \left(2^{2|q_i|} - 1\right) \cdot \left(4 \cdot \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k|^2 + 4\right)$$

Beweis: Lemma 3.4 zufolge gilt für jedes $i \in \{1, ..., n\}$:

$$\left| \rightarrow'(q_1, q_1', ..., q_n, q_n')|_i \cup \{\emptyset\} \right| \leq 4 \cdot \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k|^2 + 4.$$

Nach Satz 3.3 folgt für jedes $i \in \{1, ..., n\}$

$$|\mathcal{B}|_i \leq 4 \cdot \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k|^2 + 4.$$

Damit ergibt sich aus Satz 3.2 die obere Schranke für die Anzahl aller BDD-Knoten, die q_i und q_i' kodieren:

$$\begin{aligned} & \sum_{l=0}^{2|q_i|-1} 2^l \cdot \left(4 \cdot \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k|^2 + 4\right) \\ = \; & \left(2^{2|q_i|} - 1\right) \cdot \left(4 \cdot \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k|^2 + 4\right). \end{aligned}$$

Die Summe über alle $i \in \{1, ..., n\}$ führt zur Behauptung. □

Die Aussage der oberen Schranke aus Satz 3.6 über die BDD-Größe beinhaltet offensichtlich die beiden Eigenschaften guter Variablenordnungen, die in Abschnitt 3.3.1 beschrieben wurden: Wenn kommunizierende Komponenten auf benachbarten Positionen in der Variablenordnung angeordnet werden und wenn die Komponenten, die mit vielen anderen Komponenten kommunizieren, am Anfang der Variablenordnung plaziert werden, dann haben die Mengen $Comm_{\mathcal{A}}(i)$ nur wenige Elemente und die obere Schranke ist relativ klein.

Bei der oberen Schranke wird die Annahme benutzt, daß q und q' einer Komponente aufeinander folgende Positionen innerhalb der Variablenordnung haben. Diese Anordnung ist sinnvoll, weil gewöhnlich jedes Bit der Folgekonfiguration von allen Bits der aktuellen Konfiguration abhängt und somit viel "Kommunikation" innerhalb einer Komponente (also zwischen q und q') existiert. Da ein Bit der Folgekonfiguration am stärksten vom korrespondierenden Bit der aktuellen Konfiguration abhängt, wird in der Werkzeugimplementierung eine verzahnte Variablenordnung angewendet, d. h. jedem Bit der aktuellen Konfiguration folgt direkt das korrespondierende Bit der Folgekonfiguration.

Um die Erkenntnisse aus der oberen Schranke für die Transitionsrelation als Größenschätzung des BDDs für die Erreichbarkeitsmenge benutzen zu können, muß eine Anpassung des Resultats von Satz 3.6 erfolgen. Das Ziel ist eine Schätzfunktion, anhand der ein Algorithmus verschiedene Variablenordnungen automatisch vergleichen kann. Die Größenschätzung soll für eine Variablenordnung einen umso kleineren Wert liefern, je kleiner die tatsächliche BDD-Größe ist, d. h. eine relative Übereinstimmung ist ausreichend. Die Qualität der Größenschätzung beeinflußt direkt die Qualität der Variablenordnung, die vom Algorithmus als beste ausgewählt wird.

Im Unterschied zum BDD für Transitionsrelationen enthalten BDDs für Mengen erreichbarer Konfigurationen keine gestrichenen Variablen für die Folgekonfigurationen. Dies führt zur folgenden Schätzung:

$$\sum_{i=1}^{n} \left(2^{|q_i|} - 1\right) \cdot \left(4 \cdot \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k| + 4\right).$$

Sei $\mathcal{B}$ der BDD über $(q_1, ..., q_n)$ mit $[\![\mathcal{B}]\!] = Reach([\![\mathcal{A}]\!]_I)(q_1, ..., q_n)$. Für $i \in \{1, ..., n\}$ sei die Variable q_i kodiert durch die Booleschen Variablen $x_{i,1}, ..., x_{i,|q_i|}$, so daß $\mathcal{B}$ ein BDD über $(x_{1,1}, ..., x_{1,|q_1|}, ..., x_{n,1}, ..., x_{n,|q_n|})$ ist. Die Schätzung enthält die pessimistische (Worst-Case-) Annahme, daß die Anzahl der $x_{i,k+1}$-Knoten in $\mathcal{B}$ doppelt so groß ist wie die Anzahl der $x_{i,k}$-Knoten ($1 \leq k < |q_i|$). Diese Annahme ist nicht realistisch für Variablen q_i mit einer großen Anzahl von Bits und die Schätzung ist somit nicht sehr ähnlich zur tatsächlichen Größe. In diesem Fall kann zur besseren Schätzung der Anzahl der $x_{i,k+1}$-Knoten eine lineare oder exponentielle Interpolation sinnvoll sein. Für Variablen q_i mit kleiner Anzahl von Bits gibt die oben angegebene Schätzung die Relation zwischen verschiedenen Variablenordnungen approximativ wieder, und sie ist für den vorgesehenen Zweck völlig zufriedenstellend.

3.3.3 Finden guter Variablenordnungen für CTA-Modelle

Ein Algorithmus zum Finden der Variablenordnung mit der besten Größenschätzung, der die eine Komponente kodierenden Bits als Einheit betrachtet, müßte die Größenschätzung für alle Permutationen der Folge $q_1, \ldots, q_n$ berechnen. Solch ein Algorithmus wäre von enormer Zeitkomplexität und somit für die praktische Anwendung nicht relevant. Bei einer Berechnung der Größenschätzung in $O(n^2)$ würde die Gesamt-Zeitkomplexität in $O(n^2 n!)$ liegen.

Die Anwendung von dynamischer Programmierung würde diese Komplexität zu $O(n^3 2^n)$ reduzieren, indem die bereits in früheren Iterationen berechneten Ergebnisse für Teile der Ordnung gespeichert werden. Dies ist jedoch nicht effizient genug; es wird ein Algorithmus mit polynomiellem Laufzeitverhalten benötigt. Daher sind exakte Algorithmen – besonders bei großer Anzahl von Komponenten – nicht hilfreich, und es müssen heuristische Lösungen akzeptiert werden. Es kann z. B. die Arbitrary Insertion Heuristic angewendet werden [LLKS85], um die Variablenordnung bzgl. der Größenschätzung zu optimieren. Wird diese Heuristik benutzt, so liegt die Zeitkomplexität in $O(n^3)$, was ausreichend ist.

Ausnutzung der Struktur der CTA-Modelle. Cottbus Timed Automata können als Komposition von Timed Automata betrachtet werden, und somit sind die oben erwähnten Algorithmen anwendbar. Jedoch ist es für die Effizienz der Erreichbarkeitsanalyse von entscheidender Bedeutung, speziell auf CTA ausgelegte Algorithmen zu benutzen, die bei der Suche nach guten (initialen) Variablenordnungen die im Modell enthaltenen Informationen über die hierarchische Struktur ausnutzen. Die Wirksamkeit dieser Strategie wird bei der Behandlung der Fallstudie im Kapitel 6 belegt.

Die zugrunde liegende Idee dabei ist, daß die Bit-Kodierungen von allen Objekten (Variablen, Automaten), die in der gleichen Modulinstanz enthalten sind, zusammenhängende

Positionen innerhalb der Variablenordnung haben sollen. Vom Entwickler des Modells werden gerade die Objekte innerhalb eines Moduls angeordnet, die stark miteinander kommunizieren. In der Modulhierarchie wird davon ausgegangen, daß auf Synchronisationsmarken und Variablen nur dort zugegriffen werden kann, wo es auch wirklich notwendig ist. Die Schnittstelle eines Moduls soll so klein wie möglich sein. Dies führt zu einem Modell, in welchem die Kommunikation (im Sinne von Abschnitt 3.3.1) innerhalb eines Moduls stärker ist als die Kommunikation eines Moduls mit seiner Umgebung. So führt die reine Präfix-Linearisierung der Modulstruktur bereits zu einer guten Variablenordnung, die durch die schätzungsbasierte Heuristik noch weiter verbessert werden kann.

Es kann nicht garantiert werden, daß ein Algorithmus, der die hierarchische Struktur zur Variablenordnung nutzt, immer bessere Variablenordnungen berechnet als der Algorithmus für die einfache, flache Komposition von Timed Automata, aber die Nutzung der modularen Struktur hat folgende Vorteile:

- Das Wissen des Modellierers über das System wird genutzt.
- Das Problem des Variablenordnens kann in kleinere Teilprobleme aufgeteilt werden, was zu geringeren Laufzeiten oder zur Anwendbarkeit exakter Algorithmen führt. Dadurch sind bessere Variablenordnungen möglich.

3.3.4 Beispiel: Fischers Protokoll

Um die Auswirkungen der Variablenordnung auf Größenschätzung und tatsächliche Größe des BDDs der Erreichbarkeitsmenge anschaulich zu illustrieren, werden einige Ergebnisse aus dem Abschnitt 5.4.2 vorweggenommen. Als Beispiel wird das Modell für Fischers zeitbasiertes Protokoll für gegenseitigen Ausschluß [Lam87] verwendet. Das Modell besteht aus n Timed Automata. Jeder dieser Automaten modelliert einen Prozeß i, wie in Abbildung 3.11 auf der nächsten Seite dargestellt. Die Prozesse kommunizieren über eine gemeinsame Variable k mit dem Wertebereich $\{0, 1, ..., n\}$. Diese diskrete Variable könnte auch in einem separaten, zusätzlichen Timed Automaton modelliert werden, hier wird jedoch diese kompaktere Notation bevorzugt. Der Anfangswert $k = 0$ zeigt an, daß gerade kein Prozeß versucht, die kritische Sektion zu betreten (oder sich bereits darin befindet). Wenn k den Wert $i \neq 0$ hat, dann hat der Prozeß mit der Identifikationsnummer i das Vorrecht, die kritische Sektion zu betreten (oder dieser Prozeß befindet sich bereits in der kritischen Sektion).

Jeder Prozeß wird als Automat mit vier Zuständen modelliert. Am Anfang ist jeder Prozeß außerhalb der kritischen Sektion. Wenn kein anderer Prozeß versucht, in die kritische Sektion zu kommen ($k = 0$), dann kann ein Prozeß zum Zustand *Assign* wechseln. Dieser Zustand modelliert, daß ein Prozeß höchstens eine Zeiteinheit benötigt, um der Variablen k seine Identifikationsnummer zuzuordnen. Deshalb mißt die Uhr x_i die Zeit, die der Automat in diesem Zustand verweilt, und die Invariante erzwingt das Verlassen des Zustands (innerhalb einer Zeiteinheit). Der Zustandsübergang zum Zustand *Wait* setzt den Wert von k auf die Prozeß-Identifikationsnummer. Nach dem Verweilen von zwei Zeiteinheiten im Zustand *Wait* ist garantiert, daß sich kein anderer Prozeß im Zustand *Assign*

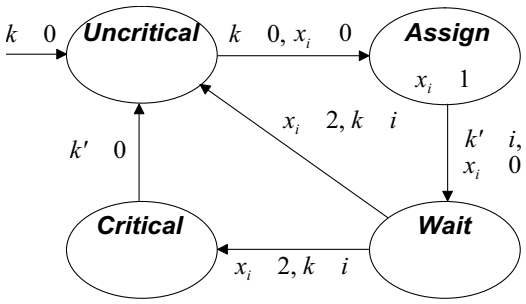


Abbildung 3.11: Prozeß i von Fischers Protokoll für gegenseitigen Ausschluß.

befindet. (Um den gegenseitigen Ausschluß zu gewährleisten, muß die minimale Aufenthaltsdauer im Zustand *Wait* größer sein als die im Zustand *Assign* angegebene Maximalzeit.) Nun ist es dem Prozeß gestattet, die kritische Sektion zu betreten, falls der Wert von k immer noch die Identifikationsnummer des betrachteten Prozesses ist; anderenfalls muß der Prozeß zurückgehen zum Zustand *Uncritical*.

3.3.5 Variablenordnen aufgrund der Größenschätzung

Wie im Abschnitt 3.3.1 gezeigt, hängt die Variablenordnung (und damit die BDD-Größe) stark von der Kommunikationsstruktur des Modells ab. Die im Abschnitt 3.3.2 entwickelte Größenschätzung bringt die beiden Grundprinzipien für das Variablenordnen in eine meßbare Form. Aus der Größenschätzung läßt sich ableiten, daß die BDD-basierte Repräsentation effizient ist für Modelle mit einem schwach verbundenen Kommunikationsgraphen, wie z. B. Bäume, Reihungen und Ringe (vgl. Abbildung 3.12). Die Modelle müssen nicht notwendigerweise eine reguläre Struktur aufweisen. Die meisten bisher analysierten Modelle, auch das der Fertigungsanlage aus Kapitel 6, entsprechen einer solchen schwach verbundenen Struktur.

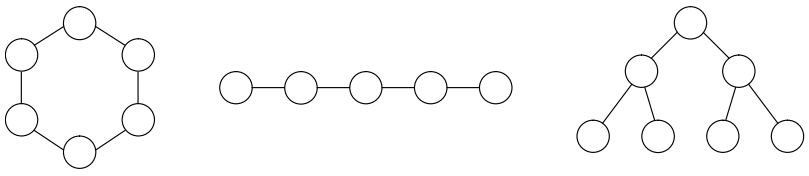


Abbildung 3.12: Schwach verbundene Modell-Strukturen, die in der Modell-Verifikation häufig auftreten: Ringe, Reihungen und Bäume.

Im folgenden wird die Wirkungsweise der Größenschätzung an einem Beispiel illustriert. Abbildung 3.13 zeigt den Kommunikationsgraphen von Fischers Protokoll für n Prozesse. An diesem Beispiel lassen sich die beiden grundsätzlichen Eigenschaften guter Variablenordnungen erklären. Experimente mit Fischers Protokoll bestätigen die Korrelation der Größenschätzung zur tatsächlichen BDD-Größe: Das Vertauschen der Position einzelner Prozesse untereinander in der Variablenordnung hat keine Wirkung auf die Schätzung der BDD-Größe; wichtig sind nur die Position der Variablen k (vgl. Ab-

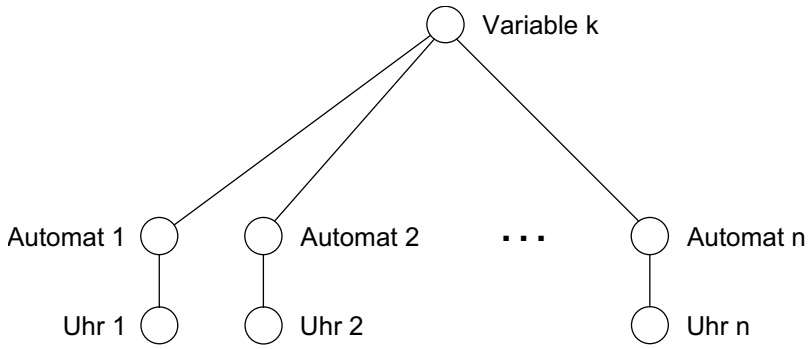


Abbildung 3.13: Kommunikationsgraph von Fischers Protokoll.

bildung 3.13) und benachbarte Positionen bei der Kodierungen der Uhr und des Zustands eines Automaten.

Die drei Einträge in Tabelle 3.1 stellen die Ergebnisse von Experimenten mit drei verschiedenen Variablenordnungen dar[5]. Zunächst werden die Anzahl der Prozesse und die Anzahl der erreichbaren Konfigurationen angegeben, um die Größe der repräsentierten Erreichbarkeitsmenge darzustellen. Die erste Zeile eines Experiments in der Tabelle enthält die Laufzeit (in Sekunden), die zweite Zeile gibt die tatsächliche Größe des BDDs der Erreichbarkeitsmenge an (in BDD-Knoten), und die dritte Zeile enthält den Wert der Größenschätzung für diesen BDD (in BDD-Knoten) entsprechend dem im Werkzeug implementierten Algorithmus.

Anzahl Prozesse	4	6	8	10	12	14	16	32	64	128
erreichb. Konfig.	1319	41979	10^6	10^7	10^9	10^{10}	10^{12}	10^{23}	10^{46}	10^{91}
Uhren separat	0.06	2.18	40.6	506						
BDD tatsächlich	828	10983	132245	1356639						
BDD geschätzt	2182	26236	295242	3188642						
Var. k am Ende	0.05	0.37	3.21	17.3	83.0	281	1750			
BDD tatsächlich	464	2127	9174	36421	145454	507821	2096957			
BDD geschätzt	1290	11658	157458	1417170	10^7	10^8	10^9			
Var. k am Anfang	0.04	0.15	0.50	1.35	1.61	3.81	6.50	61.4	559	5200
BDD tatsächlich	326	812	1497	2375	3450	4720	6190	24983	100200	401161
BDD geschätzt	616	1360	2400	3720	5328	7224	9424	37296	148336	591600

Tabelle 3.1: Effizienzvergleich verschiedener Variablenordnungen für die Verifikation des gegenseitigen Ausschlusses von **Fischers Protokoll**.

Im ersten Experiment wurde eine Variablenordnung benutzt, die die erste heuristische Regel für gute Variablenordnungen verletzt: die Variablen einer Komponenten (eines Prozesses) haben keine benachbarten Positionen. Es wurde die Variablenordnung (Variable k, Automat 1, ..., Automat n, Uhr 1, ..., Uhr n) benutzt. Dies führt zu einem schnellen Wachstum der BDD-Größe.

[5]Der maximal zur Verfügung stehende Speicher für das BDD-Paket wurde der wachsenden BDD-Knotenanzahl proportional angepaßt. Dies ist bei allen Messungen für die BDD-Repräsentation notwendig, da die kleineren Modelle sonst zu stark von dem überproportional großen Cache profitieren und die Meßwerte somit verfälscht würden. Alle Messungen wurden auf einem Linux-Rechner mit Prozessor AMD Athlon, 1 GHz Taktfrequenz und 1.2 GB Hauptspeicher durchgeführt.

Wenn die Variable k an letzter Stelle in der Variablenordnung positioniert wird, gilt $Comm_1 = Comm_{n+2} = \emptyset$ und $Comm_i = \{1, ..., i-1\}$ für $i \in \{2, ..., n+1\}$. Die Schätzung für den BDD $\mathcal{B}$ aller erreichbaren Konfigurationen kann wie folgt berechnet werden: Zuerst wird $|\mathcal{B}|_i := \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k|$ berechnet (die 4 in der ursprünglichen Gleichung kann vernachlässigt werden, da die relativen Größen interessieren). Weil $Comm_1 = Comm_{n+2} = \emptyset$ gilt, hat die Schätzung für $|\mathcal{B}|_1$ und $|\mathcal{B}|_{n+2}$ den Wert 1. Für $|\mathcal{B}|_i$ ($i \in \{2, ..., n+1\}$) beträgt die Größenschätzung 12^{i-1}, weil $|Q_k| = 12$ für $k \in \{1, ..., n\}$ (Anzahl der Konfigurationen für jeden Prozeß = 4 Zustände $\times$ 3 Uhrenwerte[6]) und $Comm_i = \{1, ..., i-1\}$ für $i \in \{2, ..., n+1\}$. Da die Variable k an der Position $n+1$ steht, ist der größte Term in der Summe der Größenschätzung derjenige für Komponente k, nämlich $\left(2^{|q_{n+1}|} - 1\right) \cdot 12^n$. Die Schätzung für $|\mathcal{B}|$ liegt damit in $O(n \cdot 12^n)$. Das zweite Experiment belegt, daß die BDD-Größe tatsächlich exponentiell wächst.

Wird die Variable k an erster Stelle innerhalb der Variablenordnung plaziert, ist $Comm_1 = Comm_{n+2} = \emptyset$ und $Comm_2 = ... = Comm_{n+1} = \{1\}$. Die Schätzung für $|\mathcal{B}|_1$ und $|\mathcal{B}|_{n+2}$ ist wiederum 1, aber die Schätzung für $|\mathcal{B}|_2, ..., |\mathcal{B}|_{n+1}$ ist $n+1$ (da $|Q_1| = n+1$). Somit liegt die Schätzung für die Größe von $\mathcal{B}$ in $O(n^2)$. Im dritten Experiment stimmt die Schätzung mit der tatsächlichen Größe gut überein. Die benötigte Laufzeit für die Verifikation verhält sich für dieses Beispiel polynomiell.

In der Abbildung 3.14 auf der nächsten Seite werden die Meßergebnisse graphisch veranschaulicht. Das obere Diagramm stellt für drei verschiedene Variablenordnungen den Speicherbedarf in Abhängigkeit von der Anzahl der beteiligten Fischer-Prozesse dar (Anzahl der zur Repräsentation der Erreichbarkeitsmenge benötigten BDD-Knoten). Das untere Diagramm gibt für die gleichen Experimente die Berechnungszeit in Abhängigkeit von der Anzahl der Prozesse wieder (in Sekunden).

Der dramatische Einfluß der Variablenordnung auf den Speicherbedarf wird durch eine zusätzliche Visualisierung (siehe Abbildung 3.15 auf Seite 78) der BDDs für zwei unterschiedliche Variablenordnungen illustriert: In der Abbildung ist die Gestalt der BDDs der Erreichbarkeitsmenge für 8 Fischer-Prozesse wiedergegeben. Die große Figur resultiert aus der Variablenordnung, bei der die Variable k unter Verletzung einer der Charakteristiken für gute Variablenordnungen am Ende eingeordnet ist (maximale BDD-Breite: 1028 Knoten). Die kleine Figur in der Mitte resultiert aus der Ordnung, bei der die Variable k am Anfang steht (maximale BDD-Breite: 106 Knoten).

Um Abweichungen zwischen der Schätzung und der tatsächlichen Größe der BDDs zu verstehen, wird folgendes betrachtet: Sei $\mathcal{B}$ der BDD über $(q_1, ..., q_n)$, der $Reach([\![\mathcal{A}]\!]_I)$ repräsentiert. Für $i \in \{1, ..., n\}$ sei die Variable q_i durch die Booleschen Variablen $x_{i,1}, ..., x_{i,|q_i|}$ kodiert, so daß $\mathcal{B}$ der BDD über $(x_{1,1}, ..., x_{1,|q_1|}, ..., x_{n,1}, ..., x_{n,|q_n|})$ ist. Die Schätzung beinhaltet die pessimistische Annahme, daß die Anzahl der $x_{i,k+1}$-Knoten in $\mathcal{B}$ doppelt so groß ist wie die Anzahl der $x_{i,k}$-Knoten ($1 \leq k < |q_i|$). Da diese Annahme

[6]Es werden 3 Uhrenwerte innerhalb des Modells benutzt, und die Werkzeugimplementierung benutzt einen Wertebereich, der durch eine Wertebereichsdefinition im Modell vorgegeben wird (anstatt $C_{\mathcal{A}}(x) + 1$ zu benutzen, was 4 Werte ergäbe). Die Meßwerte in Tabelle 3.1 auf der vorherigen Seite basieren somit auch auf 3 Uhrenwerten. Nach der Definition von $v(x) \oplus \delta$ müßten 4 Uhrenwerte berücksichtigt werden, was zu $|Q_k| = 16$ statt $|Q_k| = 12$ führt. Solange alle Vergleiche der Uhr x mit $C_{\mathcal{A}}(x)$ die Form $x \geq C_{\mathcal{A}}(x)$ haben, genügt $C_{\mathcal{A}}(x)$ als Maximalwert von x.

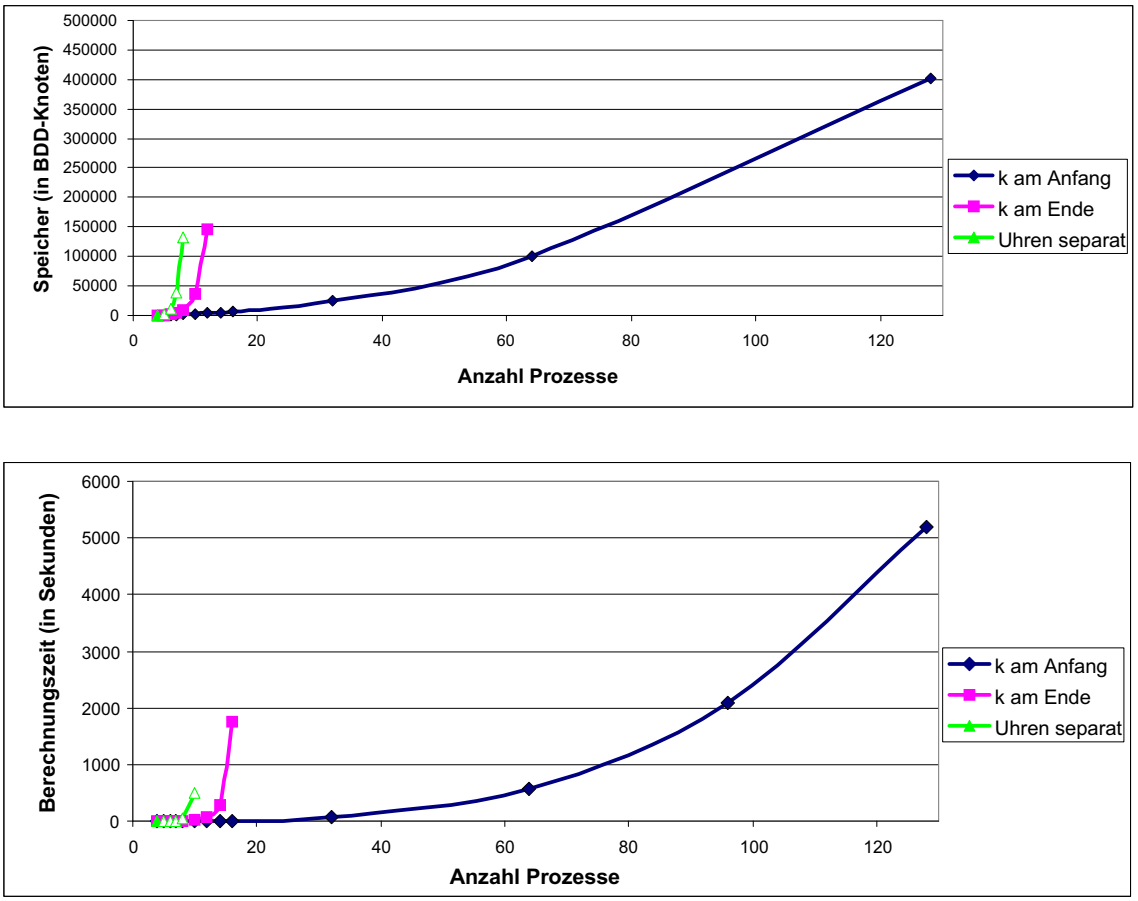


Abbildung 3.14: Speicherbedarf und Berechnungszeit für Fischers Protokoll unter Benutzung von drei verschiedenen Variablenordnungen.

für Variablen q_i mit großer Anzahl von Bits nicht realistisch ist, stimmt die Schätzung nicht gut mit der tatsächlichen Größe überein. Im dritten Experiment führt der (der Annahme zufolge sehr große) Zustandsraum der q_i nicht zu einer allzu starken Aufblähung des Gesamtzustandsraumes, da jeweils nach der Variable innerhalb des BDDs gleich eine Variable der gleichen Komponente folgt (jeweils Zustand des Automaten und Uhr zusammen), die den BDD verschmälern, d. h. die zweite Variable ”konsumiert” den Zustandsraum der ersten Variable. Um eine bessere Schätzung für die Anzahl der $x_{i,k+1}$-Knoten zu bekommen, kann eine lineare Interpolation benutzt werden. Für den gewünschten Zweck des relativen Vergleichs zweier Ordnungen ist dies jedoch nicht zwingend notwendig, da nur die ungefähre Relation zwischen zwei Variablenordungen reflektiert werden soll. Eine weitere Abweichung kann dadurch auftreten, daß in der Schätzung Begrenzungen des Wachstums nicht berücksichtigt werden, die durch die BDD-Datenstruktur gegeben sind: $2^m/m$ ist eine obere Schranke für die Größe eines BDDs über m binäre Variablen. Die Division durch m erfolgt, weil aufgrund der Reduktionsregeln des BDDs bei maximal möglichem Wachstum spätestens in den letzten $\log m$ Variablen die Reduktion der BDD-

Für das Verständnis der Funktionsweise der BDD-Repräsentation und des Einflusses der Parameter (z. B. Variablenordnung) auf den BDD ist es wichtig, die Struktur des BDDs zu kennen. Deshalb wurden bei der Diskussion der Auswirkungen verschiedener Techniken auf die BDD-Größe oft mit Papier und Bleistift BDD-Gestalten gezeichnet und mühsam überprüft, ob sich die Behauptungen bestätigen.
Um dieses Vorgehen zu vereinfachen, wurde eine automatisierte Visualisierung von BDDs eingeführt. In dieser Visualisierung repräsentiert jeder Bildpunkt eine feste Anzahl von BDD-Knoten. Alle Knoten (bzw. Bildpunkte) für ein bestimmtes Bit einer Variable werden in einer horizontalen Linie angeordnet, d. h. eine lange horizontale Linie repräsentiert eine BDD-Schicht mit vielen Knoten.
Anhand einer solchen Visualisierung sind viele Zusammenhänge der Struktur eines BDDs anschaulich erklärbar. Oft werden solche Visualisierungen zur Verdeutlichung von Unterschieden in der BDD-Struktur (z. B. durch verschiedene Variablenordnungen) mit unterschiedlichen Farben übereinandergelegt. Das oben gezeigte Bild gibt beispielsweise die Gestalten zweier BDDs der Erreichbarkeitsmengen für unterschiedliche Variablenordnungen wieder.
Mitunter wird diese Darstellungsform auch auf BDDs angewendet, die gar nicht existieren: Die aus der Schätzfunktion abgeleitete Voraussage über Struktur und Form kann ebenfalls auf diese anschauliche Weise graphisch dargestellt werden.

Abbildung 3.15: Visualisierung von BDDs.

Knoten beginnt. So kann der BDD im zweiten Experiment für die Variable k nur noch maximal $|\mathcal{B}|_{n+1} \leq 2^{n+1}$ Kofaktoren repräsentieren, da $|q_{n+1}| = n + 1$.

Zur graphischen Veranschaulichung zeigt Abbildung 3.16 eine visuelle Reflexion der vorgeschlagenen Größenschätzung (Variablenordnung: Variable k am Anfang): die Figur des virtuellen, erwarteten BDDs, wie er nach der Schätzfunktion aussehen würde (maximale BDD-Breite: 320 Knoten). Abbildung 3.17 stellt demgegenüber die Gestalt des tatsächlichen BDDs für die Erreichbarkeitsmenge dar (maximale BDD-Breite: 106 Knoten). Die Gestalt des geschätzten BDDs unterscheidet sich aus folgenden Gründen vom tatsächlichen: einerseits wegen der pessimistischen Annahme, daß jeweils vor jeder Prozeß-Komponenten der volle Zustandsraum der Variable k vorhanden ist (alle Kofaktoren der Variable k). Dies wird in der Graphik dadurch deutlich, daß die Breite des geschätzten BDDs zwischen den Prozeß-Komponenten konstant bleibt, obwohl in Wirklichkeit der Teil dieses Zustandsraumes, den die nachfolgenden Komponenten nicht

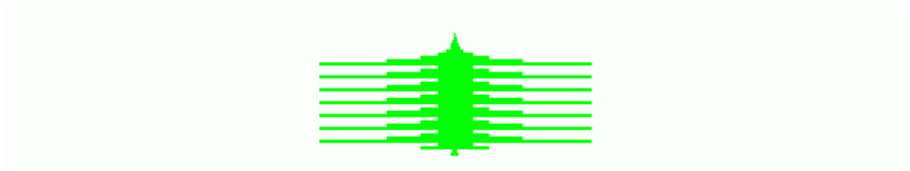


Abbildung 3.16: Visualisierung der Gestalt des geschätzten, real nicht vorhandenen BDDs für die Erreichbarkeitsmenge (Fischer8).

Abbildung 3.17: Gestalt des tatsächlichen BDDs für die Erreichbarkeitsmenge (Fischer8).

benötigen, ”konsumiert” wird. Andererseits wird von der Schätzung angenommen, daß der Platzbedarf für eine Komponente selbst nach der Funktion 2^n von der Anzahl der Bits zu ihrer Speicherung abhängt, was auch nicht der Realität entspricht (siehe oben). Diese beiden Parameter ändern jedoch nichts an der relativen Bewertung zweier Variablenordnungen.[7]

Einer der wichtigsten Vorteile der hier vorgestellten Strategie ist, daß der diskrete Teil des Zustandsraums (der Kontrollzustände) mit dem ursprünglich kontinuierlichen Teil des Zustandsraums (der Uhren) vereinheitlicht wurde und daß somit die Variablen innerhalb des BDDs in beliebiger Reihenfolge auftreten können. Dies erlaubt das Anwenden aller möglichen Variablenordnungen. Im Unterschied zu den meisten bestehenden BDD-Anwendungen wird hier eine statische, auf der Größenschätzung beruhende Methode zum Variablenordnen bevorzugt, anstatt ein dynamisches Variablenordnen einzusetzen, das einen großen Teil der Laufzeit für sich benötigt. Darüberhinaus hat das statische Variablenordnen den Vorteil, daß die Strukturinformationen des Modells noch verfügbar sind und ausgenutzt werden können.

3.4 Verfeinerungsanalyse – Dritter Schritt zur Bewältigung großer Systeme

Viele reale Software-Systeme sind sehr groß, so daß die Erreichbarkeitsanalyse selbst bei Ausnutzung der bekannten symbolischen Techniken wegen der hohen Speicher- und Zeitkomplexität nicht möglich ist. Dadurch wird die Anwendbarkeit der ausschließlich auf Erreichbarkeitsanalyse basierenden Verifikation stark eingeschränkt und es kann hilf-

[7] Ein weiterer Parameter, der jedoch von der Schätzung nicht berücksichtigt werden kann, ist die zeitbasierte Abhängigkeit der Komponenten untereinander. Diese Abhängigkeit führt zu dem breiteren inneren (vollständig eingefärbten) Bereich des tatsächlichen BDDs. Ausführungen dazu folgen im Abschnitt zur On-the-fly-Analyse und zur Ordnung der Transitionen.

reich sein, bei der Analyse modular vorzugehen. Es wird somit notwendig, auch eine werkzeugunterstützte Verfeinerungsanalyse als Voraussetzung für das modulare Beweisen bereitzustellen. Für die BDD-Repräsentation wurde ein Algorithmus zur Prüfung auf Existenz einer Simulationsrelation eingesetzt, um die Möglichkeiten der Verfeinerungsanalyse auf Cottbus Timed Automata anwenden zu können [Bey01a]. Deren effiziente Realisierung durch Verwendung von Erreichbarkeitsalgorithmen wird im folgenden vorgestellt (vgl. Lösung zu Forderung 3 auf Seite 17). Die praktische Relevanz wird an zwei Beispielen verdeutlicht.

Die für die Definition einer modularen Kompositionsstruktur notwendigen Konzepte wurden in Abschnitt 2.3 (bzw. [BR01]) beschrieben. Es soll der Nachweis ermöglicht werden, ob ein bestimmtes CTA-Modul (detailliert, Implementierung) ein anderes CTA-Modul (abstrakt, Schnittstelle) implementiert. Implementationsbeziehungen für einige Formalismen, z. B. Hybrid Modules, wurden in der Literatur zwar vorgestellt, aber noch nicht effizient implementiert, so daß der Ansatz der Verfeinerungsanalyse im Bereich Model-Checking bisher weitgehend unbeachtet geblieben ist.

3.4.1 Abstraktion und Verfeinerung

Die CTA-Notation unterstützt das Entwickeln eines strukturierten Modells unter Benutzung einer kompositionellen (Enthaltenseins-) Hierarchie. Mehrere verschiedene Ebenen einer solchen Hierarchie können benutzt werden, um verschiedene Abstraktionsebenen des System-Modells auszudrücken. Ziel des Modulkonzeptes ist, nicht nur eine vorteilhafte Modellierungstechnik, sondern auch eine Technik zur modularen Verifikation für größere Modelle anzubieten.

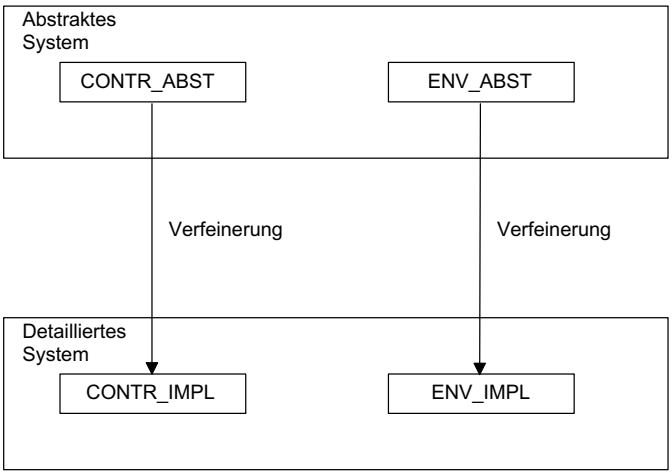


Abbildung 3.18: Modellierung durch Verfeinerung.

In jedem Verfeinerungsschritt eines Modellierungsprozesses, der nach dem Top-Down-Vorgehen arbeitet, wird ein abstraktes Modul ersetzt durch ein spezielleres Modul

mit einer tieferen inneren Hierarchie oder einem spezielleren Verhalten. In der Abbildung 3.18 besteht das Modell eines eingebettetes Systems aus einem Umgebungsmodell und einem Modell der Steuerung. Dort gibt es zwei verschiedene Versionen jeder Komponente: ein abstraktes Modul und eine detailliertes Modul.

Im Beispiel von Abbildung 3.18 ist zur Demonstration der modularen Beweistechnik ein detailliertes System-Modell gegeben, welches aus den beiden Komponenten CONTROLLER_IMPLEMENTATION und ENVIRONMENT_IMPLEMENTATION (als CONTR_IMPL und ENV_IMPL bezeichnet) besteht, und die Sicherheitseigenschaft P soll bewiesen werden. Falls dieses Gesamtsystem für die automatische Analyse zu komplex ist, ist es evtl. möglich zu beweisen, daß die Eigenschaft für das System CONTR_IMPL || ENV_ABST gilt, wobei ENV_ABST ein abstrakteres Modell der Umgebung als ENV_IMPL ist. Für den modularen Beweis kann benutzt werden, daß die Sicherheitseigenschaft P für das System CONTR_IMPL || ENV_IMPL gültig ist, wenn die folgenden zwei Eigenschaften bewiesen werden können:

- Das System CONTR_IMPL || ENV_ABST hat die Sicherheitseigenschaft P.
- ENV_IMPL ist eine Verfeinerung von (implementiert) ENV_ABST.

Wenn die Abstraktion günstig gewählt wurde, ist zu erwarten, daß diese beiden Schritte wesentlich einfacher zu berechnen sind als der Beweis in einem Schritt. Für den ersten Schritt wird die Erreichbarkeitsanalyse benutzt, für den zweiten Schritt die im folgenden beschriebene Methode.

Die Intuition hinter dem hier behandelten Verfeinerungskonzept ist ein Annahme-Garantie-Paradigma (für das allgemeine Prinzip siehe [Lam83a] und [dR98]). Dabei wird CONTR_IMPL als Beschreibung aller Eigenschaften, die das betrachtete System (Steuerung) der Umgebung garantiert, aufgefaßt. ENV_IMPL ist die Beschreibung aller Eigenschaften, von denen angenommen wird, daß die Umgebung des Systems sie erfüllt. Eine vereinfachte Verifikation kann dann vollzogen werden, wenn eine Abstraktion der Umgebung benutzt wird, die nicht mehr Annahmen beinhaltet, oder aber, wenn eine Abstraktion des Systems benutzt wird, die nicht mehr garantiert. Unter Beachtung des in dieser Arbeit benutzten Formalismus wird dieses Prinzip wie folgt beschrieben:

Definition 3.6 *Seien $\mathcal{P}$ und $\mathcal{Q}$ zwei CTA-Module. Dann besteht die* **Verfeinerungsrelation** *$\mathcal{P}$* **refines** *$\mathcal{Q}$, wenn die folgenden Bedingungen erfüllt sind:*

1. $\mathcal{Q}.par_{\Sigma}^{-1}(\mathtt{I}) \subseteq \mathcal{P}.par_{\Sigma}^{-1}(\mathtt{I})$
2. $\mathcal{P}.par_{\Sigma}^{-1}(\mathtt{O}) \subseteq \mathcal{Q}.par_{\Sigma}^{-1}(\mathtt{O})$
3. $\mathcal{Q}.\Sigma \setminus \mathcal{Q}.par_{\Sigma}^{-1}(\mathtt{L}) = \mathcal{P}.\Sigma \setminus \mathcal{P}.par_{\Sigma}^{-1}(\mathtt{L})$
4. $[\![[\![\mathcal{P}]\!]_I]\!]_{\mathcal{L}} \subseteq [\![[\![\mathcal{Q}]\!]_I]\!]_{\mathcal{L}}$

Anmerkung: Die Bedingungen in der Definitionen können wie folgt interpretiert werden:

1. Wenn ein Synchronisationssignal σ in einem Modul $\mathcal{M}$ als Eingabe-Signal deklariert ist, dann garantiert $\mathcal{M}$, σ nicht einzuschränken. Deshalb ist jedes Eingabe-Signal der Spezifikation ein Eingabe-Signal der Implementation.

2. Beim Auftreten eines Signals σ als Ausgabe-Signal in einem Modul $\mathcal{M}$ nimmt $\mathcal{M}$ an, daß σ von der Umgebung nicht eingeschränkt wird. Die Implementierung darf nicht mehr Annahmen machen als die Spezifikation, also ist jedes Ausgabe-Signal der Implementation auch ein Ausgabe-Signal der Spezifikation.

3. Die Signale eines Moduls $\mathcal{M}$ können eingeteilt werden in eine Menge von Schnittstellensignalen ($\mathcal{M}.par_{\Sigma}^{-1}(\mathtt{I}) \cup \mathcal{M}.par_{\Sigma}^{-1}(\mathtt{O}) \cup \mathcal{M}.par_{\Sigma}^{-1}(\mathtt{M})$) und eine Menge von lokalen Signalen ($\mathcal{M}.par_{\Sigma}^{-1}(\mathtt{L})$). Schnittstellensignale sind diejenigen Signale, über die $\mathcal{M}$ mit der Umgebung kommunizieren kann.

4. $[\![\mathcal{M}]\!]_I = [\![\mathcal{M}.\mathcal{A}]\!]_I = (Q, Q^0, \mathcal{M}.\Sigma \cup \mathbb{N}, \rightarrow)$ ist das dem Modul $\mathcal{M}$ zugehörige markierte Transitionssystem (ganzzahlige Semantik). Die Menge der Spuren $[\![[\![\mathcal{P}]\!]_I]\!]_{\mathcal{L}}$ des von $\mathcal{P}$ erzeugten Transitionssystems ist eine Untermenge der Menge der Spuren $[\![[\![\mathcal{Q}]\!]_I]\!]_{\mathcal{L}}$ des von $\mathcal{Q}$ erzeugten Transitionssystems[8]. Das Auftreten einer Spur t in $[\![[\![\mathcal{Q}]\!]_I]\!]_{\mathcal{L}}$ bedeutet, daß $\mathcal{Q}$ das Systemverhalten t erlaubt, die Verfeinerung darf nicht mehr Verhaltensweisen erlauben als die Spezifikation.

3.4.2 Simulationsrelation

Ein Modul $\mathcal{P}$ verfeinert ein Modul $\mathcal{Q}$, wenn $\mathcal{Q}$ für jeden Schritt von $\mathcal{P}$ einen Schritt mit der selben Marke ausführen kann (Sprach-Inklusion). Für die algorithmische Analyse der Verfeinerung in der Werkzeugimplementierung werden die beiden Modelle auf das Vorhandensein einer Simulationsrelation hin geprüft. Simulation ist ein hinreichendes Kriterium für Sprach-Inklusion [DHWT92]. Sprach-Inklusion (Language inclusion) für Timed Automata ist unentscheidbar [AD94]. Obwohl dies für abgeschlossene Timed Automata entscheidbar ist, ist die Sprach-Inklusion von extrem hoher Zeitkomplexität. Deshalb wird bei den werkzeugunterstützten Analysen die Existenz einer Simulationsrelation überprüft. Dieses Vorgehen ist auch durch die Annahme gerechtfertigt, daß zwei Module, zwischen denen die Verfeinerungsbeziehung besteht, eine ähnliche Struktur aufweisen. Diese Annahme ist insbesondere dann erfüllt, wenn die Module im Entwicklungsprozeß durch schrittweise Verfeinerung entstanden sind.

Bei der Definition der (Timed) Simulation wird der Begriff der *Spur* aus Abschnitt 2.4.4 und damit verbunden die *Hülle* bezüglich der internen Transitionen $\rightarrowtail$ angewendet. In einer Spur kommen keine (lokalen und außerhalb des Moduls nicht sichtbaren) Synchronisationsmarken der internen diskreten Transitionen vor. Nach der Definition der Simulation folgt die Beschreibung des Algorithmus für den Nachweis der Simulationsrelation. Dabei wurde das Konzept der *sicheren Simulationsrelation* verwendet (vgl. [DHWT92]).

Timed Simulation für CTA-Module wird wie folgt definiert [Bey01a]:

[8] Für die Definition der Spur-Semantik siehe Abschnitt 2.4.4 auf Seite 48.

Definition 3.7 *Seien $\mathcal{P}$ und $\mathcal{Q}$ zwei CTA-Module mit $[\![\mathcal{P}]\!]_I = (S_\mathcal{P}, S^0_\mathcal{P}, \Sigma_\mathcal{P}, \rightarrow_\mathcal{P})$, $[\![\mathcal{Q}]\!]_I = (S_\mathcal{Q}, S^0_\mathcal{Q}, \Sigma_\mathcal{Q}, \rightarrow_\mathcal{Q})$, $\mathcal{P}.X \cap \mathcal{Q}.X = \emptyset$ und $(\mathcal{P}.\Sigma \setminus \mathcal{P}.par^{-1}_\Sigma(\mathtt{L})) \cup \mathbb{N} = (\mathcal{Q}.\Sigma \setminus \mathcal{Q}.par^{-1}_\Sigma(\mathtt{L})) \cup \mathbb{N}$. Sei $\Sigma = (\mathcal{P}.\Sigma \setminus \mathcal{P}.par^{-1}_\Sigma(\mathtt{L})) \cup \mathbb{N}$. Das Modul $\mathcal{Q}$* **simuliert** *das Modul $\mathcal{P}$ gdw.*

- *eine Relation $\mathcal{R} \subseteq S_\mathcal{P} \times S_\mathcal{Q}$ (genannt Simulationsrelation) existiert mit*
  $\forall \sigma \in \Sigma, \forall (p,q) \in \mathcal{R}, \forall p' \in S_\mathcal{P} : \left(p \overset{\sigma}{\rightarrowtail}_\mathcal{P} p'\right) \Longrightarrow \left(\exists q' : q \overset{\sigma}{\rightarrowtail}_\mathcal{Q} q' \wedge (p',q') \in \mathcal{R}\right)$,
- *alle Initialkonfigurationen von $[\![\mathcal{P}]\!]_I$ in der Simulationsrelation enthalten sind:*
  $S^0_\mathcal{P} \subseteq \{p \in S_\mathcal{P} \mid \exists q : (p,q) \in \mathcal{R}\}$.

Anmerkung: Die Überprüfung der Simulationsrelation betrachtet nur Marken entsprechend der Spuren. Die Transitionsrelation $\rightarrowtail$ wurde in Definition 2.21 auf Seite 48 definiert.

Eingabe: markiertes Transitionssystem $[\![\mathcal{P}]\!]_I = (S_\mathcal{P}, S^0_\mathcal{P}, \Sigma_\mathcal{P}, \rightarrow_\mathcal{P})$
mit der zugehörigen Transitionsrelation $\rightarrowtail_\mathcal{P}$,
markiertes Transitionssystem $[\![\mathcal{Q}]\!]_I = (S_\mathcal{Q}, S^0_\mathcal{Q}, \Sigma_\mathcal{Q}, \rightarrow_\mathcal{Q})$
mit der zugehörigen Transitionsrelation $\rightarrowtail_\mathcal{Q}$,
$\Sigma = (\mathcal{P}.\Sigma \setminus \mathcal{P}.par^{-1}_\Sigma(\mathtt{L})) \cup \mathbb{N} = (\mathcal{Q}.\Sigma \setminus \mathcal{Q}.par^{-1}_\Sigma(\mathtt{L})) \cup \mathbb{N}$.
Ausgabe: $true$ gdw. $\mathcal{Q}$ simuliert $\mathcal{P}$
$R_{\mathcal{P}||\mathcal{Q}} := Reach([\![\mathcal{P}||\mathcal{Q}]\!]_I)$
do
$R'_{\mathcal{P}||\mathcal{Q}} := R_{\mathcal{P}||\mathcal{Q}}$
forall $\sigma \in \Sigma$
if $S^0_\mathcal{P} \not\subseteq \{p \in S_\mathcal{P} \mid \exists q : (p,q) \in R_{\mathcal{P}||\mathcal{Q}}\}$ **then return** false

$$R_{\mathcal{P}||\mathcal{Q}} := R_{\mathcal{P}||\mathcal{Q}} \cap \left\{ (p,q) \in R_{\mathcal{P}||\mathcal{Q}} \;\middle|\; \begin{array}{c} \forall p' : \left(p \overset{\sigma}{\rightarrowtail}_\mathcal{P} p'\right) \Rightarrow \\ \left(\exists q' : q \overset{\sigma}{\rightarrowtail}_\mathcal{Q} q' \wedge (p',q') \in R_{\mathcal{P}||\mathcal{Q}} \right) \end{array} \right\}$$

while $R_{\mathcal{P}||\mathcal{Q}} \neq R'_{\mathcal{P}||\mathcal{Q}}$
return true

Abbildung 3.19: Algorithmus für den Nachweis einer Simulationsrelation.

Der Algorithmus für den Nachweis einer Simulationsrelation ist in Abbildung 3.19 dargestellt. Zunächst wird für die parallele Komposition von $\mathcal{P}$ und $\mathcal{Q}$ die Menge der erreichbaren Konfigurationen berechnet. Diese Menge von Paaren (p,q) wird in einem iterativen Verfahren als initiale Relation für die Annäherung an die Simulationsrelation zwischen $\mathcal{P}$ und $\mathcal{Q}$ verwendet. Dann werden in jedem Zyklus einer Fixpunktiteration alle enthaltenen Konfigurationen dahingehend überprüft, ob sie die oben aufgeführten Bedingungen erfüllen. Sind in der aktuellen Relation Konfigurationspaare enthalten, die die Bedingungen für Simulationsrelation verletzen, so werden diese Konfigurationen aus der Menge entfernt. Falls der Algorithmus eine Initialkonfiguration von $\mathcal{P}$ aus der Relation eliminiert, dann existiert keine Simulationsrelation und der Algorithmus kann die

Überprüfung mit negativem Ergebnis abbrechen. Sind keine Konfigurationen zu entfernen, dann ist der Fixpunkt erreicht und die Simulationsrelation ist fertig berechnet (und existiert somit).

Die Simulationsrelation ist im allgemeinen keine Teilmenge von $Reach([\![\mathcal{P}||\mathcal{Q}]\!]_I)$. Es ist jedoch korrekt, im Algorithmus von Abbildung 3.19 mit $Reach([\![\mathcal{P}||\mathcal{Q}]\!]_I)$ zu beginnen, und der Algorithmus findet eine Simulationsrelation, falls eine existiert.

Satz 3.7 *Sei $\mathcal{R}_S$ eine Simulationsrelation mit $\mathcal{R}_S \not\subseteq Reach([\![\mathcal{P}||\mathcal{Q}]\!]_I)$. Dann existiert eine Simulationsrelation $\mathcal{R}'_S \subseteq Reach([\![\mathcal{P}||\mathcal{Q}]\!]_I)$.*

Beweis: Eine Simulationsrelation $\mathcal{R}'_S$ kann aus $\mathcal{R}_S$ konstruiert werden durch $\mathcal{R}'_S = \mathcal{R}_S \cap Reach([\![\mathcal{P}||\mathcal{Q}]\!]_I)$. Bei diesem Schnitt werden nur solche Paare entfernt, die von der Initialkonfiguration ausgehend nicht erreicht werden können. Werden alle nicht erreichbaren Paare entfernt, bleibt kein die erste Forderung aus Definition 3.7 verletzendes Paar im Durchschnitt übrig. Die Initialkonfigurationen von $[\![\mathcal{P}]\!]_I$ bleiben im Durchschnitt erhalten, da sie in $Reach([\![\mathcal{P}||\mathcal{Q}]\!]_I)$ enthalten sind. □

3.5 Weitere Techniken zur Effizienzverbesserung

Im folgenden werden einige Techniken vorgestellt, die u. a. in Abhängigkeit vom zu verifizierenden Modell und dem Einsatzzweck zum Teil erhebliche Performanceverbesserungen bewirken können. Um die Auswirkungen der Techniken zu demonstrieren, werden auch hier einige Ergebnisse aus dem Abschnitt 5.4.2 vorweggenommen. Durch die Angabe von Meßwerten wird die Argumentation empirisch bestätigt.

Nachdem die getrennte Behandlung von diskreten und Zeit-Transitionen festgelegt wird, werden im folgenden vier verschiedene Repräsentationen der diskreten Transitionsrelation beschrieben. Nach dieser Vorstellung werden aus Experimenten mit diesen verschiedenen Repräsentationen stammende Meßreihen für zwei Beispiel-Modelle interpretiert. Die separaten Repräsentation der Zeit-Transition wurde beibehalten. Es wurde die Strategie verfolgt, bei der jeweils zwischen zwei Zeit-Transitionen der Fixpunkt bzgl. der diskreten Transitionen berechnet wird.

3.5.1 Gesonderte Behandlung der Zeit-Transitionsrelation

Zunächst wird die Charakteristik von diskreten im Vergleich zu den Zeit-Transitionen betrachtet. Eine Zeit-Transition erhöht den Wert einer jeden Uhr des Modells und führt somit zu Abhängigkeiten zwischen den Komponenten, selbst wenn zwischen ihnen keine direkte Kommunikation stattfindet. Diskrete Transitionen ohne Synchronisationsmarke führen nicht zu zusätzlichen Abhängigkeiten innerhalb des BDDs, der die Menge aller erreichbaren Konfigurationen repräsentiert. Deshalb wird die Repräsentation der Zeit-Transitionen von der Repräsentation der diskreten Transitionen getrennt und beim Algorithmus für die Erreichbarkeitsanalyse innerhalb einer Iteration folgende Basisstrategie benutzt: Zuerst wird der Fixpunkt der Erreichbarkeitsmenge unter ausschließlicher Anwendung von diskreten Transitionen berechnet. Erst danach werden die Nachfolger bzgl. der

Zeit-Transition berechnet. Zahlreiche Experimente mit Rabbit haben gezeigt, daß diese Strategie nicht nur erfolgreich, sondern auch notwendig ist[9].

3.5.2 Repräsentation der diskreten Transitionsrelation

In diesem Abschnitt werden verschiedene Repräsentationen der Transitionsrelation beschrieben und deren Effekte anhand der Analyse einiger Beispiele diskutiert [BH01].

3.5.2.1 Geteilte Transitionsrelation (GTR)

Die Betrachtung der monolithischen Transitionsrelation gab wichtige Einsichten für die Wahl der Variablenordnung, insbesondere für die automatische. Wie im Abschnitt 3.2.6 bereits ausgeführt, wird die Transitionsrelation selbst in der Praxis jedoch nicht als monolithische Vereinigung, sondern als geteilte Transitionsrelation repräsentiert, d. h. es werden einzelne partielle Transitionsrelationen als implizite Vereinigung betrachtet. Gründe dafür liegen einerseits in der Effizienz der sequentiellen Anwendung mehrerer kleiner Transitionsrelationen nacheinander, andererseits ist der Speicherplatzbedarf der geteilten Transitionsrelation linear begrenzt. Diese Erkenntnis stimmt mit den Ergebnissen anderer Forschungsgruppen überein [RAB+95].

Eine geteilte Transitionsrelation nach Abschnitt 3.2.6 besteht aus einer diskreten Transitionsrelation für jede Synchronisationsmarke $a \in \Sigma$:

$$\xrightarrow{a}(q_1, q_1', ..., q_n, q_n') = \bigcap\nolimits_{k\in\{1,2,...,n\}} \begin{cases} \xrightarrow{a}_k(q_k, q_k'), & falls\ a \in \Sigma_k \\ [\![q_k' = q_k]\!], & sonst \end{cases}$$

und einer Transitionsrelation für das Vergehen einer Zeiteinheit:

$$\xrightarrow{1}(q_1, q_1', ..., q_n, q_n') = \bigcap\nolimits_{k\in\{1,2,...,n\}} \xrightarrow{1}_k(q_k, q_k').$$

Bezüglich der Kofaktoren gilt für alle $a \in \Sigma$ und alle $i \in \{1, ..., n\}$:

$$\xrightarrow{a}(q_1, q_1', ..., q_n, q_n')|_{\{q_1,q_1',...,q_{i-1},q_{i-1}'\}} \subseteq \left\{ \emptyset, \bigcap\nolimits_{k\in\{i,...,n\}} \begin{cases} \xrightarrow{a}_k(q_k, q_k'), & falls\ a \in \Sigma_k \\ [\![q_k' = q_k]\!], & sonst \end{cases} \right\}.$$

Daraus folgt mit Satz 3.3, daß die BDD-Größe für $\xrightarrow{a}$ in $O(n)$ liegt. Dies gilt unabhängig von der Reihenfolge der Komponenten in der Variablenordnung, es wird nur vorausgesetzt, daß die Bits einer Komponente in der Variablenordnung zusammenhängende Positionen einnehmen. Analog liegt die Größe des BDDs für die Zeit-Transitionsrelation $\xrightarrow{1}$ in $O(n)$. Die Zahl der Synchronisationsmarken in Σ liegt ebenfalls in $O(n)$, da $\Sigma = \bigcup_{k\in\{1,...,n\}} \Sigma_k$. Damit liegt die Summe der BDD-Größen der partiellen Transitionsrelationen in $O(n^2)$. Der Speicheraufwand für die Repräsentation der geteilten Transitionsrelation ist also akzeptabel.

[9] In Gesprächen mit anderen Wissenschaftlern wurde deutlich, daß die Performance der BDD-basierten Analyse von Realzeit-Systemen unter anderem wegen der Nichtbeachtung dieser Strategie bisher weit unter den Erwartungen lag.

3.5.2.2 Teilweise vereinigte Transitionsrelation (TVTR)

Die erste Alternative zur Repräsentation der Transitionsrelation besteht darin, alle BDDs für die diskreten Transitionen eines Automaten (nur interne Transitionen werden betrachtet) zu vereinigen. Für diese Vereinigung kann die BDD-Größe kleiner sein als die Summe der einzelnen BDDs der partiellen Transitionsrelationen. Da die Komplexität der Erreichbarkeitsoperationen von der Größe der die Transitionsrelation repräsentierenden BDDs abhängt, kann somit auch die Berechnungszeit geringer werden.

Es gibt jedoch keine Garantie dafür, daß die BDD-Größe der TVTR kleiner ist als die summierte BDD-Größe der GTR. Ein Werkzeug könnte aber die TVTR berechnen und die Größen der beiden Repräsentationen der Transitionsrelation miteinander vergleichen, um zu entscheiden, welche Repräsentation für die Verifikation sinnvoll ist.

Genauso wie bei der GTR wächst auch die Größe der teilweise vereinigten Transitionsrelation quadratisch mit der Anzahl der im Modell vorhandenen Automaten. Dies ist bei den beiden folgenden Ansätzen im allgemeinen nicht der Fall. Trotzdem können sie prinzipiell zu kleineren BDD-Repräsentationen als bei der GTR oder TVTR führen.

3.5.2.3 Vereinigte Transitionsrelation (VTR)

Die Vereinigung sämtlicher diskreter Transitionen stellt eine weitere Alternative dar. Wenn die BDD-Größe der resultierenden Transitionsrelation klein genug ist, dann kann diese Repräsentation nützlich sein. Jedoch gibt es einige Argumente gegen die vereinigte Transitionsrelation:

- Die größten Zwischen-BDDs entstehen als Ergebnis des Durchschnitts einer Erreichbarkeitsmenge mit einer Transitionsrelation. Diese BDDs sind in der Tendenz um so kleiner, je kleiner die jeweils an der Operation beteiligten BDDs für die Transitionsrelation sind. Daher kann es deutlich vorteilhafter sein, viele kleine BDDs (TVTR, GTR) anstelle weniger großer BDDs (VTR) anzuwenden.

- Wird die vereinigte Transitionsrelation verwendet, werden bei einem Schritt der Fixpunktiteration alle mit genau einem Erreichbarkeitsschritt erreichbaren Nachfolgerkonfigurationen berechnet. Werden die partielle Transitionsrelation oder die teilweise vereinigte Transitionsrelation verwendet, können in einem Schritt der Fixpunktiteration mehrere Transitionen nacheinander schalten (da die berechneten neuen Konfigurationen sofort zur Erreichbarkeitsmenge hinzugefügt werden). Somit kann der Algorithmus in einer Fixpunktiteration schon viele Konfigurationen mit berechnen, die bei der VTR erst mit mehr als einer Iteration erreichbar sind. Dadurch werden bei der VTR, speziell bei vielen nebenläufigen Transitionen, meist mehr Iterationen für die Berechnung des diskreten Fixpunktes notwendig, was den eventuellen Vorteil der kleinen Transitionsrelation (VTR) wieder zunichte machen kann.

3.5.2.4 Transitive Hülle der Transitionsrelation (THTR)

Beim Erreichbarkeitsalgorithmus erfolgt zwischen je zwei Zeit-Transitionen eine Fixpunktiteration über die diskreten Transitionen, und bei der vereinigten Transitionsrelation existiert nur ein BDD für die Transitionsrelation. Deshalb liegt die Strategie nahe, die Berechnung der diskreten Fixpunkte zu verkürzen, indem für die Transitionsrelation die transitive Hülle errechnet wird. Dann würde ein BDD die Relation für die gesamte Fixpunktiteration repräsentieren. Diese Strategie führt besonders dann zur Performance-Verbesserung, wenn ein Modell eine große Anzahl an Fixpunktiteration benötigt.

Die transitive Hülle für eine einzelne Transition ist nicht sehr sinnvoll, weil in den meisten Fällen direkt nach dem Schalten einer Transition die gleiche Transition nicht noch einmal einen Fortschritt bringt (außer bei Schleifen). Somit wird nur die Hülle der vereinigten Transitionsrelation in Betracht gezogen oder ggf. die Hülle der teilweise vereinigten Transitionsrelation. Die transitive Hülle aller Transitionen ist ineffizient, wenn die BDD-Repräsentation sehr groß ist oder die Berechnung der transitiven Hülle selbst aufwendig ist.

3.5.2.5 Diskussion anhand von Meßergebnissen

Im folgenden werden einige Resultate aus Experimenten mit zwei verschiedenen Modellen wiedergegeben: Die Ergebnisse für das Modell mit acht parallel laufenden Mini-Automaten (TwoState8) nach Abbildung 3.23 auf Seite 94 werden in Tabelle 3.2 dargestellt, die Ergebnisse für das Modell mit 24 Fischer-Prozessen (Fischer24) nach Abbildung 3.11 auf Seite 74 in Tabelle 3.3. Die erste Zeile jeder Tabelle gibt die verwendete Repräsentation der Transitionsrelation an. Die zweite Zeile gibt die Laufzeit zur Berechnung der Erreichbarkeitsmenge wieder. Die Größe des größten in der diskreten Fixpunktiteration zwischen je zwei Zeit-Transitionen auftretenden Zwischen-BDDs wird aus der dritten Zeile ersichtlich. Ein wichtiger Einflußfaktor für die Performance ist die in der vierten Zeile aufgeführte Anzahl der bei der Analyse auszuführenden diskreten Transitionen in der Fixpunktiteration. Um die Struktur und Größe der Transitionsrelation zu verdeutlichen, werden die Anzahl der an der Repräsentation der vollständigen Transitionsrelation beteiligten BDDs in der fünften Zeile und die Gesamtgröße der Transitionsrelation in der sechsten Zeile der Tabelle angegeben.

Typ der Transitionsrelation	GTR	TVTR	VTR	THTR
Berechnungszeit (in Sekunden)	158.39	100.59	77.53	55.67
Knotenanzahl des größten Zwischen-BDD	165545	165545	159340	127388
Anzahl diskreter Schritte	2752	1376	530	172
Anzahl der BDDs zur Repräsentation der Transitionsrelation	16	8	1	1
Gesamtgröße der Transitionsrelation (in BDD-Knoten)	208	112	112	112

Tabelle 3.2: Verifikationsresultate mit vier verschiedenen Repräsentationen der Transitionsrelation (TwoState8).

Modell "TwoState8": Aus Tabelle 3.2 ist ersichtlich, daß die teilweise vereinigte Transitionsrelation die benötigte Rechenzeit für das Modell erheblich verringert. Der

Speicherbedarf des größten Zwischen-BDDs bleibt unverändert, was bedeutet, daß die alternierende Nachfolgerberechnung mit unterschiedlichen Transitionen eines Automaten bei diesem Modell nicht für die Größe des Zwischen-BDDs verantwortlich ist.

Die verringerte Laufzeit bei der TVTR resultiert aus der Tatsache, daß nur halb so viele diskrete Schritte berechnet werden müssen (da von jedem Automaten die beiden Transitionen vereinigt wurden). Die teilweise vereinigte Transitionsrelation benötigt halbsoviele BDD-Knoten zur Repräsentation als die beiden einzelnen Transitionen bei der GTR. Das zweimalige Berechnen der Nachfolger mit einer größeren Transitionsrelation braucht bei der GTR viel mehr Zeit als bei der TVTR.

Die Gesamtgröße verändert sich bei der vereinigten Transitionsrelation dadurch nicht, weil alle Automaten voneinander unabhängig sind. Durch diese vereinigte Repräsentation können in einem Schritt mehrere Folgekonfigurationen gleichzeitig berechnet werden, bei denen je ein Automaten einen Zustandsübergang vollzieht. Es wird schneller ein Fortschritt in der diskreten Fixpunktiteration erzielt, es werden also insgesamt viel weniger diskrete Transitionen geschaltet und der Algorithmus erhält einen nochmaligen Performancezuwachs. Die Größe der Zwischen-BDDs ist kleiner, weil nicht mehr so viele Zwischenergebnisse entstehen und die größten BDDs aus Abhängigkeiten resultierten, die bei der VTR wegen der disjunktiven Kombination der Transitionen mehrerer Automaten nicht mehr explizit vorkommen.

Da die Transitionsrelation durch die Vereinigung nicht größer geworden ist, ist ein weiterer Schritt zur Performancesteigerung offensichtlich: Durch die Berechnung der transitiven Hülle der vereinigten Transitionsrelation können weitere diskrete Transitionen vermieden werden, wobei insbesondere diejenigen Transitionen entfallen, die bei der Überprüfung des Fixpunkt-Kriteriums angewendet werden, um festzustellen, daß keine neuen Konfigurationen hinzugekommen sind. Die Laufzeitverkürzung resultiert aus der Verringerung der Anzahl der diskreten Transitionen. Da die Fixpunktiteration in einem Schritt berechnet wird, werden weniger große Zwischen-BDDs erzeugt. Interessant ist, daß die Größe der Zwischen-BDDs nur so wenig abfällt. Dies bestätigt empirisch, daß die Zeitabhängigkeiten für die zwischenzeitliche Aufblähung der BDDs in der Erreichbarkeitsanalyse verantwortlich ist. Zu beachten ist die Größe des End-BDDs für die Erreichbarkeitsmenge von 16 Knoten im extremen Gegensatz zur Größe des größten Zwischen-BDDs von 127.388 Knoten.

Typ der Transitionsrelation	GTR	TVTR	VTR
Berechnungszeit (in Sekunden)	37.74	42.10	487.79
Knotenanzahl des größten Zwischen-BDD	47627	47627	141956
Anzahl diskreter Schritte	2160	432	146
Anzahl der BDDs zur Repräsentation der Transitionsrelation	120	24	1
Gesamtgröße der Transitionsrelation (in BDD-Knoten)	1695	3160	67615

Tabelle 3.3: Verifikationsresultate mit drei verschiedenen Repräsentationen der Transitionsrelation (Fischer24).

Modell "Fischer24": Bei dem Modell für Fischers Protokoll führt das Vereinigen von Transitionsrelationen bei der TVTR und der VTR nicht zu einem besseren Laufzeitverhal-

ten, weil die Kommunikation innerhalb des Modells zu großen BDDs führt. In Tabelle 3.3 auf der vorherigen Seite wird reflektiert, daß die Anzahl der BDD-Knoten des größten Zwischen-BDD unverändert bleibt, wenn die TVTR statt der GTR verwendet wird.

Die durchschnittliche Größe einer Transitionsrelation bei der GTR beträgt 14 (1695 / 120) BDD-Knoten. Bei der TVTR ist diese Größe gestiegen auf 131 (3160 / 24) BDD-Knoten pro Transitionsrelation. Obwohl sich die Anzahl der diskreten Schritte bei der TVTR auf ein Fünftel der Anzahl bei der GTR verringert hat, steigt die Berechnungszeit etwas an. Dies resultiert aus den Operationen mit den ziemlich großen BDDs der Transitionsrelation.

Wird die vereinigte Transitionsrelation eingesetzt, so steigt die Anzahl der BDD-Knoten für die Transitionsrelation auf 67.615 Knoten (statt 3160 Knoten bei der TVTR). Die größten Zwischen-BDDs steigen auf das Dreifache der Größe bei der GTR (bzw. TVTR) an. Die disjunktive Kombination der Nachfolgerkonfigurationen verschiedener Komponenten vergrößert die Zwischen-BDDs. Aus diesen beiden Gründen ist eine enorme Verschlechterung der Berechnungsperformance zu verzeichnen.

Damit gilt für die vereinigte und für die teilweise vereinigte Transitionsrelation bei diesem Modell: Der Vorteil, weniger BDDs zur Repräsentation der Transitionsrelation zu haben, und die stark reduzierte Anzahl diskreter Schritte in den Fixpunktiterationen können den Performanceverlust durch die enorm großen BDDs für die Transitionsrelation und die großen BDDs für die Erreichbarkeitsmenge nicht ausgleichen.

Durch die schlecht komprimierte Repräsentation der vereinigten Transitionsrelationen wird von der Verwendung einer transitiven Hülle für die Transitionsrelation bei Fischers Protokoll abgesehen.

Modell "AND-Schaltung": Ein weiteres Beispiel, bei dem der Einsatz einer vom Standard abweichenden Repräsentation der Transitionsrelation zu erheblicher Performanceverbesserung führt, ist das bereits im Kapitel 2 vorgestellte Modell einer AND-Schaltung. In der Tabelle 5.3 auf Seite 175 im Abschnitt 5.4.2.2 wird verdeutlicht, daß die Verwendung einer Kombination der teilweise vereinigten Transitionsrelation mit der transitiven Hülle sich in stark verkürzter Laufzeit bemerkbar macht.

3.5.2.6 Schlußfolgerungen

Die Experimente mit verschiedenen Modellen führen zu folgender Argumentation: Die geteilte Transitionsrelation ist generell eine sehr gute Lösung, die einfach ist und höchstens quadratische Speicherplatzkomplexität aufweist und damit die Voraussetzung für eine ziemlich effiziente Analyse liefert. Wenn die BDD-Repräsentation der vereinigten Transitionsrelation (TVTR oder VTR) kleiner ist, ist mit diesen Repräsentationen eine Performancesteigerung möglich, wobei bei der VTR das Risiko explodierender Zwischen-BDDs besteht. Ist die transitive Hülle der vereinigten Transitionsrelation nicht viel größer als die geteilte Transitionsrelation, ist mit dieser Strategie eine weitere Performancesteigerung möglich. Somit kann ein Werkzeug vor dem Start aufwendiger Analysen verschiedene Transitionsrelationen berechnen und durch Vergleich der BDD-Größen die für die Erreichbarkeitsanalyse am besten geeignete auswählen.

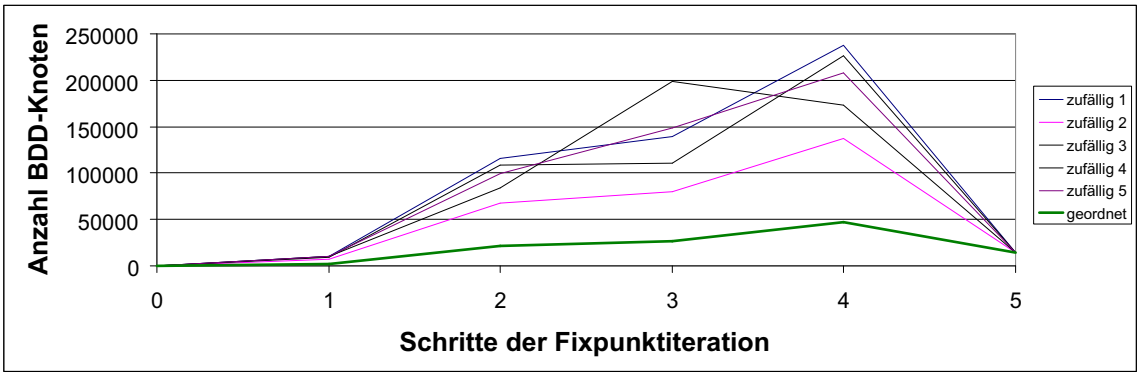


Abbildung 3.20: Anzahl der BDD-Knoten der Erreichbarkeitsmenge während der Fixpunktiteration bei verschiedenen Ordnungen der Anwendung der diskreten Transitionen in Abhängigkeit von den Zeitschritten (Fischer24).

3.5.3 Ordnung der diskreten Transitionen

Wird die geteilte oder teilweise vereinigte Transitionsrelation genutzt, sind mehrere diskrete Transitionen vorhanden und eine *Reihenfolge ihrer Anwendung* in der diskreten Fixpunktiteration ist festzulegen. Die zwischenzeitlich entstehende Menge erreichter Konfigurationen im Erreichbarkeitsalgorithmus (Zwischenwerte von R in Abbildung 3.5 auf Seite 62) hängt stark von dieser Ordnung ab und somit auch die Größe der diese Mengen repräsentierenden BDDs. Eine schlecht gewählte Ordnung der diskreten Transitionen führt zu wesentlich größeren Zwischen-BDDs als eine gut gewählte Ordnung. Im folgenden wird eine Heuristik zur Lösung dieses Problems vorgestellt.

Zur Illustration der Wichtigkeit dieser Betrachtung dient Abbildung 3.20. Es wird für jede diskrete Fixpunktiteration die Anzahl der BDD-Knoten des größten Erreichbarkeits-BDDs dargestellt. Die oberen fünf Graphen sind die Ergebnisse der Anwendung von zufällig gewählten Ordnungen der diskreten Transitionen. Der Graph mit den kleinen Zwischen-BDDs resultiert aus der folgenden Ordnung der Anwendung: Die internen Transitionen sind in der gleiche Reihenfolge angeordnet wie die Automaten, zu denen die Transitionen gehören, in der Variablenordnung eingeordnet sind. Als erstes kommen also die Transitionen des ersten Automaten in der Variablenordnung, dann die Transitionen des zweiten, u.s.w. Synchronisierte Transitionen stehen ganz am Anfang vor den internen Transitionen.

In der Literatur wird die Aufgabe dieses Ordnens ”Quantification Scheduling Problem” genannt. Chauhan et al. versuchen, eine gute Ordnung durch kombinatorische Optimierung zu finden [CCJ+01]. Sie benutzen die von Moon und Somenzi vorgeschlagene Abhängigkeitsmatrix [MHS00], um die Anzahl der beteiligten abhängigen Variablen so klein wie möglich zu halten und mittels Heuristiken wie *Hill Climbing* oder *Simulated Annealing* eine gute Reihenfolge zu finden. Bei dem dort betrachteten BDD-Paket wird die Variablenordnung dynamisch zur Laufzeit ermittelt. Darin differenziert sich Rabbit von vergleichbaren Werkzeugen: Sowohl die Variablenordnung als auch die Ordnung der diskreten Transitionen werden statisch *vor Beginn* der Analyse berechnet. Dabei fließen alle

verfügbaren Informationen über die Modellstruktur und den Kommunikationsgraphen ein. Dies bedeutet, daß die Ordnungen bei Rabbit durch abstrakte Information einer höheren Ebene (Modellstruktur) festgelegt werden, während vergleichbare Werkzeuge erst durch geeignete Verfahren (Heuristiken) verwertbare Informationen aus einer niedrigeren Ebene (Transitionen) mittels Abstraktion gewinnen müssen. Daß dieses Vorgehen zu Performanceverbesserungen führt, wird durch die Meßergebnisse in Abschnitt 5.4.2 nachgewiesen.

3.5.4 On-the-fly-Analyse

In diesem Abschnitt wird eine nützliche Technik vorgestellt, mit der die explodierende BDD-Größe aufgrund zeitbasierter Abhängigkeiten bei einigen Modellen vermieden werden kann. Selbst wenn die Komponenten nicht direkt miteinander kommunizieren, kann die BDD-Repräsentation der erreichbaren Konfigurationen während des Berechnungsprozesses sehr groß sein. Der Grund dafür ist, daß die Konfigurationen der einzelnen Komponenten vom Vergehen der Zeit abhängig sind. Somit sind die Komponenten indirekt miteinander verbunden.

Im Gegensatz zum Standard-Algorithmus, der zuerst die Menge aller erreichbarer Konfigurationen berechnet und dann überprüft, ob der Durchschnitt mit der Menge der verbotenen Konfigurationen leer ist, zeichnet sich der hier vorgeschlagene *On-the-fly*-Algorithmus[10] durch die folgenden Eigenschaften aus:

- **Speicherplatz sparen.** Daten, die für den weiteren Berechnungsvorgang nicht mehr von entscheidendem Interesse sind, werden aus dem Speicher entfernt. Dies bedeutet für die Erreichbarkeitsanalyse, daß nicht die volle Menge der erreichbaren Konfigurationen berechnet wird. Der Nachteil eines solchen Algorithmus ist, daß dieser möglicherweise mehr Iterationen und ein komplizierteres Abbruchkriterium benötigt als der Standardalgorithmus.

- **Schneller Abbruch.** Sobald Daten eine Entscheidung über das Resultat der Berechnung erlauben, bricht der Algorithmus den Berechnungsvorgang ab. Für die Erreichbarkeitsanalyse bedeutet dies, daß die Fixpunkt-Iteration sofort abgebrochen wird, sobald eine der verbotenen Konfigurationen berechnet worden ist. Diese Eigenschaft wurde bereits im Algorithmus von Abbildung 3.1 auf Seite 52 implementiert. Besonders sinnvoll ist dieses Vorgehen in der Entwicklungsphase eines Modells, wenn das Modell noch Fehler enthält.

Bei der Erreichbarkeitsanalyse ist es nicht unbedingt erforderlich, die volle Menge mit allen erreichbaren Konfigurationen zu speichern. Es ist prinzipiell ausreichend, jede erreichbare Konfiguration ein einziges Mal zu berechnen, zu prüfen, ob es sich um eine verbotene Konfiguration handelt, und sie nach dieser Überprüfung wieder aus dem Speicher zu löschen. Der hier verwendete Ansatz ist der, die Folge $Reach([\![\mathcal{A}]\!]_I)(0,0), Reach([\![\mathcal{A}]\!]_I)(1,1), ...$ zu berechnen und nur die aktuelle Menge

[10] Zur Unterscheidung erhält der bisher betrachtete Algorithmus die Bezeichnung *Full-set*-Berechnung.

Eingabe: Abgeschlossener Timed Automaton $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$
mit der ganzzahligen Semantik $[\![\mathcal{A}]\!]_I = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow)$,
Menge von Kontrollzuständen L^F
Ausgabe: $true$ gdw. $L^F \cap [\![[\![\mathcal{A}]\!]_C]\!]_R \neq \emptyset$

```
1  R := L^0 × {v^0}
2  DiscreteFixedPoint(A, R)
3  s := 0; p := 0
4  R_prev := ∅
5  while R_prev ⊉ R do
6    if p ≤ s/2 then
7      p := s
8      R_prev := R
9    R := {q' ∈ L × Val(X) | ∃q : q ∈ R ∧ q --1--> q'}
10   DiscreteFixedPoint(A, R)
11   s := s + 1
12   if R ∩ (L^F × Val(X)) ≠ ∅ then return true
13 return false
```

Abbildung 3.21: Algorithmus für die On-the-fly-Analyse.

$Reach([\![\mathcal{A}]\!]_I)(i,i)$ und eine der vorher berechneten Mengen im Speicher zu halten. Diese Strategie führt zu dem Problem, zu entscheiden, wann alle erreichbaren Konfigurationen überprüft worden sind. Im allgemeinen existiert kein $i \in \mathbb{N}$, so daß die Bedingung $Reach([\![\mathcal{A}]\!]_I)(i,i) \supseteq Reach([\![\mathcal{A}]\!]_I)(i+1,i+1)$ gilt, was ein einfaches Abbruchkriterium wäre.

Ein simpler Algorithmus könnte das Abbruchkriterium überprüfen, indem er alle berechneten Mengen $Reach([\![\mathcal{A}]\!]_I)(0,0), ..., Reach([\![\mathcal{A}]\!]_I)(i,i)$ speichert und überprüft, ob $Reach([\![\mathcal{A}]\!]_I)(i+1,i+1)$ in einer der bereits berechneten Mengen enthalten ist. Dieses Vorgehen würde zu einem enormen Speicherverbrauch für die Erreichbarkeitsanalyse führen.

Der Algorithmus in Abbildung 3.21 speichert nicht alle bereits berechneten Mengen $Reach([\![\mathcal{A}]\!]_I)(i,i)$ der vorhergehenden Schritte, sondern nur eine einzige, im Algorithmus R_{prev} genannte. Die Berechnung des Fixpunktes der Menge R unter ausschließlicher Anwendung diskreter Transitionen ist im Algorithmus mit `DiscreteFixedPoint`$(\mathcal{A}, R)$ abgekürzt.

Im folgenden werden die Terminierung und der Aufwand für die On-the-fly-Analyse nachgewiesen.

Satz 3.8 *Seien $T_{\mathcal{A}}$ und $\Delta_{\mathcal{A}}$ zwei einem abgeschlossenen Timed Automaton $\mathcal{A}$ zugeordnete natürliche Zahlen mit den folgenden Eigenschaften:*

1. $\Delta_{\mathcal{A}} > 0$,

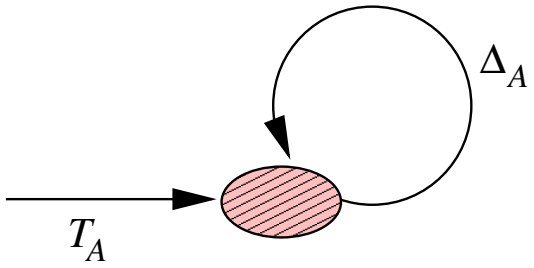


Abbildung 3.22: Kleinster gemeinsamer Zyklus für alle Läufe.

2. $Reach(\llbracket\mathcal{A}\rrbracket_I)(T_\mathcal{A}, T_\mathcal{A}) \supseteq Reach(\llbracket\mathcal{A}\rrbracket_I)(T_\mathcal{A}+\Delta_\mathcal{A}, T_\mathcal{A}+\Delta_\mathcal{A})$ *und*

3. *für alle* $T', \Delta' \in \mathbb{N}$ *mit den Eigenschaften 1 und 2 gilt:*

   - $T'+\Delta' > T_\mathcal{A}+\Delta_\mathcal{A}$ *oder*
   - $T'+\Delta' = T_\mathcal{A}+\Delta_\mathcal{A}$ *und* $\Delta' \geq \Delta_\mathcal{A}$.

Der Algorithmus für die On-the-fly-Analyse nach Abbildung 3.21 terminiert nach weniger als $3(T_\mathcal{A}+\Delta_\mathcal{A})$ *Iterationen der (while-) Schleife und liefert das Ergebnis* $true$ *gdw.* $L^F \cap \llbracket\llbracket\mathcal{A}\rrbracket_C\rrbracket_R \neq \emptyset$.

Beweis: Wegen der Endlichkeit des Zustandsraumes (bei Benutzung der ganzzahligen Semantik) existieren natürliche Zahlen $T_\mathcal{A}$ und $\Delta_\mathcal{A}$ mit den definierten Eigenschaften. Nach der Berechnung von $Reach(\llbracket\mathcal{A}\rrbracket_I)(0,0), ..., Reach(\llbracket\mathcal{A}\rrbracket_I)(T_\mathcal{A}+\Delta_\mathcal{A}, T_\mathcal{A}+\Delta_\mathcal{A})$ wurden tatsächlich alle erreichbaren Konfigurationen überprüft, und der Algorithmus kann terminieren.

Während jeder Überprüfung des Abbruchkriteriums $R_{prev} \supseteq R$ in Zeile 5 gilt $R = Reach(\llbracket\mathcal{A}\rrbracket_I)(s,s)$ und $R_{prev} = Reach(\llbracket\mathcal{A}\rrbracket_I)(p,p)$, wobei p die größte Zweierpotenz ist, die kleiner als s ist (ausgenommen $R_{prev} = \emptyset$ falls $s = 0$; $p = 0$ falls $s \leq 1$). Sei k die kleinste Zweierpotenz mit $k \geq T_\mathcal{A}$ und $k \geq \Delta_\mathcal{A}$. Wegen $Reach(\llbracket\mathcal{A}\rrbracket_I)(k,k) \supseteq Reach(\llbracket\mathcal{A}\rrbracket_I)(k+\Delta_\mathcal{A}, k+\Delta_\mathcal{A})$ terminiert der Algorithmus spätestens nach der Berechnung von $Reach(\llbracket\mathcal{A}\rrbracket_I)(k+\Delta_\mathcal{A}, k+\Delta_\mathcal{A})$. Weil $k+\Delta_\mathcal{A} < 3(T_\mathcal{A}+\Delta_\mathcal{A})$ gilt, benötigt der Algorithmus höchstens die dreifache Anzahl von Iterationen des Algorithmus, der alle Konfigurationsmengen $Reach(\llbracket\mathcal{A}\rrbracket_I)(i,i)$ speichert. Die Ungleichung $k+\Delta_\mathcal{A} < 3(T_\mathcal{A}+\Delta_\mathcal{A})$ folgt aus $k < 2\Delta_\mathcal{A} \leq 2(T_\mathcal{A}+\Delta_\mathcal{A})$. □

Abbildung 3.22 illustriert diese Situation. Nach $T_\mathcal{A}$ Schritten ist eine Konfigurationsmenge erreicht, die alle nach $T_\mathcal{A}+\Delta_\mathcal{A}$ Schritten erreichbaren Konfigurationen beinhaltet.

Ein potentielles Problem für die Effizienz dieses Algorithmus ist, daß die Anzahl der Iterationen $T_\mathcal{A}+\Delta_\mathcal{A}$ beträchtlich größer sein kann im Vergleich zur Anzahl der Iterationen des herkömmlichen Algorithmus, welche $min\Big(\{k \in \mathbb{N} \mid Reach(\llbracket\mathcal{A}\rrbracket_I)(0,k) = Reach(\llbracket\mathcal{A}\rrbracket_I)(0,k+1)\}\Big)+1$ beträgt. Die Erfahrung im praktischen Umgang mit verschiedenen Modellen führt zu der Vermutung, daß diese Strategie zu enormer Performancesteigerung bei Modellen mit wenig expliziter

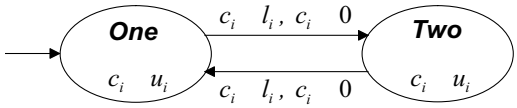


Abbildung 3.23: Mini-Automat mit zwei Zuständen. c_i ist eine Uhr, u_i und l_i sind Konstanten.

# Kompon.	4	8	16	32	64	128	256
# Konfigur.	$3.3 \cdot 10^5$	$1.1 \cdot 10^{11}$	$1.2 \cdot 10^{22}$	$1.5 \cdot 10^{44}$	$2.2 \cdot 10^{88}$	$4.8 \cdot 10^{176}$	$2.3 \cdot 10^{353}$
BDD-Größe	8	16	32	64	128	256	512
On-the-fly	0.06	0.19	1.23	6.11	26.8	123	582
Full-set	0.75	63.7	>400 MB				

Tabelle 3.4: Laufzeit zur Berechnung aller erreichbarer Konfigurationen für das Modell mit zwei Zuständen und einer Uhr je Komponente (für $u_i = 12, l_i = 9$).

Kommunikation führt und sich nicht sehr nachteilig bei Modellen mit viel expliziter Kommunikation auswirkt.

Meßergebnisse. In der Tabelle 3.4 wird gezeigt, daß nicht nur die Variablenordnung von entscheidender Bedeutung für die Effizient ist, sondern auch die Strategie des Algorithmus: die On-the-fly-Analyse kann die Effizienz enorm verbessern. Als Beispiel wird das kleine Zwei-Zustands-Modell aus [BMPY97] benutzt, dessen Automat in Abbildung 3.23 dargestellt ist. Für die Komposition mehrerer solcher Automaten wird in der zweiten Zeile der Tabelle die Anzahl der erreichbaren Konfigurationen in Abhängigkeit von der Anzahl der beteiligten Automaten (erste Zeile) dargestellt, wobei zu jedem Automaten eine Uhr gehört. Die Größe des BDD für die Menge aller erreichbarer Konfigurationen ist in der dritten Zeile dargestellt. Die vierte Zeile enthält die Berechnungszeiten bei der Benutzung des On-the-fly-Algorithmus und die fünfte Zeile verdeutlicht die Explosion der Repräsentation der Zwischenergebnisse bei einem Algorithmus, der alle erreichten Konfigurationen speichert (vgl. den Algorithmus in Abbildung 3.1 auf Seite 52).

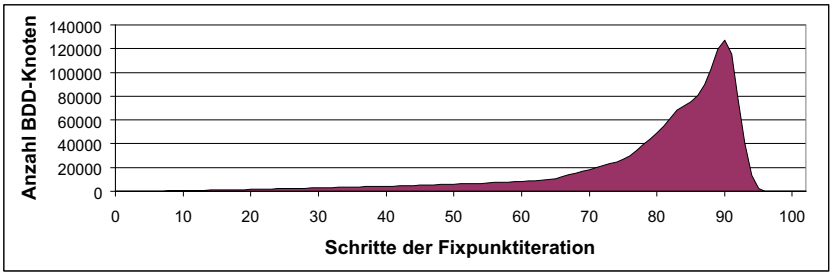


Abbildung 3.24: Anzahl der BDD-Knoten der Erreichbarkeitsmenge während der Iterationen mit *Full-set*–Berechnung (TwoState8).

Abbildung 3.24 visualisiert die Situation beim herkömmlichen Algorithmus: Ein starkes Wachstum der Repräsentation der Zwischenresultate der Erreichbarkeitsmenge während der Fixpunktiteration wird deutlich. Die Darstellung zeigt die Knotenanzahl

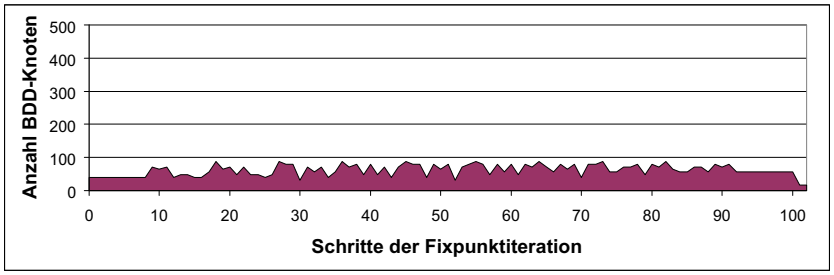


Abbildung 3.25: Anzahl der BDD-Knoten der Erreichbarkeitsmenge während der Iterationen mit *On-the-fly*–Berechnung (TwoState8).

des größten Zwischen-BDD jeweils zwischen zwei Zeit-Transitionen. Obwohl die verschiedenen Komponenten nicht direkt miteinander kommunizieren, sind ihre Konfigurationen voneinander abhängig wegen der Zeit-Transition. Die On-the-fly-Version profitiert von der Tatsache, daß die Konfigurationen zu einem bestimmten Zeitpunkt voneinander unabhängig sind; die maximale BDD-Größe beträgt weniger als 44 Knoten (vgl. Abbildung 3.25).

3.5.5 Wahl der Hash-Funktion

Einem die Performance eines BDD-Pakets sehr stark beeinflussenden Parameter wurde in der Literatur bisher erstaunlich wenig Aufmerksamkeit geschenkt. In einem BDD-Paket werden zur effizienten Suche von BDD-Knoten Hash-Tabellen und Caches eingesetzt. Sowohl in der Hash-Tabelle der BDD-Knoten als auch beim Cache für die Operationen ist die gute Streuung der Hash-Funktion *eine* signifikante Voraussetzung für gute Performance des BDD-Pakets. Diese wird von den gängigen BDD-Paketen im allgemeinen auch berücksichtigt.

Die für die Berechnung des Hash-Wertes selbst benötigte Laufzeit minimal zu halten, ist jedoch eine *weitere* wichtige Voraussetzung. Da die Hash-Funktion vor jedem Operationsaufruf für die Berechnung des Cache-Index und vor jedem Erzeugen eines neuen BDD-Knotens berechnet wird, geht die Laufzeit dieser Funktion direkt in die Gesamtlaufzeit ein. In einer an der BTU Cottbus angefertigten Studie wurde empirisch nachgewiesen, daß die Laufzeit der Hash-Funktion einen großen Anteil an der Gesamtlaufzeit hat [Noa99]. Aus diesem Grund wird in dem in Rabbit integrierten BDD-Paket eine sehr gut streuende Hash-Funktion verwendet, die zur Minimierung des Berechnungsaufwands ausschließlich die Operationen bitweises Verschieben, Addition und bitweises XOR verwendet.

3.5.6 Behandlung inaktiver Uhren

In der Erreichbarkeitsanalyse kommt es darauf an, möglichst wenige Schritte der Fixpunktiteration durchführen zu müssen und bei der Berechnung möglichst komprimiert repräsentierbare Mengen von Uhrenbelegungen zu erhalten.

Es ist zu beobachten, daß in den meisten Modellen Zustände vorkommen, in denen die Werte einiger Uhren keine Rolle spielen. Es wäre ein unnötiger Aufwand, für diese Uhren verschiedene auftretende Wertekombinationen zu speichern. Daws und Yovine haben dieses Verhalten bei der DBM-basierten Erreichbarkeitsanalyse beobachtet und eine entsprechende Reduktion des Modells vorgeschlagen [DY96].

Definition 3.8 *Eine Uhr ist in einem Zustand* **aktiv***, wenn ihr Wert Einfluß auf das zukünftige Verhalten des Automaten hat, d. h.*

- *wenn die Uhr in der Invariante des Zustands oder dem Wächter eines ausgehenden Zustandsübergangs auftritt oder*
- *wenn die Uhr im Zielzustand eines ausgehenden Zustandsübergangs aktiv ist und von diesem Zustandsübergang nicht zurückgesetzt wird.*

Durch diese Betrachtungen kann eine generelle Richtlinie für die zu analysierenden Modelle festgelegt werden, bei denen zur Vermeidung von unnötigem Speicher- und Rechenzeit-Aufwand eine Sonderbehandlung der inaktiven Uhren erfolgt:
Für Zustände, in denen die Uhrenwerte keine Rolle spielen, werden die inaktiven Uhren vor den Eintritt in den Zustand auf den ihnen zugeordneten Maximalwert gesetzt. Somit werden bei einer Zeit-Transition keine zusätzlichen Uhrenwerte für diese Uhr erzeugt.

Diese Richtlinie kann problemlos eingehalten werden, indem das Modell von einem Werkzeug durch statische Analyse auf inaktive Uhren geprüft und anschließend durch automatische Transformation angepaßt wird.

Modell	Fischer16	Fischer32	AND4	AND8
Ohne Anhalten inaktiver Uhren:				
Verifikationszeit	6.70	62.8	24.7	3170
Anzahl BDD-Knoten	22484	95725	122118	–
Mit Anhalten inaktiver Uhren:				
Verifikationszeit	6.42	61.5	9.54	450
Anzahl BDD-Knoten	6190	24983	45869	712469

Tabelle 3.5: Performanceverbesserung durch ”Anhalten” inaktiver Uhren.

Meßergebnisse. An einigen Beispielen wird nachgewiesen, daß die Richtlinie zur Behandlung inaktiver Variablen tatsächlich zu Performanceverbesserung führt. In der Tabelle 3.5 werden Meßergebnisse für Modelle von Fischers Protokoll mit 16 und 32 Prozessen sowie von einer AND-Schaltung mit 4 und 8 Eingängen in zwei Versionen wiedergegeben. Beim ersten Experiment werden die Verifikationszeit der On-the-fly-Berechnung (in Sekunden) und die Größe der Erreichbarkeitsmenge (in BDD-Knoten) für Modelle aufgeführt, in denen inaktive Uhren nicht angehalten werden. Im zweiten Experiment werden entsprechende Werte für die Modelle unter Berücksichtigung der inaktiven Uhren dargestellt. Die Meßergebnisse zeigen einen deutlichen Effizienzgewinn bei der vorgeschlagenen Sonderbehandlung von inaktiven Uhren.

Im verbleibenden Abschnitt dieses Kapitels wird auf einen Effekt eingegangen, der zu einer Verkürzung der Verifikationszeit führen kann, jedoch weder bei normalen Analysen in dieser Arbeit verwendet noch in die Werkzeugimplementierung integriert wurde.

Kernaussagen im theoretischen Teil dieser Arbeit werden formal bewiesen. Die Tauglichkeit der Modellierungskonzepte und die Performance der Werkzeugimplementierung werden im praktischen Teil empirisch validiert. Dazwischen liegt die sogenannte *Argumentative Begründung*. Als äußere Form wird das folgende Schema verwendet:

Behauptung 3.9 *Die Aussage wird zunächst als Behauptung formuliert.*

Begründung: Ähnlich dem Beweis eines Satzes erfolgt in der Begründung die Rechtfertigung der Behauptung. Jedoch erhebt die argumentative Begründung im Gegensatz zum formalen Beweis nicht den Anspruch an formale Exaktheit und Ausführlichkeit. Vielmehr soll in der Begründung skizzenhaft eine verbale Argumentation dargestellt werden, die auf den formalen Grundlagen beruht und die für den Beweis notwendigen Ideen und Strategien enthält. Abschließend werden Behauptung und Argumente an Versuchsreihen nachvollzogen und somit empirisch nachgewiesen. □

Abbildung 3.26: Schema Begründung.

3.5.7 Zeitabhängigkeiten explizit modellieren

In diesem Abschnitt wird gezeigt, daß die Reduktion der Anzahl von Uhren im Modell nicht unbedingt zu Effizienzsteigerung führt. Es wird ein Modell vorgestellt, bei dem das Hinzufügen einer zusätzlichen Uhr zu einer drastischen Performancesteigerung führt! Der Grund dafür liegt in der Rolle der zusätzlichen Uhr: Sie dient nicht der Modellierung eines bestimmten Verhaltens des Systems, sondern lediglich der *expliziten Darstellung* sonst nur implizit vorhandener Abhängigkeiten.

Behauptung 3.10 *Durch Hinzufügen einer zusätzlichen Uhr bei Fischers Protokoll für gegenseitigen Ausschluß kann eine Performanceverbesserung erzielt werden. Die Speicherkomplexität der BDD-Repräsentation der Erreichbarkeitsmenge für Fischers Protokoll beträgt* $S(n,m) = n^2 \cdot m^3 \cdot \log m$*, wobei* n *die Anzahl der an Fischers Protokoll beteiligten Prozesse und* m *die größte in diesem Modell auftretende Konstante für den Vergleich mit einer Uhr ist. Durch eine zusätzliche Uhr kann diese auf* $S(n,m) = n^2 \cdot m^2 \cdot \log m$ *reduziert werden.*

Begründung: Bei Fischers Protokoll (siehe Abbildung 3.11 auf Seite 74) gibt es zwei wesentliche implizite Zeitabhängigkeiten zwischen den Prozessen. Sei der i-te Prozeß betrachtet. Er befinde sich im Zustand *Assign*. Dann ist der Wert seiner Uhr x_i immer größergleich dem Wert der Uhr eines beliebigen Prozesses, der bereits in den Zustand *Wait* übergegangen ist[11]. Dieser Grundsatz gilt, weil der Zustand *Assign* nur dann

[11] Ein Prozess ist nicht gezwungen, den Zustand *Wait* zu verlassen. So kann es Prozesse geben, die noch von einem vorherigen Versuch im Zustand *Wait* stehen und eine Uhr mit einem größeren Wert haben. Diese

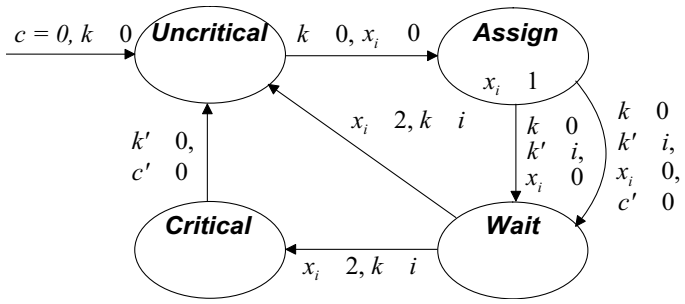


Abbildung 3.27: Prozeß i von Fischers Protokoll mit globaler Uhr c.

betreten werden darf, wenn der Wert der gemeinsamen Variable k noch nicht verändert wurde ($k = 0$). Dies heißt wiederum, daß noch kein Prozeß den Zustand *Assign* betreten hat. Formal: $loc(p_i) = l$ bezeichne die Eigenschaft, daß sich der Automat für Prozeß i im Zustand l befindet. Dann gilt:

$$\forall i, j : loc(p_i) = \textit{Assign} \ \wedge \ loc(p_j) = \textit{Wait} \Rightarrow x_i \geq x_j,$$

d. h. es muß folgende Abhängigkeit in der Erreichbarkeitsmenge enthalten sein:

$$\forall i : loc(p_i) = \textit{Assign} \ \Rightarrow \ x_i \geq \max_{j \in M}(x_j), M = \{m \mid loc(p_m) = \textit{Wait}\}.$$

Die zweite implizite Abhängigkeit der Prozesse untereinander betrifft die Umkehrung der oben behandelten Beobachtung. Befinde sich der i-te Prozeß im Zustand *Wait*. Dann ist der Wert seiner Uhr grundsätzlich kleiner als der Wert der Uhr eines beliebigen Prozesses, der sich noch im Zustand *Assign* befindet. Formal gilt dann:

$$\forall i, j : loc(p_i) = \textit{Wait} \ \wedge \ loc(p_j) = \textit{Assign} \Rightarrow x_i \leq x_j,$$

d. h. es muß folgende Abhängigkeit in der Erreichbarkeitsmenge enthalten sein:

$$\forall i : loc(p_i) = \textit{Wait} \ \Rightarrow \ x_i \leq \min_{j \in M}(x_j), M = \{m \mid loc(p_m) = \textit{Assign}\}.$$

Alle Abhängigkeiten der Prozesse untereinander entsprechen im BDD Abhängigkeiten der Variablen. Seien der Prozeß i und alle in der Variablenordnung vor dem Prozeß i auftretenden Prozesse betrachtet. Sei weiter $\mathcal{B}$ der (Teil-) BDD für die Menge aller Konfigurationen, bei denen der Prozeß i als erster in den Zustand *Wait* übergegangen ist. Daraus ergibt sich, daß in $\mathcal{B}$ Abhängigkeiten zu allen anderen Prozessen existieren, deren Automat sich im Zustand *Assign* oder *Wait* befindet (siehe oben ausgeführte Abhängigkeiten). Da der i-te Prozeß in der Variablenordnung nicht vor all diesen Prozessen steht, führt dies zu einem großen BDD.

Dieses Problem kann gelöst werden durch eine zusätzliche Uhr c, die mißt, wie lange k den Wert 0 hat (wenn $k = 0$) bzw. wie lange k ungleich 0 ist (wenn $k \neq 0$). Das

Konfigurationen haben keinen großen Einfluß auf die Komplexitätsbetrachtung und könnten durch Einfügen einer Invariante $x_i \leq 2$ für den Zustand *Wait* ausgeschlossen werden.

Konstantengröße	2	4	8	16	32
Fischer8, ohne globale Uhr:					
Berechnungszeit	0.44	2.89	27,6	368	
Anzahl BDD-Knoten	1345	5427	30556	214065	
mit globaler Uhr:					
Berechnungszeit	0.47	2.54	19.9	190	2630
Anzahl BDD-Knoten	1806	6545	25635	107575	470923
Fischer16, ohne globale Uhr:					
Berechnungszeit	5.19	50.3	621		
Anzahl BDD-Knoten	5502	25456	157993		
mit globaler Uhr:					
Berechnungszeit	4.60	35.2	303	5430	
Anzahl BDD-Knoten	7023	27202	110932	477976	
Fischer32, ohne globale Uhr:					
Berechnungszeit	51.1	624			
Anzahl BDD-Knoten	22071	108329			
mit globaler Uhr:					
Berechnungszeit	37.2	377	6150		
Anzahl BDD-Knoten	27440	109603	454709		

Tabelle 3.6: Performanceverbesserung durch eine zusätzliche Uhr für verschiedene Fischer-Modelle in Abhängigkeit von der Konstantengröße.

Automatenmodell ist in Abbildung 3.27 auf der vorherigen Seite dargestellt. Nun gilt für die Uhr c und für die Prozesse p_i:

$$\forall i : loc(p_i) = \textsf{Assign} \;\Rightarrow\; x_i \geq c \quad \text{und} \quad \forall i : loc(p_i) = \textsf{Wait} \;\Rightarrow\; x_i \leq c.$$

Diese Uhr muß gemeinsam mit der Variable k ganz am Anfang der Variablenordnung stehen. Durch das Vorziehen der globalen Uhr wird erreicht, daß die gesamte Information bzgl. der Abhängigkeiten gebündelt vorgezogen worden ist. Es werden über die gesamte Tiefe des BDDs die Kofaktoren bzgl. der einen globalen Uhr gespeichert.

Die Auswirkungen dieses Effekts auf die Speicherkomplexität wird im folgenden begründet. Sei n die Anzahl der an Fischers Protokoll beteiligten Prozesse und m die größte in diesem Modell auftretende Konstante für den Vergleich mit einer Uhr. Dann ist $S(n, m) = n^2 \cdot m^3 \cdot \log m$ die Speicherkomplexität der BDD-Repräsentation der Erreichbarkeitsmenge für Fischers Protokoll ohne die vorgeschlagene globale Uhr. Die Größe n^2 resultiert aus der Anzahl der beteiligten Prozesse. Die Tiefe des BDDs hängt von der Anzahl der Variablen ab und die Breite von der Anzahl der durch die Variable k erzeugten Kofaktoren (siehe Abschnitt 3.3.5 für eine genaue Beschreibung dieser Situation aufgrund der Größenschätzung). Ein weiterer Einflußfaktor für die Tiefe des BDDs ist die Anzahl der Bits für die Kodierung der Uhren, die wiederum von der Konstantengröße abhängt (Faktor $\log m$).

Durch den Einsatz der globalen Uhr wird eine Speicherkomplexität von $S(n, m) = n^2 \cdot m^2 \cdot \log m$ erreicht (siehe Tabelle 3.6 für die entsprechenden Meßwerte). Die Redukti-

on der Komplexität um eine Potenz in der Abhängigkeit von der Konstantengröße entsteht dadurch, daß nunmehr nicht alle Kombinationen der Uhrenwerte beim Eintreten in den Zustand *Wait* im BDD gespeichert werden müssen ($O(m^2)$ verschiedene Möglichkeiten bzw. Kofaktoren für Minimal- und Maximal-Wert), sondern lediglich alle Werte der globalen Uhr c ($O(m)$ Kofaktoren).

Durch Experimente konnte diese Argumentation bestätigt werden. In den Spalten für die Konstantengrößen 2 und 4 wird deutlich, daß trotz *größerer Anzahl* an BDD-Knoten zur Berechnung der Erreichbarkeitsmenge *weniger Zeit* notwendig ist. Offensichtlich hat die explizite Repräsentation der Zeitabhängigkeiten positiven Einfluß auf die BDD-Algorithmen. Das repräsentierende BDD ist noch größer, weil der Mehraufwand für die Speicherung der zusätzlichen Uhrenwerte noch nicht durch die insgesamt kompaktere Darstellung aufgewogen wird. Bei dem kleinen Modell für 8 Fischer-Prozesse ist dieser Effekt noch nicht sichtbar; die Berechnungszeiten unterscheiden sich kaum. In der Spalte für Konstantengröße 8 kommt der Effekt ganz zum Tragen: Obwohl im Modell eine zusätzliche Uhr vorhanden ist, ist eine deutliche Abnahme der Rechenzeit *und* der Anzahl an BDD-Knoten nachweisbar. Schon bei der Konstantengröße 16 kommt es zu einer Halbierung an Rechenzeit und Speicheraufwand. Bei den Experimenten mit globaler Uhr ist ein quadratischer Anstieg der Anzahl an BDD-Knoten zu verzeichnen (zuzüglich des logarithmischen Faktors). Dagegen nähert sich der Anstieg bei den Experimenten ohne globale Uhr einer kubischen Funktion an; dies wird aufgrund der begrenzten Meßreihen nur bei 8 Fischer-Prozessen deutlich. □

Die Reduktion der Speicherkomplexität wurde hier lediglich verbal begründet und empirische nachgewiesen; es erfolgt kein formaler Beweis, da diese Betrachtung in der vorliegenden Arbeit eine untergeordnete Rolle spielt. Auf eine Begründung kann jedoch nicht ganz verzichtet werden, um eine Lösung zur Erklärung einiger, sonst recht merkwürdig erscheinender, Effekte bzgl. der Performance bei größer werdenden Konstanten geben zu können.

3.6 Zusammenfassung

Die Erreichbarkeitsanalyse für Timed Automata ist zwar schon in Werkzeugen wie z. B. Kronos und Uppaal implementiert worden, jedoch stellt der explodierende Verbrauch an Prozessorzeit zur Berechnung und an Hauptspeicher zur Repräsentation der erreichbaren Konfigurationen nach wie vor eines der Hauptprobleme der Anwendung solcher Werkzeuge dar. Die Datenstrukturen und Algorithmen zur Berechnung der Konfigurationsmengen sind von entscheidender Bedeutung. Die Lösungsvorschläge dieses Kapitels beruhen auf der Verwendung von Binary Decision Diagrams als Basis-Datenstruktur.

Die Schwierigkeit der Repräsentation besteht darin, daß die Konfigurationen aus einem diskreten Zustand des Automaten und der zugeordneten Menge von kontinuierlichen Uhrenbelegungen (einer Untermenge von $\mathbb{R}_+^d$) bestehen. Im Bereich der reaktiven Systeme werden für die symbolische Repräsentation von Mengen von diskreten Zuständen bereits vielfach BDDs eingesetzt. Kronos und Uppaal benutzen für die Repräsentation von Men-

gen von Uhrenbelegungen eine auf Difference Bound Matrices (DBMs) basierende Datenstruktur. Dies führt zu zwei bedeutenden Engpässen in der Performance:

- Bei der Verwendung von DBMs ist keine effiziente Repräsentation von nichtkonvexen Mengen möglich.
- Die Anwendung verschiedener Datenstrukturen für Zustände und Uhrenbelegungen führt oft zu einer ineffizienten Repräsentation von Konfigurationsmengen mit vielen Zuständen.

Eine einheitliche Repräsentation von Zuständen und Uhrenbelegungen als BDDs wurde von Asarin et al. bereits vorgeschlagen [ABK+97]. Die Realisierung der vorgeschlagenen Konzepte in Werkzeugimplementierungen führte jedoch nicht zum gewünschten Erfolg. Das Verwenden von BDDs zur Repräsentation der Mengen allein kann die Probleme nicht lösen, wenn bestimmte Parameter nicht beachtet werden, die für die Effizienz der BDD-Repräsentation von entscheidender Bedeutung sind. So wurden in den veröffentlichten Ansätzen lediglich Standard-BDD-Pakete unter Verwendung des dynamischen Variablenordnens genutzt und die Rolle der Zeit-Transition nicht untersucht. Der Beitrag des vorliegenden Kapitels liegt in der Lösung dieser Probleme.

Basierend auf der ganzzahligen Semantik wurde eine Implementierung vorgeschlagen, die die diskreten Zustände der Automaten und den kontinuierlichen Zustandsraum der Uhren in einer einheitlichen Repräsentation als BDD darstellt. Ein wesentlicher Beitrag der Arbeit besteht in der Entwicklung einer Strategie zur automatischen Berechnung einer guten Variablenordnung. Es wurde eine obere Schranke für die Größe des BDDs der Transitionsrelation hergeleitet und formal bewiesen. Um aus einer Menge möglicher Variablenordnungen eine gute Auswahl treffen zu können, wurde aus der oberen Schranke eine Schätzfunktion für die Größe des BDD für die Erreichbarkeitsmenge abgeleitet [Bey01b]. Die Größenschätzung deutet darauf hin, daß die Berechnung der Erreichbarkeitsmenge für bestimmte Modelle in polynomieller Zeit- und Platzkomplexität durchgeführt werden kann. Dies wird im Kapitel 5 für mehrere Modelle empirisch bestätigt. Durch die Lösungsvorschläge dieses Kapitels können die Probleme der bestehenden BDD-basierten Ansätze weitgehend gelöst werden (siehe Forderung 2 auf Seite 14); die BDD-Repräsentation mit schätzungsbasiertem Variablenordnen stellt den *zweiten Schritt zur Bewältigung großer Systeme dar*.

Um durch die Verwendung von modularen Beweisen und kompositionellen Techniken auch größere Modelle verifizieren zu können, wurde eine Verfeinerungsanalyse realisiert. Durch die Kombination dieses Ansatzes mit der herkömmlichen Erreichbarkeitsanalyse können selbst viele der bisher nicht algorithmisch handhabbaren Verifikationsaufgaben effizient gelöst werden. Mit der Verfeinerungsanalyse ist der *dritter Schritt zur Bewältigung großer Systeme* umgesetzt worden.

Weiterhin wurden Fragen, die sich aus der Repräsentation der Transitionsrelation und der Erreichbarkeitsmenge ergeben, eingehend untersucht und über die Basistechniken hinausgehende Strategien zur weiteren Performancesteigerung angegeben: Behandlung der Zeit-Transition, Repräsentationsformen und Reihenfolge der diskreten Transitionen, On-the-fly-Analyse sowie Behandlung von inaktiven Uhren.

Kapitel 4

Modellierung und Verifikation hybrider Systeme

Für die Modellierung und Verifikation hybrider Systeme ist der bisher genutzte Formalismus nicht ausdrucksstark genug. Deshalb wird in diesem Kapitel der Modellierungsformalismus Cottbus Timed Automata so erweitert, daß ein Modul auch einen hybriden Automaten enthalten kann und die Konzepte der Komposition und Instanziierung erhalten bleiben. Diese Automatenklasse ist mächtiger als Timed Automata, und das Erreichbarkeitsproblem ist im allgemeinen nicht entscheidbar. Für viele interessante Modelle, die auf hybriden Automaten basieren, und für einige Teilklassen der hybriden Automaten terminiert die Erreichbarkeitsanalyse. Deshalb werden in diesem Kapitel die wichtigsten Automatentypen vorgestellt und Entscheidbarkeitsresultate angegeben.

Bei der Definition der hybriden Automaten und deren Semantik wird von der Referenzliteratur abgewichen. Die herkömmliche Semantik verbietet einige der intuitiv erwarteten Verhaltensweisen. Außerdem ist die Standardsemantik nicht kompositionell, d. h. die Semantik eines Systems ist nicht aus der Semantik der Bestandteile ableitbar. Zur Lösung dieses Problems wird neben der geänderten Standardsemantik eine kompositionelle Update-Semantik eingeführt. An einem Beispiel wird die Modellierung begleitend illustriert.

Für die algorithmische Analyse hybrider Automaten lassen sich nicht alle für Timed Automata gültigen Verfahren anwenden. Um auch für hybride Modelle eine Erreichbarkeitsanalyse bereitstellen zu können, wird eine auf Polyedern beruhende Repräsentation verwendet. Eine solche Polyeder-Repräsentation wurde von Halbwachs vorgeschlagen [Hal93] und für hybride Automaten bereits im Werkzeug HyTech verwendet [HHWT95b]. Als Grundlage für die Darstellung jeweils konvexer Mengen von Variablenbelegungen dient die Double Description Method [MRTT53].

4.1 Modellierung

Die Modellierung von Realzeit-Systemen machte es erforderlich, von zeitlosen finiten Automaten zu Formalismen überzugehen, die eine explizite Modellierung von Zeitaspek-

ten ermöglichen. Bei der Klasse der hybriden Systeme kommt hinzu, daß nicht nur die Zeit modelliert werden muß, sondern zusätzlich noch Größen, deren Ableitung nach der Zeit ungleich eins ist. So soll es zum Beispiel möglich sein, zurückgelegte Wege mittels veränderlicher Geschwindigkeiten zu berechnen oder Temperaturveränderungen in einem System aus der Heizleistung abzuleiten.

Der Übergang von Timed Automata zu hybriden Automaten erfolgt, indem als Variablen nicht nur Uhren, sondern auch andere analoge Variablen erlaubt sind. Bei analogen Variablen wird für jeden Zustand die Ableitung durch ein Prädikat definiert, nach dem sich der Wert der analogen Variablen beim Vergehen von Zeit verändert. Dadurch können veränderliche Größen, deren Ableitung nach der Zeit stückweise konstant ist oder in einem festen Intervall liegt, direkt modelliert werden. In den Zuständen sind also Invarianten und Ableitungsdefinitionen enthalten. Die Zustandswechsel werden durch bedingte Zustandsübergänge mit Zuweisungen angegeben. Zustandsübergänge können wie bei Timed Automata durch Synchronisationsmarken zur Kommunikation mit parallellaufenden Automaten markiert werden.

Nach der Einführung der mathematischen Notationen und Begriffe und einer kurzen Illustration am Beispiel wird zunächst der hybride Automat definiert. Wichtige Teilklassen der hybriden Automaten werden vorgestellt und Entscheidbarkeitsresultate referiert. Es folgt eine Anpassung der Konzepte zur modularen Modellstrukturierung. Die kontinuierliche Standard-Semantik des zugrundeliegenden Automatenmodells ist ähnlich der des hybriden Automaten von Henzinger. Darüberhinaus wird eine kompositionelle Semantik eingeführt. Für die Erreichbarkeitsanalyse werden zunächst einige Algorithmen betrachtet und dann entsprechende Datenstrukturen vorgestellt.

4.1.1 Mathematische Notationen und Begriffe

Zunächst werden analog zu Abschnitt 2.1 die Begriffe Variablenbedingung und Variablenbelegung definiert. Variablenbedingungen werden benutzt zur Definition von Invarianten und Ableitungen der Zuständen sowie von Wächtern und Wertveränderungen der Transitionen von hybriden Automaten. Variablenbedingungen sind das syntaktische Element zur Definition von Mengen von Variablenbelegungen. Variablenbedingungen legen die möglichen Werte der Variablen auf der syntaktischen Ebene fest; auf der semantischen Ebene wird der Begriff der Variablenbelegungen benutzt.

Definition 4.1 *Sei $X = \{x_1, \ldots, x_n\}$ eine endliche Menge von Variablen. Dann hat ein* **linearer Term** *über X die Form $c_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n$ für $x_i \in X$, $c_0, c_i \in \mathbb{Z}, 1 \leq i \leq n$. Aus zwei linearen Termen kann eine lineare Ungleichung durch einen Relationsoperator zusammengesetzt werden.* **Variablenbedingungen** *entstehen aus den linearen Ungleichungen durch die Operatoren Konjunktion und Disjunktion, oder aus den Booleschen Konstanten $true$ und $false$. Formal wird die Menge $\Phi(X)$ aller Variablenbedingungen über X durch die folgende Grammatik erzeugt:*

$$\begin{array}{rcl} \tau & := & c_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n \\ \varphi & := & \tau \sim 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid true \mid false \end{array}$$

mit $x_i \in X$, $c_0, c_i \in \mathbb{Z}, 1 \leq i \leq n$ *und* $\sim \in \{=, \leq, \geq, <, >\}$.

Anmerkung: Kommt eine Variable x_i in einer Ungleichung nicht vor, so wird dies so interpretiert, als sei ihr Koeffizient c_i auf 0 gesetzt, d. h. der Wert der Variable ist beliebig. Es sind auch Ausdrücke der Form $\tau \sim \tau$ möglich, diese sind jedoch durch Umstellen normalisierbar und müssen nicht extra behandelt werden.

Definition 4.2 *Sei X eine Menge von Variablen. Dann ist eine* **Variablenbelegung** v **von** X *eine totale Funktion $v : X \to \mathbb{R}$ von X in die Menge der reellen Zahlen $\mathbb{R}$. $Val(X)$ ist die Menge aller Variablenbelegungen von X.*

Durch die Variablenbedingungen werden Variablenbelegungen definiert; zu jeder Variablenbedingung kann durch die Anwendung der folgenden Abbildung die entsprechende Menge von Variablenbelegungen gefunden werden.

Definition 4.3 *Sei $X = \{x_1, \ldots, x_n\}$ eine endliche Menge von Variablen. Dann ist die* **Semantik einer Variablenbedingung** *gegeben durch die Interpretationsfunktion $[\![.]\!] : \Phi(X) \to 2^{Val(X)}$, die wie folgt induktiv definiert wird:*

$$
\begin{array}{rcll}
[\![c_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n \sim 0]\!] & & & \\
\qquad := & \left\{ v \in Val(X) \;\middle|\; c_0 + \sum_{i=1}^{n} c_i \cdot v(x_i) \sim 0 \right\} & & \textit{(Ungleichung)} \\
[\![\varphi \wedge \varphi']\!] & := & [\![\varphi]\!] \cap [\![\varphi']\!] & \textit{(Durchschnitt)} \\
[\![\varphi \vee \varphi']\!] & := & [\![\varphi]\!] \cup [\![\varphi']\!] & \textit{(Vereinigung)} \\
[\![true]\!] & := & Val(X) & \textit{(Volle Menge)} \\
[\![false]\!] & := & \emptyset & \textit{(Leere Menge)}
\end{array}
$$

mit $x_i \in X$, $c_0, c_i \in \mathbb{Z}, 1 \leq i \leq |X|$ und $\sim \in \{=, \leq, \geq, <, >\}$, d. h. φ wird interpretiert als die Menge aller Variablenbelegungen über X, die das logische Prädikat φ erfüllen. Bei festgelegter Reihenfolge der Variablen innerhalb X wird für $v \in Val(X)$ auch die Vektor-Schreibweise $\bar{v} = (v(x_1), v(x_2), \ldots, v(x_n))$ verwendet.

Für die formale Definition der Wertveränderung bei Zustandsübergängen in einem hybriden Automat ist neben der Variablenbelegung ein weiterer Begriff notwendig: Im Unterschied zu einer Variablenbelegung werden durch ein Update nicht für alle, sondern nur für bestimmte Variablen Werte definiert. Syntaktisch wird ein Update durch ein logisches Prädikat über Variablen aus X definiert, wobei nur die Variablen aus $Y \subseteq X$ im Prädikat vorkommen. Intuitiv sind die Variablen aus $X \setminus Y$ diejenigen, deren Wert bei einer Wertveränderung nicht berührt werden soll.

Definition 4.4 *Sei X eine Menge von Variablen. Dann ist ein* **Update** u **von** X *eine partielle Funktion $u : X \multimap \mathbb{R}$ von einer Teilmenge $Y \subseteq X$ in die Menge der reellen Zahlen $\mathbb{R}$. $Updt(X)$ ist die Menge aller Updates von X.*

Anmerkung: Die Menge von Updates, die die Werte aller Variablen unverändert läßt, ist die Einermenge von Funktionen $\{\emptyset\}$. Diese Menge enthält nur eine Funktion, welche für keine Variable definiert ist, d. h. alle Updates mit leerem Definitionsbereich. Die widersprüchliche Menge von Updates $\{\}$ ist die leere Menge, d. h. es existiert kein Update, der Definitionsbereich spielt keine Rolle.

Bei hybriden Automaten treten mehrere Arten von Variablenbedingungen auf. Für Wächter, Invarianten und Ableitungen werden allgemeine Variablenbedingungen nach Definition 4.1 verwendet, wobei bei der Ableitungsdefinition die Variablenwerte als zulässige Ableitungen für die entsprechende Variable interpretiert werden. Bei den Variablenbedingungen für die Wertveränderungen kommen Variablen für die Definition der Werte nach der Wertveränderung hinzu.

Variablenbedingungen über $X \cup X'$ werden Wertveränderungen genannt. Für eine Wertveränderung (Syntax) kann mit der folgenden Abbildung eine Zuordnung einer Menge von Updates von X zu einer Variablenbelegung von X (Semantik) gefunden werden. Für eine Menge X von Variablen ist X' die Menge von Variablen, die genau für jede Variable $x \in X$ eine Variable x' enthält ($X \cap X' = \emptyset$).

Definition 4.5 *Sei $X = \{x_1, \ldots, x_n\}$ eine endlich Menge von Variablen. Dann ist die* **Update-Semantik einer Variablenbedingung** *gegeben durch die Interpretationsfunktion $[\![.]\!]_U : \Phi(X \cup X') \to \left(Val(X) \to 2^{Updt(X)}\right)$, die wie folgt induktiv definiert ist:*

$$[\![c_0 + c_1 \cdot x_1 + \cdots + c_n \cdot x_n + c_{n+1} \cdot x'_1 + \cdots + c_{2n} \cdot x'_n \sim 0]\!]_U(v)$$
$$:= \left\{ u \in Updt(X) \;\middle|\; \begin{array}{ll} & c_0 + \sum\limits_{x_i \in X} c_i \cdot v(x_i) + \sum\limits_{x_i \in dom(u)} c_{n+i} \cdot u(x'_i) \sim 0 \\ \wedge & \forall x_i \in X : x_i \in dom(u) \Leftrightarrow c_{n+i} \neq 0 \end{array} \right\}$$

$$[\![\varphi \wedge \varphi']\!]_U(v) := \left\{ u : X \multimap \mathbb{R} \;\middle|\; \begin{array}{ll} & \exists u_1 \in [\![\varphi]\!]_U(v) : \; u|_{\mathrm{dom}(u_1)} = u_1 \\ \wedge & \exists u_2 \in [\![\varphi']\!]_U(v) : \; u|_{\mathrm{dom}(u_2)} = u_2 \\ \wedge & \mathrm{dom}(u) = \mathrm{dom}(u_1) \cup \mathrm{dom}(u_2) \end{array} \right\}$$

$$[\![\varphi \vee \varphi']\!]_U(v) := [\![\varphi]\!]_U(v) \cup [\![\varphi']\!]_U(v) \qquad \textit{(Vereinigung)}$$

$$[\![true]\!]_U(v) := \{\emptyset\} \qquad \textit{(Keine Veränderung)}$$

$$[\![false]\!]_U(v) := \emptyset \qquad \textit{(Widersprüchliche Update-Menge)}$$

mit $x_i \in X$, $c_0, c_i \in \mathbb{Z}, 1 \leq i \leq |X|$ und $\sim \in \{=, \leq, \geq, <, >\}$. φ wird interpretiert als die Variablenbelegungen von X zugeordneten Mengen aller Updates von X', die das logische Prädikat φ erfüllen und nicht mehr Variablen aus X' im Definitionsbereich haben als in φ vorkommen.

4.1.2 Beispiel: Bahnschranke

Um einen Eindruck von den hybriden Automaten zu vermitteln, wird an dieser Stelle ein Modell einer Schranke am Bahnübergang vorgestellt. Die Erklärung eines kompletten Schrankensteuerungsmodells folgt in Abschnitt 4.1.7.

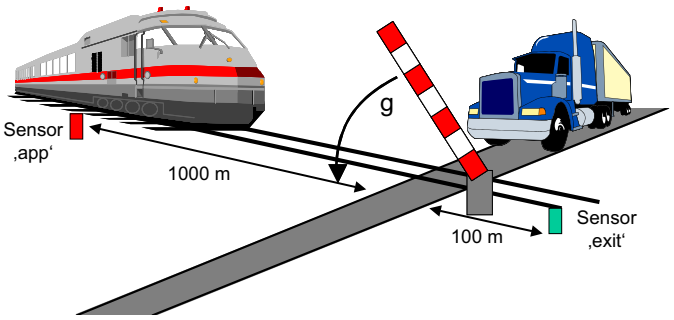


Abbildung 4.1: Bahnübergang mit zu modellierenden Größen.

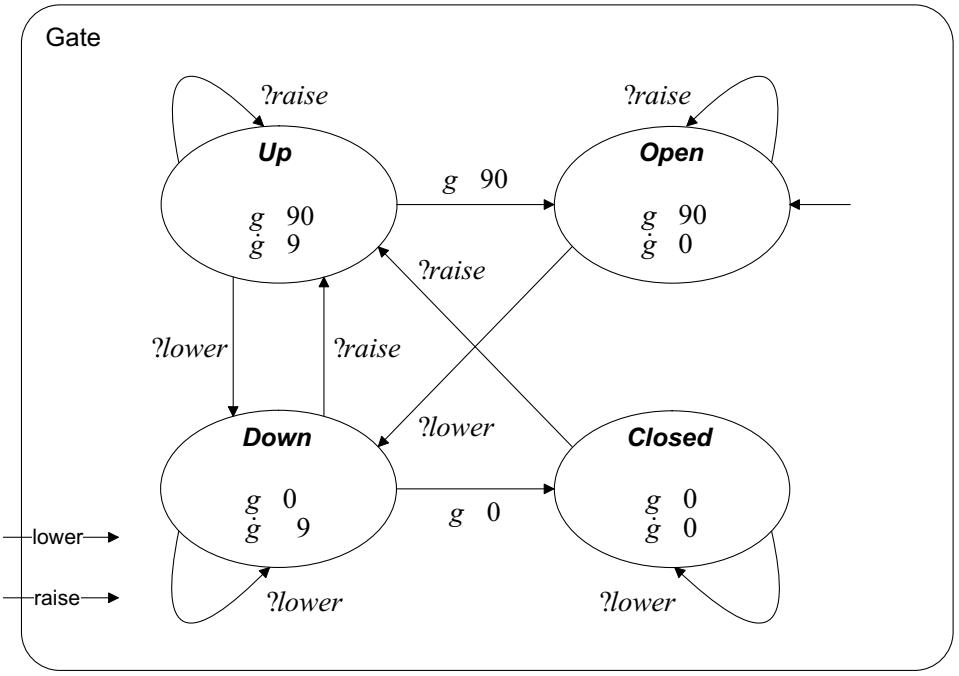


Abbildung 4.2: Hybrides Modell einer Bahnschranke. $lower$ und $raise$ sind Synchronisationsmarken, g ist eine lokale analoge Variable.

Die für den Bahnübergang wesentliche Eigenschaft der Schranke ist ihr Öffnungswinkel, in Abbildung 4.1 mit g bezeichnet. Im Automat innerhalb des Moduls Gate (siehe Abbildung 4.2) entspricht dieser Größe die analoge Variable g. Diese Variable ist eine lokale Variable, weil kein Automat außerhalb des Moduls Gate auf diese Variable zugreifen darf. Der Automat kennt zwei Synchronisationsmarken, $lower$ und $raise$, auf die er reagieren muß (daher werden diese als Eingabe-Marken deklariert).

Der Automat startet im Zustand *Open* ($g = 90$). Erreicht den Automaten das Signal $lower$, so erfolgt ein Übergang in den Zustand *Down*, der die Situation modelliert, daß sich die Schranke gerade schließt, jedoch noch nicht vollständig geschlossen ist. Die

Winkelgeschwindigkeit, mit der sich die Schranke schließt, wird durch die Ableitung von g definiert (bezeichnet mit $\dot{g}$). Erreicht die Schranke den Winkel 0 Grad, dann wechselt der Automat zum Zustand *Closed*. In diesem Zustand wartet der Automat auf das Signal $raise$, das einen Übergang zu Zustand *Up* erzwingt und somit den Öffnungsprozeß startet. Beim Erreichen des Winkels 90 Grad geht der Automat zum Ausgangszustand *Open* über und der Prozeß kann für den nächsten Zug von neuem beginnen.

4.1.3 Hybride Automaten

Für die Modellierung hybrider Systeme wird in dieser Arbeit der Formalismus der hybriden Automaten verwendet. Die Theorie der hybriden Automaten wurde von Henzinger untersucht [Hen96]. Im Gegensatz zu den Timed Automata besitzen die hybriden Automaten komplexere Zustandsübergänge. Es ist nicht nur ein Rücksetzen von Variablen erlaubt, sondern die Werte der Variablen nach dem Zustandsübergang werden durch ein logisches Prädikat in Abhängigkeit der Variablenbelegung vor dem Übergang definiert. Die von Henzinger angegebene Definition von hybriden Automaten und deren Semantik führt dazu, daß die Produktautomatenbildung nicht kompositionell ist (siehe Abschnitt 4.2.1). Dies wird hier vermieden durch die Verwendung von Updates für die Wertveränderung der Variablen.

Definition 4.6 *Ein* **linearer hybrider Automat (kurz: hybrider Automat[1])** *ist ein Tupel* $\mathcal{H} = (L, X, Q^0, \Sigma, I, D, E)$ *mit den folgenden Komponenten:*

- L *ist eine endliche Menge von* **Zuständen**.
- X *ist eine endliche Menge von* **analogen Variablen**.
- $Q^0 \subseteq (L \times \Phi(X))$ *ist eine Menge von* **Initialbedingungen**.
- Σ*, mit* $\Sigma \cap \mathbb{R} = \emptyset$*, ist eine endliche Menge von* **Synchronisationsmarken**.
- $I : L \rightarrow \Phi(X)$ *ist eine totale Funktion, die jedem Zustand in* L *eine* **Invariante** *aus* $\Phi(X)$ *zuordnet.*
- $D : L \rightarrow \Phi(X)$ *ist eine totale Funktion, die jedem Zustand in* L *eine* **Ableitungsbedingung** *aus* $\Phi(X)$ *zuordnet.*
- $E \subseteq L \times \Sigma \times \Phi(X \cup X') \times L$ *ist eine Menge von* **Zustandsübergängen**.

Zusätzlich werden für einen hybriden Automaten abkürzende Schreibweisen eingeführt.

Definition 4.7 *Sei* $\mathcal{H} = (L, X, Q^0, \Sigma, I, D, E)$ *ein hybrider Automat und* $e = (l, \sigma, \mu, l') \in E$ *ein Zustandsübergang. Dann gilt:*

[1] In dieser Arbeit wird im allgemeinen unter dem Begriff *hybrider Automat* immer ein *linearer hybrider Automat* verstanden. Nur in Abschnitt 4.1.5 werden für einen Überblick auch andere hybride Automaten betrachtet.

- $\mathcal{H}.guard : E \to 2^{Val(X)}$ *ist die totale Funktion, die jedem Zustandsübergang die Menge von Variablenbelegungen von* X *für seinen* **Wächter** *zuordnet:*
  $\mathcal{H}.guard(e) = \{v \in Val(X) \mid [\![\mu]\!]_U(v) \neq \emptyset\}$.

- $\mathcal{H}.sync : E \to \Sigma$ *ist die totale Funktion, die jedem Zustandsübergang seine* **Synchronisationsmarke** *zuordnet:*
  $\mathcal{H}.sync(e) = \sigma$.

- $\mathcal{H}.update : E \to \left(Val(X) \to 2^{Updt(X)}\right)$ *ist die totale Funktion, die jedem Zustandsübergang die Update-Semantik für seine* **Wertveränderung** *zuordnet:*
  $\mathcal{H}.update(e) = [\![\mu]\!]_U$.

Anmerkung: Einige Komponenten des hybriden Automaten werden im folgenden verbal erläutert.

- Die durch die *Ableitungsbedingung* $D(l)$ festgelegten Werte werden als zulässige Ableitungen nach der Zeit für die Variablen interpretiert, solange der Automat in dem Zustand l ist.

- Ein *Zustandsübergang* $e = (l, \sigma, \mu, l') \in E$ repräsentiert einen mit der Synchronisationsmarke σ markierten Übergang von Startzustand l zu Zielzustand l'. Der Wächter muß erfüllt sein, d. h. die Variablenbelegung vor dem Zustandsübergang muß Element der Menge $\mathcal{H}.guard(e)$ sein, damit der Übergang möglich ist. Der Übergang setzt die neuen Werte der Variablen entsprechend der Wertveränderung μ.

- Die Interpretation $[\![\mu]\!]_U$ einer Variablenbedingung μ ergibt eine Funktion, die zu einer Variablenbelegung eine Menge von Updates liefert. Diese Funktion definiert, wie sich die Werte der Variablen verändern. Für einen bestimmten Update $u \in [\![\mu]\!]_U(v)$ eines Zustandsübergangs e und für eine Variablenbelegung v werden die Variablen aus $X \setminus dom(u)$ nicht verändert. Bei Wertveränderungen repräsentieren die nicht gestrichenen Variablen die Werte des Funktionsarguments (der Variablenbelegung vor dem Zustandsübergang) und die gestrichenen Variablen repräsentieren die Werte der resultierenden Variablenbelegungen (nach dem Zustandsübergang). Im Definitionsbereich eines Updates treten nur solche Variablen auf, die durch das entsprechende Prädikat eingeschränkt werden.

4.1.4 Parallele Komposition

In diesem Abschnitt wird beschrieben, was es bedeutet, wenn zwei hybride Automaten parallel laufen und so ein Gesamtsystem bilden (vgl. [Hen96] für das allgemeine Prinzip). Die später in diesem Kapitel vorgestellte modulare Erweiterung des Formalismus macht es notwendig, mehrere hybride Automaten in dieser Weise zu kombinieren.

Definition 4.8 *Seien* $\mathcal{H}_1 = (L_1, X_1, Q_1^0, \Sigma_1, I_1, D_1, E_1)$ *und* $\mathcal{H}_2 = (L_2, X_2, Q_2^0, \Sigma_2, I_2, D_2, E_2)$ *zwei hybride Automaten. Dann ist der* **Produktautomat** $\mathcal{H}_1 \,||\, \mathcal{H}_2$

durch den hybriden Automaten $(L_1 \times L_2, X_1 \cup X_2, Q^0, \Sigma_1 \cup \Sigma_2, I, D, E)$ *gegeben, wobei* $I((l_1, l_2)) = I_1(l_1) \wedge I_2(l_2)$, $D((l_1, l_2)) = D_1(l_1) \wedge D_2(l_2)$ *und:*

- $((l_1, l_2), \varphi_1 \wedge \varphi_2) \in Q^0$ *gdw.*
  - $(l_1, \varphi_1) \in Q_1^0 \quad \wedge \quad (l_2, \varphi_2) \in Q_2^0$
- $((l_1, l_2), \sigma, \mu, (l_1', l_2')) \in E$ *gdw.*
  - $\sigma \in \mathcal{H}.\Sigma \cap \mathcal{H}'.\Sigma$ *und es existieren Zustandsübergänge* $e_1 = (l_1, \sigma, \mu_1, l_1') \in \mathcal{H}.E$ *und* $e_2 = (l_2, \sigma, \mu_2, l_2') \in \mathcal{H}'.E$*, so daß* $\mu = \mu_1 \wedge \mu_2$ *gilt, oder*
  - $\sigma \in \mathcal{H}.\Sigma \setminus \mathcal{H}'.\Sigma$, $l_2 = l_2'$ *und es existiert ein Zustandsübergang* $(l_1, \sigma, \mu, l_1') \in \mathcal{H}.E$*, oder*
  - $\sigma \in \mathcal{H}'.\Sigma \setminus \mathcal{H}.\Sigma$, $l_1 = l_1'$ *und es existiert ein Zustandsübergang* $(l_2, \sigma, \mu, l_2') \in \mathcal{H}'.E$.

Anmerkung: Die Zustandsübergänge einer parallelen Komposition resultieren aus je zwei Zustandsübergängen, die gemeinsam gleichzeitig ausgeführt werden. Unsynchronisierte Übergänge der beiden Automaten werden auch in der parallelen Komposition unsynchronisiert ausgeführt. Bei unsynchronisierten Übergängen eines der parallellaufenden Automaten ist der Übergang des Produktautomaten durch den Übergang dieses einen Automaten definiert.

Die parallele Komposition erlaubt alle Transformationen von Variablenbelegungen, die von $\mathcal{H}.update$ und $\mathcal{H}'.update$ erlaubt sind, d. h. es entstehen keine widersprüchliche Update-Mengen. Wenn eine Variable vom ausgewählten Update eines Übergangs nicht eingeschränkt wird, bleibt der Wert der Variablen unverändert.

Die parallele Komposition zweier hybrider Automaten ist wieder ein hybrider Automat:

Satz 4.1 *Seien* $\mathcal{H}$ *und* $\mathcal{H}'$ *zwei hybride Automaten. Dann ist* $\mathcal{H} \mid\mid \mathcal{H}'$ *auch ein hybrider Automat.*

(Parallele Komposition mehrerer hybrider Automaten.) Sei H eine nicht leere, endliche Menge von hybriden Automaten. Dann bezeichnet $\prod_{\mathcal{H} \in H} \mathcal{H}$ die parallele Komposition aller Elemente von H in einer bestimmten Ordnung.

4.1.5 Abgeleitete Automatenklassen

Das Erreichbarkeitsproblem für hybride Automaten ist nicht entscheidbar. Durch einige Einschränkungen der Ausdrucksstärke entstehen verschiedene spezielle Automatenklassen, für die eine algorithmische Analyse sinnvoll sein kann. Um einen Überblick über diese verschiedenen Klassen zu geben und zu vermitteln, wo genau die Grenze von entscheidbar zu unentscheidbar liegt, werden in diesem Abschnitt Definitionen verschiedener

Varianten des hybriden Automaten angegeben und entsprechende Entscheidbarkeitsresultate referiert.

Im letzten Abschnitt wurde bereits ein spezieller Typ hybrider Automaten eingeführt, der lineare hybride Automat. Dies ist darin begründet, daß für diesen Automat im Abschnitt 4.3 die Erreichbarkeitsanalyse behandelt wird.

Definition 4.9 *Der* **allgemeine hybride Automat** *entsteht aus dem linearen hybriden Automaten durch den Wegfall folgender Einschränkungen, so daß:*

1. *als Variablenbedingungen nicht nur lineare Ungleichungen erlaubt sind, d. h.* $\Phi(X)$ *ist die Menge beliebiger Prädikate über* X*, und insbesondere*

2. *für die Definition der Ableitung der Variablen in einem Zustand nicht nur Prädikate über die Ableitungen zugelassen sind, sondern Prädikate über die Variablen und deren Ableitungen, d. h.* $D : L \rightarrow \Phi(X \cup \dot{X})$ *mit* X *ist die Menge der Variablen und* $\dot{X}$ *ist die Menge ihrer Ableitungen.*

Somit sind Differentialgleichungen erster Ordnung erlaubt. Dies gibt dem Formalismus eine hohe Ausdrucksstärke, führt jedoch gleichzeitig dazu, daß Modelle dieses Formalismus kaum algorithmisch analysierbar sind.

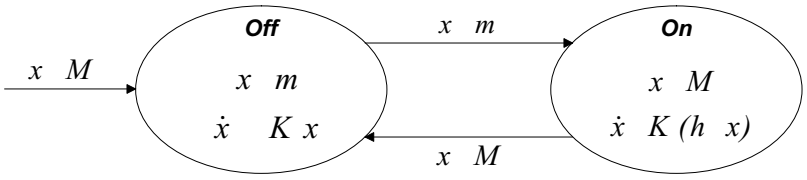


Abbildung 4.3: Modell einer Temperaturregelung.

Beispiel Temperatursteuerung. Um zu illustrieren, welche Art von Systemen mit einem allgemeinen hybriden Automaten modelliert werden kann, wird im folgenden das Beispiel einer Temperatursteuerung in einem Raum erklärt [ACH+95]. Abbildung 4.3 zeigt das Automatenmodell mit der analogen Variable x. Die Temperatur eines Raumes wird durch ein Thermostat gesteuert, wobei der Thermostat ständig die Temperatur mißt und in Abhängigkeit davon eine Heizung ein- bzw. ausschaltet. Die Temperatur läßt sich dabei durch Differentialgleichungen bestimmen. Bezeichne x die Temperatur zu einem bestimmten Zeitpunkt t und ϑ die Anfangstemperatur. K sei eine Raumkonstante, die den Wärmeverlust wiedergibt, und h sei eine Konstante, die von der Leistung der Heizung abhängt. Dann gilt $x(t) = \vartheta e^{-Kt}$, falls die Heizung ausgeschaltet ist, und $x(t) = \vartheta e^{-Kt} + h(1 - e^{-Kt})$, falls die Heizung eingeschaltet ist. Die Temperatursteuerung soll die Temperatur zwischen einer Minimaltemperatur m und einer Maximaltemperatur M regeln. Bei dem in Abbildung 4.3 gezeigten Automaten steht der Zustand *Off* für ausgeschaltete Heizung und *On* für eingeschaltete Heizung. In den Zuständen befinden sich Differentialgleichungen, durch die die Wertveränderung der Variablen x bestimmt ist: $\dot{x}(t) = -Kx$ bzw. $\dot{x}(t) = K(h - x)$. Dieser Automat ist kein linearer hybrider Automat.

Ein hybrider Automat ist **rectangular**, wenn alle Variablen voneinander unabhängig und die Ableitungsbedingungen der Variablen von den Zuständen des Automaten unabhängig sind. Für jede Variable wird die erste Ableitung nach der Zeit durch die Angabe eines Intervalls möglicher Werte definiert. Dieses Intervall ändert sich nicht bei Zustandsübergängen. Bei jedem Zustandsübergang wird der neue Wert einer Variablen entweder unverändert belassen oder nichtdeterministisch auf einen Wert aus einem gegebenen Intervall gesetzt. Dabei sind die Variablen voneinander entkoppelt, d. h. der Wert einer Variablen hängt nicht direkt vom Wert anderer Variablen oder deren Ableitung ab. Rectangular Automata wurden bereits eingehend untersucht [PV94, HKPV98]. Die folgende Definition orientiert sich an der von Henzinger [Hen96].

Definition 4.10 *Sei $X = \{x_1, \ldots, x_n\}$ eine endliche Menge von Variablen. Dann ist ein* **Rectangle der Dimension** *n ein Prädikat aus der Menge $\Phi_R(X)$, die von der folgenden Grammatik erzeugt wird:*

$$\varphi \ := \ x \geq c_1 \wedge x \leq c_2 \mid \varphi \wedge \varphi \mid \mathit{true} \mid \mathit{false}$$

mit $x \in X$ und $c_1, c_2 \in \mathbb{Q} \cup \infty$. Ein Rectangle $\varphi \in \Phi_R(X)$ ist **beschränkt**, *falls $c_1, c_2 \in \mathbb{Q}$, und* **singulär**, *falls $c_1, c_2 \in \mathbb{Q}$ und $c_1 = c_2$.*

Anmerkung: Sei φ ein Rectangle der Dimension n. Dann kann φ als n-dimensionaler Quader interpretiert werden: $[\![\varphi]\!] = \prod_{i=1}^{n} \mathbb{I}_i$ ist das Kreuzprodukt von n Intervallen $\mathbb{I}_i \subseteq \mathbb{R}$ auf einem reellen Strahl mit Endpunkten aus $\mathbb{Q} \cup \infty$. Ist φ beschränkt, dann gilt für alle i mit $1 \leq i \leq n$: das Intervall $\mathbb{I}_i$ ist beschränkt. Ist φ singulär, dann gilt für alle i mit $1 \leq i \leq n$: $\mathbb{I}_i = \{c_i\}$ für ein $c_i \in \mathbb{Q}$.

Definition 4.11 *Ein hybrider Automat $\mathcal{H}$ ist ein* **Rectangular Automaton**, *falls die folgenden drei Bedingungen gelten. Bezeichne $\mathcal{H}.Q^0(l)$ die initiale Variablenbedingung für einen Zustand l mit $\mathcal{H}.Q^0(l) = \{\varphi \mid (l, \varphi) \in \mathcal{H}.Q^0\}$ und sei $\mathcal{H}.X = \{x_1, \ldots, x_n\}$.*

1. *Für jeden Zustand $l \in \mathcal{H}.L$ gilt: Die Bedingung für die Invariante $\mathcal{H}.I(l)$ ist ein Rectangle der Dimension n. Die Initialbedingung $\mathcal{H}.Q^0(l) \in \Phi_R(\mathcal{H}.X)$ ist ein beschränktes Rectangle der Dimension n.*

2. *Es existiert eine Ableitungsbedingung $\varphi \in \Phi_R(\mathcal{H}.X)$, so daß φ ein beschränktes Rectangle der Dimension n ist und für alle $l \in \mathcal{H}.L$ gilt: $\mathcal{H}.D(l) = \varphi$.*

3. *Für jeden Zustandsübergang $(l, \sigma, \mu, l') \in \mathcal{H}.E$ gilt: Die Bedingung für die Wertveränderung $\mu \in \Phi_R(\mathcal{H}.X \cup \mathcal{H}.X')$ ist ein Rectangle der Dimension n. Für jede in μ vorkommende Variable $x' \in \mathcal{H}.X'$ muß die Bedingung in μ die Form $x \geq c_1 \wedge x \leq c_2$ mit $c_1, c_2 \in \mathbb{Q}$ haben.*

Durch die Beschränkung der Variablenbelegungen auf bestimmte Intervalle ergeben sich sofort weitere Spezialklassen der Rectangular Automata.

Definition 4.12 *Ein* **Singulärautomat** $\mathcal{H}$ *ist ein Rectangular Automaton, für den eine Ableitungsbedingung* φ *existiert, so daß* φ *ein singuläres Rectangle der Dimension* $|\mathcal{H}.X|$ *ist und für alle* $l \in \mathcal{H}.L$ *gilt:* $\mathcal{H}.D(l) = \varphi$.

Definition 4.13 *Ein* **Multi-Rectangular Automaton** $\mathcal{H}$ *entsteht aus einem Rectangular Automaton, indem in Definition 4.11 der Punkt 2 so verändert wird, daß für verschiedene Zustände* l *und* l' *auch verschiedene beschränkte Rectangles* $\mathcal{H}.D(l)$ *und* $\mathcal{H}.D(l')$ *für die Ableitungen erlaubt sind. Ein* **Multi-Singulärautomat** $\mathcal{H}$ *ist ein Multi-Rectangular Automaton, bei dem die beschränkten Rectangles für die Ableitungen singulär sind.*

Definition 4.14 *Ein* **Timed Automaton** $\mathcal{H}$ *ist ein Singulärautomat, bei dem folgende zwei Einschränkungen gelten:*

- *Die Ableitungsbedingung ist eine Konjunktion von Bedingungen der Form* $x = 1$ *für alle Variablen* $x \in \mathcal{H}.X$. *Diese Bedingung wird bei Timed Automata nicht explizit geschrieben.*
- *Variablen können nur auf den Wert 0 zurückgesetzt werden, d. h. alle Variablen* $x \in \mathcal{H}.X$ *kommen in der Wertveränderung eines Zustandsübergangs* $e \in \mathcal{H}.E$ *gestrichen nur in der Form* $x' = 0$ *vor.*

Definition 4.15 *Ein* **Stoppuhr-Automat** $\mathcal{H}$ *ist ein Multi-Singulärautomat, bei dem die Ableitungsbedingung eine Konjunktion von Bedingungen der Form* $x = c$ *für alle Variablen* $x \in \mathcal{H}.X$ *mit* $c \in \{0, 1\}$ *ist.*

Variablentypen. Nach den in den vorausgehenden Definitionen vorgestellten Kriterien zur Klassifizierung der Automaten können auch die sogenannten analogen Variablen eines hybriden Automaten klassifiziert werden (siehe Abbildung 4.4 auf der nächsten Seite).

Nicht-lineare Variablen: Die zeitabhängige Wertveränderung dieser Variablen wird durch eine Differentialgleichung erster Ordnung, die aus linearen Termen zusammengesetzt ist, bestimmt.

Lineare Variablen: Dieser Variablentyp wird vom linearen hybriden Automaten verwendet. Die Ableitungen dürfen nur durch Ungleichungen über die Ableitungen der Variablen und Boolesche Kombination davon spezifiziert werden.

Stoppuhren: Eine Stoppuhr ist eine Variable, deren Ableitung nach der Zeit in allen Zuständen entweder 1 oder 0 ist. Für eine Stoppuhr x gilt für alle Zustände aus $\mathcal{H}.L$: Die Ableitungsbedingung für x in $\mathcal{H}.D(l)$ hat die Form $x = 0$ oder die Form $x = 1$.

Uhren: Eine Uhr ist eine Variable, deren Ableitung nach der Zeit in allen Zuständen 1 ist. Für eine Uhr x gilt für alle Zustände aus $\mathcal{H}.L$: Die Ableitungsbedingung für x in $\mathcal{H}.D(l)$ hat die Form $x = 1$.

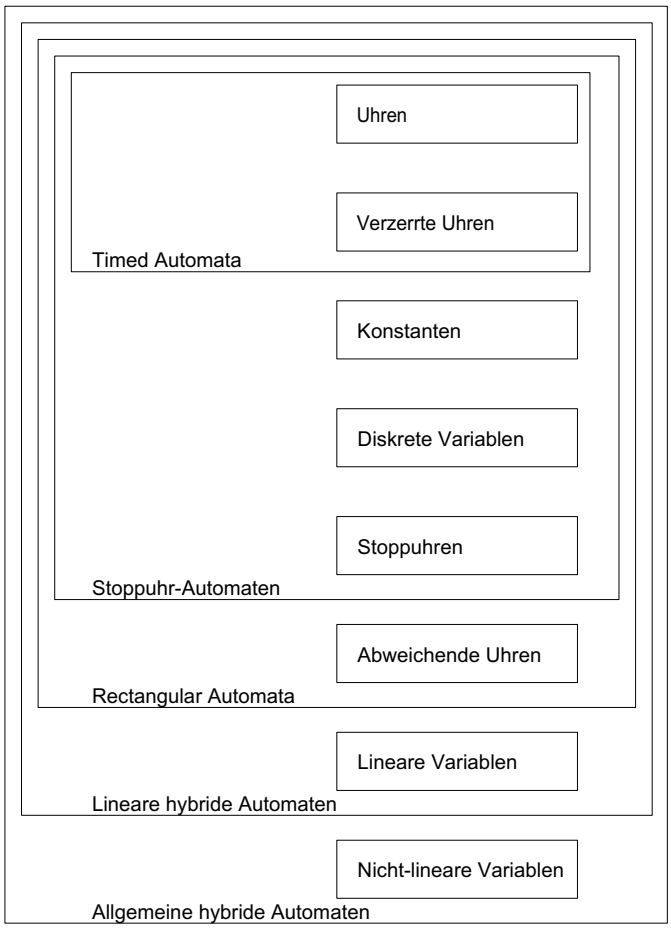


Abbildung 4.4: Variablentypen und Automatenklassen, in denen sie erlaubt sind.

Uhren mit Abweichung: Soll eine Uhr mit einer Abweichung ϵ (engl. Drifting Clock) mit $\epsilon \in \mathbb{Q}_{\geq 0}$ modelliert werden, so ist dies durch Angabe eines entsprechenden Intervalls für die Ableitung möglich. Für eine Uhr x mit Abweichung ϵ gilt für alle Zustände aus $\mathcal{H}.L$: Die Ableitungsbedingung für x in $\mathcal{H}.D(l)$ hat die Form $x \leq c_{max} \wedge x \geq c_{min}$ mit $c_{max} = 1 + \epsilon, c_{min} = 1 - \epsilon, \epsilon \in \mathbb{Q}_{\geq 0}$.

Uhren mit Verzerrung: Soll eine Uhr mit einer Verzerrung k (engl. Skewed Clock) mit $k \in \mathbb{Z} \setminus \{0\}$ modelliert werden, so ist dies durch Angabe einer ganzzahligen Ableitung ungleich der Null möglich: Für eine Uhr x mit Verzerrung k gilt für alle Zustände aus $\mathcal{H}.L$: Die Ableitungsbedingung für x in $\mathcal{H}.D(l)$ hat die Form $x = k$ mit $k \in \mathbb{Z} \setminus \{0\}$. Wird dieser Typ in der Klasse der Rectangular Automata verwendet, so ist dieser Variablentyp nicht mächtiger als eine normale Uhr.

Diskrete Variablen: Bei einer diskreten Variable ist die Ableitung nach der Zeit immer 0. Für eine diskrete Variable[2] x gilt für alle Zustände aus $\mathcal{H}.L$: Die Ableitungsbedingung für x in $\mathcal{H}.D(l)$ hat die Form $x = 0$.

Unbekannte Konstanten: Implizite Konstanten, d. h. Bezeichner, deren Wert eine Konstante ist, werden normalerweise wie explizite Konstanten gehandhabt, der Wert wird anstelle des Bezeichners direkt eingesetzt. Soll jedoch eine Konstante modelliert werden, deren Wert nicht bekannt ist (auch Parameter genannt [ACH+95]), so ist dies dadurch möglich, daß die Ableitung in jedem Zustand den Wert 0 hat und der Wert der Variable bei keinem Zustandsübergang verändert wird. Für eine unbekannte Konstante x gilt für alle Zustände aus $\mathcal{H}.L$: Die Ableitungsbedingung für x in $\mathcal{H}.D(l)$ hat die Form $x = 0$ und x tritt in keiner Wertveränderung eines Zustandsübergangs gestrichen auf.

Betrachtungen zur Mächtigkeit hybrider Automaten. Bei der Verifikation von Sicherheitseigenschaften eines Modells ist das Erreichbarkeitsproblem das elementare Problem überhaupt. Die Entscheidung, welcher Formalismus zur Modellierung genutzt wird, ist immer ein Balanceakt zwischen Ausdrucksstärke des Formalismus und Effizienz der algorithmischen Analyse. Daher werden im folgenden einige bekannte Ergebnisse referiert.

Definition 4.16 *Das* **Erreichbarkeitsproblem** *für eine Klasse von hybriden Automaten ist die Frage danach, ob für einen gegebenen hybriden Automaten $\mathcal{H}$ aus dieser Klasse ausgehend von einer initialen Konfiguration (l_0, v_0) eine bestimmte Konfiguration (l_F, v_F) erreicht werden kann.*

Zunächst werden zwei *Entscheidbarkeitsresultate* wiedergegeben.

Satz 4.2 (Alur, Courcoubetis, Dill, 1993) *Das Erreichbarkeitsproblem für Timed Automata ist entscheidbar.*

Beweis: Der Beweis wurde in [ACD93] geführt. □

Satz 4.3 (Alur, Courcoubetis, Halbwachs, Henzinger et al., 1995) *Das Erreichbarkeitsproblem für Rectangular Automata, die nur Uhren oder verzerrte Uhren enthalten, ist entscheidbar.*

Beweis: Es wird ein konstruktiver Beweis in Anlehnung an Alur et al. [ACH+95] skizziert: Sei $\mathcal{A}$ ein Rectangular Automaton, eingeschränkt auf Variablen vom Typ *Uhr mit Verzerrung*. Dann kann $\mathcal{A}$ in folgender Weise in einen bzgl. der erreichbaren Zustände äquivalenten Timed Automaton $ta(\mathcal{A})$ transformiert werden:

- Die Ableitungen aller verzerrten Uhren werden auf 1 gesetzt.

[2] Ist der Wertebereich einer solchen diskreten Variable endlich, so gibt es effizientere Möglichkeiten der Repräsentation, z. B. Gleichbehandlung mit Automaten.

- Alle Vorkommen einer Uhr x mit Verzerrung k_x in Zustandsinvarianten, Initialbedingungen und Wächter werden ersetzt durch $k_x \cdot x$.

□

Es folgen zwei *Unentscheidbarkeitsresultate*. Die Entscheidbarkeit des Erreichbarkeitsproblems ist von einigen wesentlichen Eigenschaften der Automatentypen abhängig:

1. Die Variablen müssen voneinander unabhängig sein. (Dies ist z. B. in der Klasse der Rectangular Automata gegeben.)

2. In den Ableitungen aller Variablen in verschiedenen Zuständen dürfen nicht unterschiedliche von 0 verschiedene Werte verwendet werden. (Dies ist dann erfüllt, wenn nur verzerrte Uhren mit der gleichen Ableitung vorkommen, z. B. bei Timed Automata.)

3. Für eine Uhr darf nicht zusätzlich die Ableitung 0 erlaubt sein. (Dies ist dann der Fall, wenn keine Stoppuhren vorhanden sind.)

Wird eine Klasse von Automaten, für die das Erreichbarkeitsproblem entscheidbar ist, so verallgemeinert, daß die dritte Bedingung verletzt wird, so führt dies direkt zur Nichtentscheidbarkeit des Erreichbarkeitsproblems (Satz 4.4). Das Erreichbarkeitsproblem kann entscheidbar bleiben, wenn nur die erste oder die zweite Bedingung verletzt wird (Satz 4.3, der die zweite Bedingung verletzt, und Timed Automata in [AD94], bei denen die erste Bedingung nicht erfüllt ist durch die Festlegung von konstanten Differenzen zwischen je zwei Uhren). Werden die erste und die zweite Bedingung nicht erfüllt, ist das Erreichbarkeitsproblem nicht entscheidbar (Satz 4.5).

Satz 4.4 (Alur et al., 1993) *Das Erreichbarkeitsproblem für Multi-Singulärautomaten ist unentscheidbar.*

Beweis: Der Satz wurde in [ACHH93] und [Čer93] bewiesen. Ein Beispiel für die Klasse der Multi-Singulärautomaten ist der Stoppuhr-Automat. □

Bereits die Kombination von Uhren mit einer einzigen Stoppuhr oder einer einzigen unbekannten Konstanten führt zur Unentscheidbarkeit des Erreichbarkeitsproblems [HKPV98].

Satz 4.5 (Alur, Courcoubetis, Halbwachs, Henzinger, et al., 1995) *Das Erreichbarkeitsproblem für lineare hybride Automaten, eingeschränkt auf verzerrte Uhren mit zwei verschiedenen Ableitungen, ist unentscheidbar.*

Beweis: Der Satz folgt aus der Unentscheidbarkeit des Halteproblems für nichtdeterministische Zählermaschinen mit 2 Zählern. Der Beweis erfolgte in [ACH+95]. □

Es folgt ein Resultat, welches die Äquivalenz mehrerer Klassen hybrider Automaten zusammenfaßt.

Satz 4.6 (Cassez, Larsen, 2000) *Die Ausdrucksstärke von Stoppuhr-Automaten ist äquivalent zur Ausdrucksstärke der linearen hybriden Automaten. Sei $\mathcal{H}$ ein linearer hybrider Automat und die Funktion $\mathfrak{L}$ bezeichne die von einem Automaten erzeugte Sprache (Menge von Spuren). Dann existiert ein Stoppuhr-Automat $\mathcal{A}_S$ mit $\mathfrak{L}(\mathcal{H}) = \mathfrak{L}(\mathcal{A}_S)$.*

Beweis: Ein Beweis, in dem ein Verfahren angegeben wird, das zu einem gegebenen linearen hybriden Automaten einen Stoppuhr-Automaten konstruiert, wird in [CL00] beschrieben. Bei den Stoppuhr-Automaten wird eine von der Umgebung nicht sichtbare Zeitverzögerung eingeführt. Dies hat jedoch auf die Erreichbarkeitsanalyse keine Auswirkungen. □

In den folgenden Abschnitten wird unter ”hybrider Automat” immer ein linearer hybrider Automat nach Definition 4.6 verstanden.

4.1.6 Modularität

Die in Abschnitt 2.3 eingeführten Konzepte zur Modularisierung lassen sich direkt auf die hybriden Automaten übertragen. Ein CTA-Modul $\mathcal{M}$ kann als Komponente $\mathcal{M}.\mathcal{A}$ (siehe Definition 2.7) auch einen linearen hybriden Automaten enthalten. Dadurch werden jedoch einige Anpassungen der Definitionen erforderlich.

Definition 4.17 *Die Definition für ein* **hybrides CTA-Modul** $\mathcal{M}$ *entsteht aus der Definition 2.7 für CTA-Modul (Seite 31) durch die folgenden Veränderungen:*

- $\mathcal{M}.\mathcal{A}$ *ist ein linearer hybrider Automat.*
- *Das vierte Axiom für die Menge der Eingabe-Variablen lautet:*
  - $\forall l \in \mathcal{M}.\mathcal{A}.L: \quad \mathcal{M}.\mathcal{A}.D(l) \in \Phi(\mathcal{M}.X \setminus \mathcal{M}.par_X^{-1}(\mathtt{I}))$
  - $\forall (l, \sigma, \mu, l') \in \mathcal{M}.\mathcal{A}.E: \quad \mu \in \Phi(\mathcal{M}.X \cup (\mathcal{M}.X \setminus \mathcal{M}.par_X^{-1}(\mathtt{I}))')$

Anmerkung: Die Definitionen für Eingabe-Variablen können wie folgt interpretiert werden:

- Die Ableitung für eine Eingabe-Variable darf von keinem Zustand eines im Modul enthaltenen Automaten eingeschränkt werden.
- Der Wert einer Eingabe-Variable darf in keinem der Zustandsübergänge durch eine Wertveränderung verändert werden.

Alle Variablen, die gestrichen in einer Wertveränderung auftreten, müssen somit als Ausgabe-, mehrfach einschränkbare oder lokale Variable deklariert sein.

Definition 4.18 *Die Definition für eine* **hybride CTA-Komposition** $\mathcal{C}$ *entsteht aus der Definition 2.8 für CTA-Komposition (Seite 34) durch die folgenden Veränderungen:*

- $\mathcal{C}.M = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ *ist eine endliche, nicht leere Menge von hybriden CTA-Modulen.*

- *Ein zusätzliches Axiom 7 für die Menge der Eingabe-Signale lautet wie folgt:*
  (Vermeidung von Einschränkungen der Eingabe-Signale)
  Sei $\mathcal{A} = \prod_{\mathcal{M} \in \mathcal{C}.M} \mathcal{M}.\mathcal{A}$ *der Produktautomat der Automaten* $\mathcal{M}_1.\mathcal{A}$ *bis* $\mathcal{M}_n.\mathcal{A}$*. Dann gilt:*

$$\forall \sigma \in \mathcal{C}.par_{\Sigma}^{-1}(\mathtt{I}): \quad \forall l \in \mathcal{A}.L: \left(\bigcup_{e \in \{d \in \mathcal{A}.E \mid \mathcal{A}.sync(d) = \sigma \wedge \pi_1(d) = l\}} \mathcal{A}.guard(e) \right) = Val(\mathcal{C}.X)$$

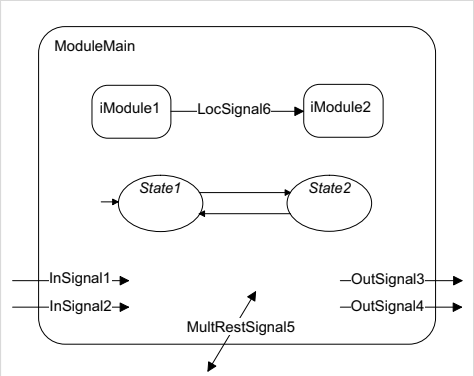
Anmerkung: Um die Einschränkungen von Eingabe-Signalen zu vermeiden, muß gewährleistet sein, daß zu jedem Zeitpunkt für jedes Eingabe-Signal ein Zustandsübergang erlaubt ist. Dazu muß für jedes Eingabe-Signal und für jeden Zustand die Disjunktion der Wächter eine Tautologie (identisch *true*) ergeben. In einem Produktautomat zweier hybrider Automaten können durch widersprüchliche Wertveränderungen bestimmte Zustandsübergänge ausgeschlossen sein, deshalb ist diese Forderung für die hybride CTA-Komposition notwendig.

Der Beweis, daß die Funktion $ctamod$ zu einer CTA-Komposition ein CTA-Modul berechnet, wird insofern einfacher, daß durch das zusätzliche Axiom der hybriden CTA-Komposition von Definition 4.18 das dritte Axiom von CTA-Modul in Definition 2.7 direkt gefordert wird und nicht mehr bewiesen werden muß. Das vierte Axiom von CTA-Modul wird analog der Argumentation für die $reset$-Mengen der Timed Automata für die Ableitungsbedingungen und die $update$-Mengen der hybriden Automaten sichergestellt.

Die CTA-Instanziierung aus Definition 2.10 auf Seite 38 kann unverändert auf hybride CTA-Module angewendet werden. Der Beweis, daß die Funktion $ctamod$ zu einer CTA-Instanziierung ein (hybrides) CTA-Modul berechnet, bleibt von der Erweiterung für hybride Automaten unberührt.

4.1.7 Beispiel: Steuerung Bahnübergang

Bei dem im folgenden zur Illustration der Modellierungsmöglichkeiten vorgestellten Steuerungssystem handelt es sich um ein vereinfachtes Modell eines Bahnübergangs (siehe auch [BLR00]). Die interessierende Sicherheitseigenschaft ist dabei, daß die Schranke vollständig geschlossen sein muß, wenn der Zug den Bahnübergang erreicht. Es gibt zwei Gleis-Sensoren, die die Position des Zuges signalisieren: Ein Sensor gibt ein Signal, wenn der Zug sich 1000 m vor dem Bahnübergang befindet, der zweite Sensor befindet sich 100 m nach dem Bahnübergang, um das Passieren des Zuges zu signalisieren (siehe Abbildung 4.1). Die Schranke kann durch einen Motorantrieb zwischen dem Winkel 0 Grad (Schranke ist vollständig geschlossen) und dem Winkel 90 Grad (Schranke ist geöffnet) bewegt werden. Die Steuerung soll die Schranke geschlossen haben, bevor der Zug einen Sicherheitsabstand von 250 m erreicht.



Zur übersichtlichen Darstellung von Modulen ist eine graphische Notation notwendig. Ein Rechteck mit abgerundeten Ecken steht für ein Modul, oben links befindet sich der Modulname (z. B. ModuleMain). Die Schnittstellen-Signale werden nach Zugriffstypen unterschieden: Ein von außen in das Modul-Rechteck eintretender Pfeil bezeichnet ein Eingabe-Signal (InSignal1), ein von innen nach außen zeigender Pfeil steht für ein Ausgabe-Signal (OutSignal3) und ein Doppelpfeil über die Modulgrenze hinweg bezeichnet ein mehrfach einschränkbares Signal (MultRestSignal5). Lokale Signale kommen nur innerhalb des Rechtecks für ein Modul vor.
Den in Abschnitt 2.3 auf Seite 29 eingeführten Modulkonzepten zufolge kann ein Modul auch aus einer Komposition mehrerer Module entstehen. In der Skizze sind zwei Modulinstanzen mit den Namen iModule1 und iModule2 dargestellt. Die Modulinstanzen kommunizieren über das Signal LocSignal6. Dieses Signal ist lokal im Modul ModuleMain, ein Ausgabe-Signal in der Modulinstanz iModule1 und ein Eingabe-Signal in Modulinstanz iModule2. Der in einem Modul enthaltene Automat wird wahlweise vollständig gezeichnet (meist bei Modulen ohne weitere Modulinstanzen), durch ein Symbol für einen Automaten abgekürzt oder ganz weggelassen, je nach Betrachtungsweise.

Abbildung 4.5: Darstellung von Modulen.

Das Modell besteht aus drei Subsystemen: dem Zug, der Schranke und der Steuerung. Die Steuerung koordiniert die Aktion der Schranke in Abhängigkeit vom Herannahen eines Zuges. Die Umgebung der Steuerung besteht aus einem Subsystem, welches das Verhalten des Zuges modelliert, und einem anderen Subsystem, welches das Verhalten der Schranke modelliert.

Die Abbildung 4.6 zeigt die Modulstruktur. Für die Darstellung von Modulen wird im folgenden das in der Abbildung 4.5 eingeführte Schema verwendet. Im Basismodul RailRoadCrossing sind vier Synchronisationsmarken definiert. Diese sind als lokale Marken deklariert, da es sich bei diesem Modul um ein Modell eines geschlossenen Systems handelt. Das Signal app modelliert das Sensorsignal der Zugposition ”1000-m vor der

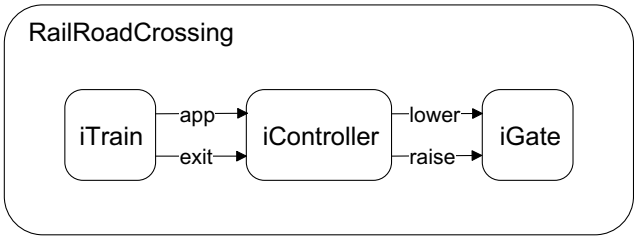


Abbildung 4.6: Schnittstellen und Anordnung der Module des Bahnübergang-Modells.

Schranke" und $exit$ modelliert das Signal für die Position "100 m nach der Schranke" für die Kommunikation zwischen Zugmodell und Steuerung. Die Signale $lower$ und $raise$ werden von der Steuerung benutzt, um Aktionen der Schranke auszulösen. Alle diese Marken müssen in diesem Modul deklariert sein, denn sie werden jeweils von mehr als einem enthaltenen Modul benutzt.

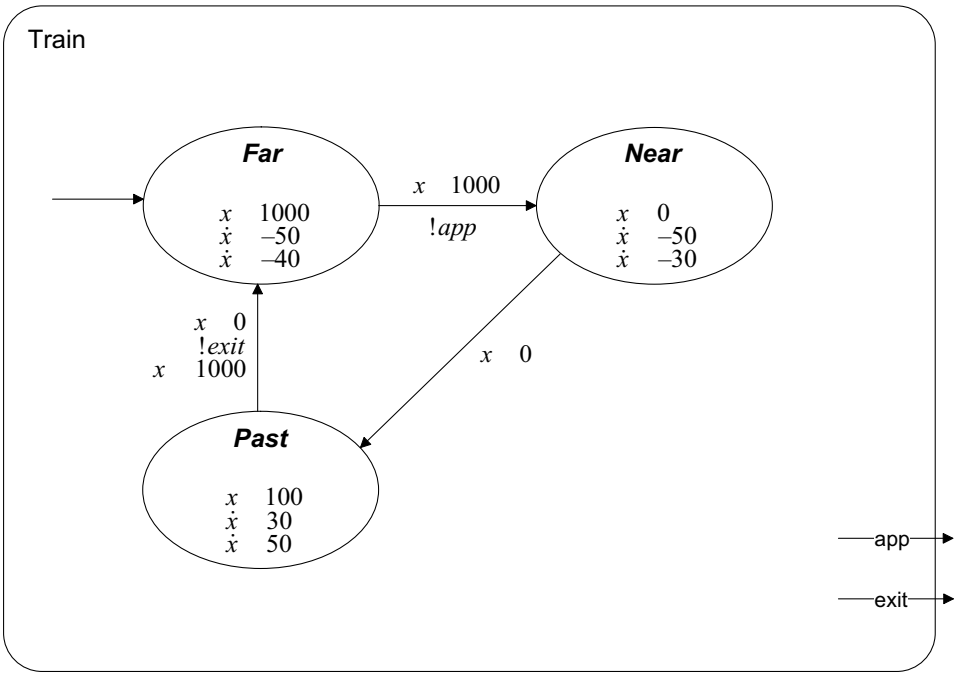


Abbildung 4.7: Zug-Modul des Bahnübergang-Modells. app und $exit$ sind Synchronisationsmarken, x ist eine lokale analoge Variable.

Das Modul Train modelliert das Verhalten des Zuges (Abbildung 4.7). Dabei wird die Position des Zuges durch die lokale Variable x ausgedrückt und seine Geschwindigkeit durch die erste Ableitung nach der Zeit $\dot{x}$. Da die beiden Sensoren die Position des Zuges anzeigen, werden die beiden Synchronisationsmarken app und $exit$ von diesem Modul gesteuert und somit als Ausgabe-Marken deklariert. Kein anderes Modul in der Umgebung

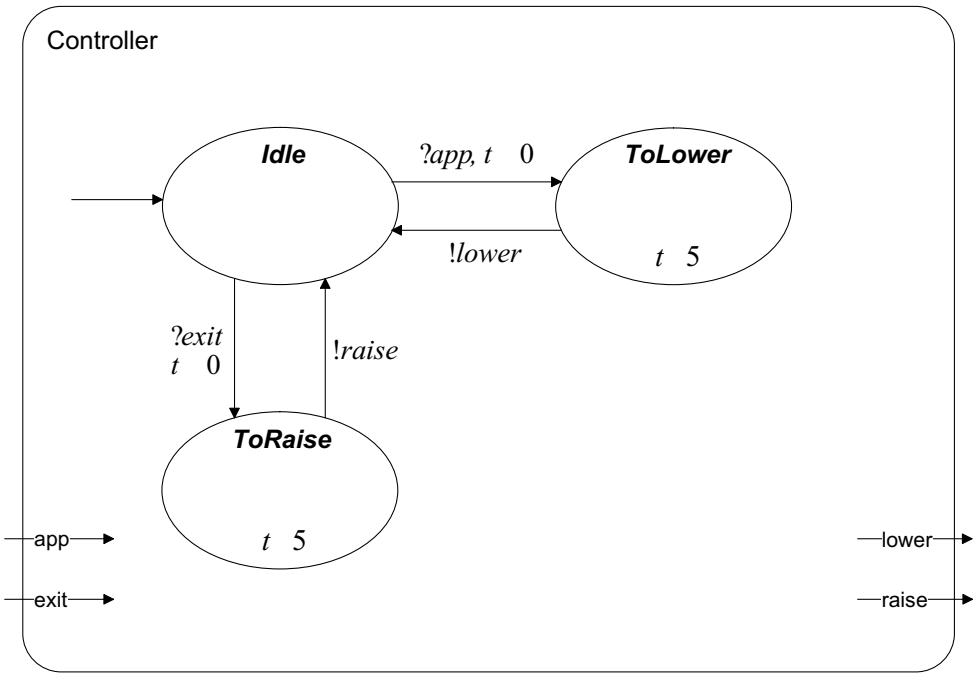


Abbildung 4.8: Steuerungsmodul des Bahnübergang-Modells. app, $exit$, $lower$ und $raise$ sind Synchronisationsmarken, t ist eine lokale Uhr.

darf diese Signale einschränken, d. h. auf dieses Signal muß jederzeit beim Auftreten reagiert werden.

Der im Modul Train enthaltene hybride Automat hat drei Zustände. Die Situation, bei der der Zug von großer Entfernung (größer 1000 m) herangefahren kommt, wird vom Zustand *Far* modelliert, der gleichzeitig auch Startzustand des Automaten ist. Die Invariante dieses Zustands ist, daß die Position des Zuges größer oder gleich 1000 m ist. Die Ableitung $\dot{x}$ liegt zwischen -50 und -40, um die Geschwindigkeit des sich nähernden Zuges zu modellieren. Wenn der Zug den ersten Sensor passiert ($x = 1000\ m$), wechselt der Automat zum Zustand *Near*. Dabei erfolgt durch die Synchronisationsmarke app eine Kommunikation mit der Steuerung. Nach dem CSP-Konzept müssen alle Automaten, die diese Synchronisation in ihrem Alphabet haben, auch einen Übergang mit der gleichen Synchronisationsmarke vollziehen. Die exklusiv einschränkende Wirkung des Übergangs wird durch Verwenden des Ausrufezeichens explizit in der Syntax wiedergegeben. Die Invariante stellt sicher, daß der Zustand verlassen werden muß, wenn der Zug gerade die Schranke passiert. Dann muß der Übergang zum Zustand *Past* gewählt werden. In einer Entfernung von 100 m nach der Schranke wird von der Invariante und von dem nächsten Übergang erzwungen, mit der Synchronisationsmarke $exit$ zu synchronisieren. Nun wird auch die Variable x auf einen Wert größer 1000 gesetzt und in den Startzustand gewechselt. Damit ist der Automat bereit, das Signalverhalten für den nächsten herannahenden Zug zu modellieren.

Das Modul Controller modelliert die eigentliche Steuerung, die die Schrankenbewegung bestimmt (das zu bauende steuernde elektronische Gerät oder die Steuersoftware). In Abbildung 4.8 auf der vorherigen Seite ist das Modul mit seinen Deklarationen und dem zugehörigen hybriden Automaten dargestellt. Das Modul kennt zwei Eingabe-Signale, um auf die Botschaften vom Zug-Modell reagieren zu können und zwei Ausgabe-Signale, um dem Schrankenmodell entsprechende Anweisungen zu erteilen. Die lokale Uhr t wird benutzt, um die Reaktionszeit der Steuerung zu messen. Es dürfen nicht mehr als 5 Zeiteinheiten zwischen der Mitteilung des Zugmodells und der Steueraktion der Schranke vergehen.

Startzustand des Automaten ist der Zustand *Idle*. In diesem Zustand wird gewartet, bis die Meldung vom Zug-Modell eintrifft, daß ein Zug die Position 1000 m erreicht hat, d. h. es wird ein mit der entsprechenden Synchronisationsmarke app markierter Zustandsübergang vollzogen. Der Automat befindet sich danach im Zustand *ToLower* und die Uhr t wurde zurückgesetzt. Der Zustand modelliert die Situation, daß die Meldung über den herannahenden Zug in der Steuerung eingetroffen ist und nach einer gewissen Reaktionszeit als nächstes die Schranke geschlossen werden muß. Die obere Schranke der Reaktionszeit α (im Beispiel auf den Wert 5 gesetzt) ist ein wichtiger Parameter zur Erfüllung der Sicherheitseigenschaft des Systems. Spätestens nach Ablauf von α muß eine Synchronisation mit dem Schrankenmodell über die Synchronisationsmarke $lower$ erfolgen und damit zum Ausgangszustand *Idle* zurückkehren. Wenn der Zug die Position "100 m nach der Schranke" passiert, synchronisiert das Zug-Modell mit der Marke $exit$, und die Steuerung wechselt in den Zustand *ToRaise*. Nach der Reaktionszeit erzwingt die Steuerung eine Synchronisation mit dem Schrankenmodell über das Signal $raise$ und wartet nach dem Übergang wieder im Anfangszustand *Idle*[3].

4.2 Interpretationen hybrider Automaten

In diesem Abschnitt wird eine Semantik vorgestellt, die von der Standardsemantik in der Weise abweicht, daß sie einen kompositionellen Operator unterstützt. Als semantische Basis wird dabei ein Update-System benutzt. Ziel ist, daß die Semantik der Komposition mehrerer hybrider Automaten äquivalent der Komposition mehrerer Semantiken hybrider Automaten ist. Anschließend wird die traditionelle Transitionssystem-Semantik als mittlere Abstraktionsschicht vorgestellt.

4.2.1 Semantik I: Kompositionelles Update-System

Während die von Alur et al. [ACH+95] und von Henzinger [Hen96] vorgeschlagenen Transitionssystem-Semantiken nicht kompositionell sind, wird hier eine kompositionelle Semantik definiert. Die Semantik eines Systems kann basierend auf den Semantiken

[3]Dieses stark vereinfachte Modell einer Schrankensteuerung dient lediglich der Veranschaulichung der Modellierung hybrider Größen wie Winkel und Geschwindigkeit. Es berücksichtigt keinerlei Sonderfälle, z. B. die Reaktion beim Herannahen eines zweiten Zuges, während sich der erste Zug noch im Schrankenbereich befindet.

der Bestandteile des Systems erklärt werden [BR01]. Dazu muß der Operator für die Parallelschaltung hybrider Automaten kompositionell sein.

Die Semantik eines dynamischen Systems wird oft als Transitionssystem definiert. Für Systeme, die synchrone Komposition erlauben, bietet ein Transitionssystem nicht genügend Struktur: Die auf markierten Transitionssystemen beruhende Semantik ist nicht kompositionell für synchrone Parallelschaltung mehrerer Automaten mit gemeinsamen Variablen. Der hier behandelte Modellierungsformalismus besitzt einen synchronen Kompositionsoperator. Deshalb wird hier eine auf Update-Systemen beruhende Semantik eingeführt, die genügend Struktur behält und somit die Definition einer synchronen Komposition auf semantischer Ebene erlaubt.

Definition 4.19 *Ein* **markiertes Update-System** *(kurz: Update-System) ist ein Tupel* $\mathcal{S} = (L, X, Q^0, \Sigma, U)$ *mit den folgenden Komponenten:*

- L *ist die Menge von* **Zuständen**.
- X *ist die Menge der* **Variablen**.
  $Q =_{def} L \times Val(X)$ *ist die Konfigurationsmenge des Update-Systems. Jede Konfiguration ist ein Paar aus Zustand und Variablenbelegung.*
- $Q^0 \subseteq Q$ *ist die Menge der* **Initialkonfigurationen**.
- Σ *ist die Menge der* **Marken**.
- $U \subseteq L \times Val(X) \times \Sigma \times L \times Updt(X)$ *sind die* **Updates** *des Update-Systems.*

Anmerkung: $(l, v, \sigma, l', u) \in U$ beschreibt, daß in der Konfiguration (l, v) bei der Marke σ der Zustand l' erreicht wird und der durch u beschriebene Update der Variablen stattfinden kann. Dies bedeutet, daß jede Variable x im Definitionsbereich von u den neuen Wert $u(x)$ zugeordnet bekommt, und daß alle anderen Variablen unverändert bleiben.

Für $v, v' \in Val(X)$ und $\delta \in \mathbb{R}$ bezeichnet $v \cdot \delta$ die Variablenbelegung von X mit $(v \cdot \delta)(x) = v(x) \cdot \delta$ für alle Variablen $x \in X$. $v + v'$ bezeichnet die Variablenbelegung von X mit $(v + v')(x) = v(x) + v'(x)$ für alle Variablen $x \in X$. Das Vergehen einer Zeit $\delta \in \mathbb{R}_+$ bei einer festgelegten Ableitung nach der Zeit $d \in Val(X)$ für die zeitabhängigen Wertveränderungen der Variablen führt von einer Konfiguration (l, v) zu einer Folgekonfiguration $(l, v + \delta \cdot d)$.

Definition 4.20 *Sei* $\mathcal{H}$ *ein hybrider Automat. Das durch* $\mathcal{H}$ *bestimmte* **markierte Update-System** $[\![\mathcal{H}]\!]_U = (L, X, Q^0, \Sigma, U)$ *ist wie folgt definiert:*

- $L =_{def} \mathcal{H}.L$
- $X =_{def} \mathcal{H}.X$
- $Q^0 =_{def} \{(l, [\![\varphi]\!]) \mid (l, \varphi) \in \mathcal{H}.Q^0\}$
- $\Sigma =_{def} \mathcal{H}.\Sigma \cup \mathbb{R}_+$

- $U =_{def} time(\mathcal{H}) \cup discrete(\mathcal{H})$ *mit:*

$$time(\mathcal{H}) \quad =_{def} \quad \left\{ (l, v, \sigma, l, u) \;\middle|\; \begin{array}{l} \exists d \in [\![\mathcal{H}.D(l)]\!] : \\ \quad u = v + \sigma \cdot d \\ \wedge\ \sigma \in (\mathbb{R}_+ \setminus 0) \\ \wedge\ \forall \delta \in [0, \sigma] : v + \delta \cdot d \in [\![\mathcal{H}.I(l)]\!] \end{array} \right\}$$

$$discrete(\mathcal{H}) \quad =_{def} \quad \left\{ (l, v, \sigma, l', u) \;\middle|\; \begin{array}{l} \exists (l, \sigma, \mu, l') \in \mathcal{H}.E : \\ \quad u \in [\![\mu]\!]_U(v) \end{array} \right\}$$

Anmerkung: Die Konfigurationsmenge des Update-Systems besteht aus den Zuständen und Variablenbelegungen des hybriden Automaten. Die Menge der Initialkonfigurationen des Update-Systems ist durch die Initialbedingung des hybriden Automaten definiert. Ein hybrider Automat kann die Konfiguration durch das Vergehen von Zeit und durch Zustandsübergänge verändern; somit besteht die Menge der Updates des Update-Systems aus allen *Zeit-Updates* und allen *diskreten Updates*. Die Zustandskomponente einer Konfiguration wird durch einen Zeit-Update nicht verändert.

Falls eine Variable in keiner der Ungleichungen der Wertveränderung eines hybriden Automaten in gestrichener Form auftritt, ist damit nicht gemeint, daß der gesamte Wertebereich von $\mathbb{R}$ möglich ist. Für eine solche Variable ist die Interpretation die folgende: Der Zustandsübergang verändert den Wert der Variablen nicht, aber parallel ausgeführte Zustandsübergänge der Umgebungsautomaten können die Variable einschränken. Schränkt kein parallel ausgeführter Zustandsübergang eines Umgebungsautomaten die Variable x ein, dann wird der Wert der Variablen nicht verändert. Um auszudrücken, daß der ganze Wertebereich von $\mathbb{R}$ möglich ist für x nach der Ausführung des Zustandsübergangs ohne die Einwirkung eines parallellaufenden Automaten, würde die Wertveränderung $x' \geq 0 \vee x' \leq 0$ verwendet werden.

Für die diskreten Updates ist die Invariante irrelevant. Eine Invariante, die identisch $false$ ist, kann zur Konstruktion von ”dringenden” Zuständen benutzt werden. Dringende Zustände sind Zustände des Automaten, in denen keine Zeit vergehen darf.

Die Definition der Updates stellt damit auch im Hinblick auf die Semantik der Invarianten eine wesentliche Änderung gegenüber der in der Literatur gebräuchlichen Semantik dar: Bei der Standardinterpretation (z. B. in [Hen96]) werden Zustandsübergänge, die intuitiv im Modell erlaubt wären, in der Transitionssystem-Semantik und somit auch bei der Erreichbarkeitsanalyse nicht berücksichtigt, falls im Zielzustand die Invariante nicht erfüllt ist. Im Modell sind ’Deadlocks’ möglich, die bei diesem Vorgehen in der Transitionssystem-Semantik jedoch nicht mehr enthalten und damit auch nicht nachweisbar ist. Dadurch verringert sich die Anzahl der Zustände im Produktautomaten, was zu Performancesteigerung führt. Dieses Problem wird teilweise bereinigt, indem in [HHMWT00] nachträglich eine andere Definition hybrider Automaten vorgeschlagen wurde: Nun wird für das Modell gefordert, daß die Menge der zulässigen Wächter für einen Zustandsübergang die Invariante des Anfangszustands erfüllen. Außerdem wird für jeden Übergang gefordert, daß die durch die Wertveränderung eines Übergangs berechneten neuen Werte der Variablen die Invariante des Folgezustands erfüllen. Somit wurde

dieses technische Detail zum Zwecke einer performanten Analyse in die Modellebene zurückübertragen. Hier wird eine neue Semantik definiert und der Performancenachteil bei der Werkzeug-Implementierung in Kauf genommen.

Bei beiden Semantiken von Henzinger tritt zusätzlich das folgende Problem auf: Ein Automat kann die Variablen per Zustandsübergang so setzen, daß die Zustandsinvariante eines anderen, parallel laufenden Automaten ungültig wird. Dadurch wird dieser Automat in einen 'Deadlock' gezwungen. Bei der in dieser Arbeit vorgeschlagenen Semantik hat der Automat die Möglichkeit, sofort in einen anderen Zustand zu wechseln. Die hier benutzte Invariantensemantik besagt also, daß in einem Zustand mit falscher Invariante *keine Zeit vergehen darf*. Es führt jedoch nicht zu einem unzulässigen Lauf, wenn der Automat einen solchen Zustand passiert, *ohne Zeit vergehen zu lassen*. Diese Betrachtung gilt auch für das Rücksetzen von Uhren bei Timed Automata. Deshalb wurde die neue Invariantensemantik bereits bei der Semantik von Timed Automata berücksichtigt (siehe Abschnitt 2.4.1, Definition 2.13).

Illustration. Zur Illustration der beiden Update-Arten wird hier noch einmal auf das Modell der Schrankensteuerung (siehe Abbildung 4.8 auf Seite 121 für den hybriden Automaten) zurückgegriffen. Falls der Automat im Modul Controller den Zustand *ToLower* betritt, wird die Uhr t auf den Wert 0 gesetzt[4]. D. h. für die Konfiguration (l, v) unmittelbar nach dem Übergang gilt: $l =$ *ToLower* und $v(t) = 0$. In dieser Situation sind die folgenden **Zeit-Updates** des Automaten möglich: Für jede Zeit δ aus dem Intervall $(0, 5]$ kann der Automat einen Zeit-Update durchführen, der zur Folgekonfiguration (l', v') mit $l' = l$ und $v'(t) = v(t) + \delta \cdot 1$ führt. Weiterhin hat der Automat die Möglichkeit, einen **diskreten Update** durchzuführen, indem der Zustandsübergang zum Zustand *Idle* ausgewählt wird. Dies führt zur Folgekonfiguration (l'', v'') mit $l'' =$ *Idle* und $v''(t) = v(t)$, d. h. die Uhr ist nicht verändert worden.

Update-Systeme enthalten ausreichend Struktur, um die synchrone Komposition zu ermöglichen.

Definition 4.21 *Seien* $\mathcal{S}_1 = (L_1, X_1, Q_1^0, \Sigma_1, U_1)$ *und* $\mathcal{S}_2 = (L_2, X_2, Q_2^0, \Sigma_2, U_2)$ *zwei markierte Update-Systeme. Dann ist die* **synchrone Komposition** $\mathcal{S}_1 \,||\, \mathcal{S}_2$ *durch das markierte Update-System* $(L_1 \times L_2, X_1 \cup X_2, Q^0, \Sigma_1 \cup \Sigma_2, U)$ *gegeben, wobei:*

- $((l_1, l_2), v) \in Q^0$ *gdw.* $(l_1, v|_{X_1}) \in Q_1^0 \quad \wedge \quad (l_2, v|_{X_2}) \in Q_2^0$
- $((l_1, l_2), v, \sigma, (l_1', l_2'), u) \in U$ *gdw.*

[4]Durch die Deklaration der Variablen t als Uhr (CLOCK) ist die Angabe der Ableitung $\dot{t} = 1$ überflüssig, da für Uhren nur die Ableitung 1 erlaubt ist.

$$\left(\begin{array}{ll} & \sigma \in \Sigma_1 \cap \Sigma_2 \\ \wedge & \exists (l_1, v_1, \sigma, l_1', u_1) \in U_1 : \; v|_{X_1} = v_1 \; \wedge \; u|_{\mathrm{dom}(u_1)} = u_1 \\ \wedge & \exists (l_2, v_2, \sigma, l_2', u_2) \in U_2 : \; v|_{X_2} = v_2 \; \wedge \; u|_{\mathrm{dom}(u_2)} = u_2 \\ \wedge & \mathrm{dom}(u) = \mathrm{dom}(u_1) \cup \mathrm{dom}(u_2) \end{array}\right)$$

$$\vee \left(\begin{array}{ll} & \sigma \in \Sigma_1 \setminus \Sigma_2 \\ \wedge & \exists (l_1, v_1, \sigma, l_1', u) \in U_1 : \; v|_{X_1} = v_1 \end{array}\right)$$

$$\vee \left(\begin{array}{ll} & \sigma \in \Sigma_2 \setminus \Sigma_1 \\ \wedge & \exists (l_2, v_2, \sigma, l_2', u) \in U_2 : \; v|_{X_2} = v_2 \end{array}\right)$$

Zu beachten ist dabei folgendes: Bleibt bei einem Zustandsübergang der Wert einer Variable unverändert, so ist bei einem Transitionssystem (im Gegensatz zum Update-System) nicht nachvollziehbar, ob dies daher resultiert, daß die Variable bei der Wertveränderung des Zustandsübergangs unberücksichtigt blieb oder daß der Wert der Variable explizit auf den gleichen Wert wie vor dem Zustandsübergang gesetzt wurde. Hingegen ist diese Information in einem Update-System verfügbar. Dies ist der Grund dafür, daß die auf einem Update-System basierende Semantik kompositionell ist für synchrone Parallelschaltung, die Transitionssystem-Semantik jedoch nicht.

Das Symbol $||$ wird sowohl bei hybriden Automaten als auch bei Update-Systemen als Kompositionsoperator benutzt.

Die parallele Komposition zweier Update-Systeme ist wieder ein markiertes Update-System.

Satz 4.7 *Seien $\mathcal{S}$ und $\mathcal{S}'$ zwei markierte Update-Systeme. Dann ist auch $\mathcal{S} \,||\, \mathcal{S}'$ ein markiertes Update-System.*

Beweis: Die Behauptung folgt aus der Konstruktion der einzelnen Komponenten von $\mathcal{S} \,||\, \mathcal{S}'$ entsprechend Definition 4.21. □

Satz 4.8 *Die auf markierte Update-Systeme basierende Semantik ist kompositionell für hybride Automaten, d. h. für zwei hybride Automaten $\mathcal{H}$ und $\mathcal{H}'$ gilt $[\![\mathcal{H} \,||\, \mathcal{H}']\!]_U = [\![\mathcal{H}]\!]_U \,||\, [\![\mathcal{H}']\!]_U$.*

Beweis: Der Beweis folgt ebenfalls direkt aus den Definitionen des Operators $||$ (Definition 4.8 auf Seite 109 und Definition 4.21 auf der vorherigen Seite). □

Anmerkung: In [BR99a] wurde versucht, dieses Problem durch ein sogenanntes "Double Transition Predicate" zu lösen. Jeder Transition wurden zwei Prädikate zugeordnet. Das $allowed$-Prädikat stellte die vom Automaten erzwungene Übergangsbedingung für die Wertveränderung dar, d. h. alle Variablen, über die in diesem Prädikat nichts ausgesagt wurde, sollten als unverändert interpretiert werden. Das $initiated$-Prädikat enthielt für die unveränderten Variablen die Bedingung, daß der Wert unverändert bleiben soll. Bei der Formalisierung dieses Konzeptes in [BR01] traten Probleme bezüglich der sauberen Trennung von Syntax und Semantik auf. Die parallele Komposition der $allowed$-Mengen konnte mittels Durchschnitt ermittelt werden. Bei der Konstruktion der neuen $initiated$-Menge

mußte jedoch auf die Information aus der Syntax zurückgegriffen werden um festzustellen, ob eine Variable im Prädikat erwähnt worden ist. Daher wird in dieser Arbeit eine wesentlich vorteilhaftere kompositionelle Semantik auf der Basis von Update-Systemen verwendet.

4.2.2 Semantik II: Kontinuierliches Transitionssystem

Zu einem hybriden Automaten $\mathcal{H}$ kann ein korrespondierendes kontinuierliches Transitionssystem $[\![\mathcal{H}]\!]_C$ konstruiert werden, indem zunächst das entsprechende Update-System $[\![\mathcal{H}]\!]_U$ konstruiert und daraus das Transitionssystem mit einer Semantikfunktion für Update-Systeme erzeugt wird, d. h. $[\![\mathcal{H}]\!]_C =_{def} [\![[\![\mathcal{H}]\!]_U]\!]_C$. Es muß eine Semantikfunktion definiert werden, die zu einem Update-System ein Transitionssystem liefert.

Definition 4.22 *Die* **kontinuierliche Transitionssystem-Semantik** $[\![\mathcal{S}]\!]_C$ *eines markiertes Update-Systems* $\mathcal{S}$ *ist gegeben durch das markierte Transitionssystem* $(\mathcal{S}.L \times Val(\mathcal{S}.X), \mathcal{S}.Q^0, \mathcal{S}.\Sigma, \rightarrow)$*, wobei* $\rightarrow$ *die folgenden Transitionen enthält:*

- *Für* $(l, v), (l', v') \in \mathcal{S}.L \times Val(\mathcal{S}.X)$ *und* $\sigma \in \mathcal{S}.\Sigma$ *gilt* $(l, v) \xrightarrow{\sigma} (l', v')$*, gdw. ein* $(l, v, \sigma, l', u) \in \mathcal{S}.U$ *existiert mit* $v' = v \lhd u$*.*

Anmerkung: Der Konfigurationsraum des Transitionssystems ist der Konfigurationsraum des Update-Systems. Die Menge der Initialkonfigurationen des Transitionssystems ist definiert durch die Initialkonfigurationen des Update-Systems. Die Transitionen des Transitionssystems werden berechnet, indem die Update-Funktion angewendet wird, d. h. für jede Ausgangskonfiguration der Transitionsrelation wird durch die im Update-System vorhandenen Updates eine Nachfolger-Konfiguration mittels Überschreibe-Operator erzeugt und zugeordnet.

Durch die Definition der Transitionssystem-Semantik kann zu einem Update-System $\mathcal{S}$ die Spur-Semantik $[\![\mathcal{S}]\!]_{\mathcal{L}}$ konstruiert werden, indem für $\mathcal{S}$ zunächst das zugehörige Transitionssystem $[\![\mathcal{S}]\!]_C$ gebildet wird. Dann ist $[\![\mathcal{S}]\!]_{\mathcal{L}}$ definiert als $[\![[\![\mathcal{S}]\!]_C]\!]_{\mathcal{L}}$.

Satz 4.9 *Seien* $\mathcal{S}^1 = (L^1, X^1, Q_0^1, \Sigma^1, U^1)$ *und* $\mathcal{S}^2 = (L^2, X^2, Q_0^2, \Sigma^2, U^2)$ *zwei Update-Systeme mit* $X^1 \cap X^2 = \emptyset$ *und* $\Sigma^1 = \Sigma^2$*. Dann ist die Spur-Semantik* **kompositionell** *für den Operator* $||$ *:* $[\![\mathcal{S}^1]\!]_{\mathcal{L}} \,||\, [\![\mathcal{S}^2]\!]_{\mathcal{L}} = [\![\mathcal{S}^1 \,||\, \mathcal{S}^2]\!]_{\mathcal{L}}$.

Anmerkung: Seien $\mathcal{H}^1$ und $\mathcal{H}^2$ zwei hybride Automaten mit $\mathcal{H}^1.X \cap \mathcal{H}^2.X = \emptyset$ und $\mathcal{H}^1.\Sigma = \mathcal{H}^2.\Sigma$. Dann gilt die Kompositionalität der Spursemantik wegen Satz 4.8 auch für hybride Automaten: $[\![[\![\mathcal{H}^1]\!]_U]\!]_{\mathcal{L}} \,||\, [\![[\![\mathcal{H}^2]\!]_U]\!]_{\mathcal{L}} = [\![[\![\mathcal{H}^1 \,||\, \mathcal{H}^2]\!]_U]\!]_{\mathcal{L}}$.

Beweis: $\subseteq$: Zunächst wird $[\![\mathcal{S}^1]\!]_{\mathcal{L}} \,||\, [\![\mathcal{S}^2]\!]_{\mathcal{L}} \subseteq [\![\mathcal{S}^1 \,||\, \mathcal{S}^2]\!]_{\mathcal{L}}$ gezeigt. Aus $\alpha = (\sigma_i)_{i\in\mathbb{N}} \in [\![\mathcal{S}^1]\!]_{\mathcal{L}} \,||\, [\![\mathcal{S}^2]\!]_{\mathcal{L}}$ folgt nach der Definition der Komposition zweier Spur-Mengen als Durchschnitt $\alpha \in [\![\mathcal{S}^1]\!]_{\mathcal{L}}$ und $\alpha \in [\![\mathcal{S}^2]\!]_{\mathcal{L}}$. Somit existieren zwei Läufe $((l_i^1, v_i^1), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^1$ und $((l_i^2, v_i^2), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^2$ für α. Weil die beiden Systeme miteinander nur über Marken kommunizieren und nicht über gemeinsame Variablen ($X^1 \cap X^2 = \emptyset$), kann ein Lauf $(((l_i^1, l_i^2), v_i), \sigma_i)_{i\in\mathbb{N}}$ mit $v_i|_{X^1} = v_i^1$ und $v_i|_{X^2} = v_i^2$ konstruiert werden. Da die

Komposition zweier Updates eines Update-Systems genau in dieser Weise erzeugt wird, kann geschlußfolgert werden, daß $(((l_i^1, l_i^2), v_i), \sigma_i)_{i\in\mathbb{N}}$ ein Lauf von $\mathcal{S}^1 \mid\mid \mathcal{S}^2$ ist und somit $\alpha \in [\![\mathcal{S}^1 \mid\mid \mathcal{S}^2]\!]_{\mathcal{L}}$.

$\supseteq$: Es verbleibt, $[\![\mathcal{S}^1]\!]_{\mathcal{L}} \mid\mid [\![\mathcal{S}^2]\!]_{\mathcal{L}} \supseteq [\![\mathcal{S}^1 \mid\mid \mathcal{S}^2]\!]_{\mathcal{L}}$ zu zeigen. Für jede Spur $\alpha \in [\![\mathcal{S}^1 \mid\mid \mathcal{S}^2]\!]_{\mathcal{L}}$ existiert ein Lauf $(((l_i^1, l_i^2), v_i), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^1 \mid\mid \mathcal{S}^2$. Da $X^1 \cap X^2 = \emptyset$, kann $v_i : X^1 \cup X^2 \to \mathbb{R}$ geteilt werden in $v_i^1 : X^1 \to \mathbb{R}$ und $v_i^2 : X^2 \to \mathbb{R}$ mit $v_i|_{X^1} = v_i^1$ und $v_i|_{X^2} = v_i^2$. Per Induktion entstehen bei diesem Verfahren Läufe. Damit können zwei Läufe $((l_i^1, v_i^1), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^1$ und $((l_i^2, v_i^2), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^2$ konstruiert werden, und es folgt $\alpha \in [\![\mathcal{S}^1]\!]_{\mathcal{L}}$ und $\alpha \in [\![\mathcal{S}^2]\!]_{\mathcal{L}}$. □

Anmerkung: Ohne die Bedingung $X^1 \cap X^2 = \emptyset$ gilt die Behauptung nicht für Mengen von Spuren. Als Gegenbeispiel wird eine Spur $\alpha = (\sigma_i)_{i\in\mathbb{N}} \in [\![\mathcal{S}^1]\!]_{\mathcal{L}} \mid\mid [\![\mathcal{S}^2]\!]_{\mathcal{L}}$ betrachtet und daß nur Läufe $((l_i^1, v_i^1), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^1$ und $((l_i^2, v_i^2), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^2$ existieren mit $v_k^1(x) \neq v_k^2(x)$ für ein k und $x \in X^1 \cap X^2$. Dann kann kein Lauf $(((l_i^1, l_i^2), v_i), \sigma_i)_{i\in\mathbb{N}}$ von $\mathcal{S}^1 \mid\mid \mathcal{S}^2$ existieren und somit gilt $\alpha \notin [\![\mathcal{S}^1 \mid\mid \mathcal{S}^2]\!]_{\mathcal{L}}$.

$\Sigma^1 = \Sigma^2$ ist eine weitere notwendige Bedingung. Sei $\Sigma^1 \setminus \Sigma^2 \neq \emptyset$. Dann ist eine Reaktion des Systems $\mathcal{S}^1$ auf die Marke $\sigma \in \Sigma^1 \setminus \Sigma^2$ im durch Komposition entstandenen System $\mathcal{S}^1 \mid\mid \mathcal{S}^2$ möglich, aber die Marke σ kann im Durchschnitt der Spuren $[\![\mathcal{S}^1]\!]_{\mathcal{L}}$ und $[\![\mathcal{S}^2]\!]_{\mathcal{L}}$ nicht vorkommen, weil $\mathcal{S}^2$ keine Spur hat, in der σ vorkommt.

4.3 Verifikation

Die Verifikation von hybriden Modellen erfolgt mittels Erreichbarkeitsanalyse auf der Transitionssystem-Semantik. Für die algorithmische Analyse sind nur lineare hybride Automaten zugelassen.

In diesem Abschnitt werden zunächst einige Unterschiede bei der Erreichbarkeitsanalyse im Vergleich zum Algorithmus für Timed Automata aufgezeigt. Anschließend wird die Datenstruktur für die Repräsentation der Transitionsrelation bei kontinuierlicher Semantik von hybriden Automaten sowie für die Konfigurationsmengen angegeben.

4.3.1 Erreichbarkeitsanalyse

Abbildung 4.9 auf der nächsten Seite zeigt den Algorithmus für die Erreichbarkeitsanalyse hybrider Automaten. Es werden die bereits bei der Behandlung von Timed Automata eingeführten Notationen wiederverwendet. In einer Fixpunktiteration werden immer wieder Nachfolger von bereits erreichten Konfigurationen berechnet, bis die Menge aller erreichbaren Konfigurationen $Reach([\![\mathcal{H}]\!]_C)$ (siehe Definition 2.14 auf Seite 42) erreicht ist. Gegenüber der Strategie für Timed Automata gibt es einige Besonderheiten:

- Die zu überprüfende Menge Q_F ist eine Menge von Konfigurationen. Es kann nicht nur die Erreichbarkeit von Zuständen überprüft werden, sondern auch von bestimmten Variablenbelegungen. Im Unterschied zu den Timed Automata können bei hybriden Automaten als Initialkonfigurationen auch Variablenbelegungen ungleich 0 vorkommen.

Eingabe: Hybrider Automat $\mathcal{H} = (L, Q^0, X, \Sigma, I, D, E)$
mit der kontinuierlichen Semantik $[\![\mathcal{H}]\!]_C = (L \times Val(X), Q^0, \Sigma \cup \mathbb{R}_+, \rightarrow)$,
Menge von Konfigurationen Q_F
Ausgabe: $true$ gdw. $Reach([\![\mathcal{H}]\!]_C) \cap Q_F \neq \emptyset$

$R := Q^0$
do
$R_{prev} := R$
$R := R \cup \{q' \in L \times Val(X) \mid \exists q : q \in R \wedge q \rightarrow q'\}$
while $R \not\subseteq R_{prev}$
return $R \cap Q_F \neq \emptyset$

Abbildung 4.9: Algorithmus der Erreichbarkeitsanalyse für hybride Automaten.

- Die Abbruchbedingung wird über eine Teilmengenbeziehung geprüft, da die Gleichheit nicht direkt implementiert werden kann ($= gdw. \subseteq \wedge \supseteq$).
- Es wird versucht, den kleinsten Fixpunkt der erreichbaren Konfigurationen zu berechnen. Da der Algorithmus dafür nicht bei jedem Modell terminiert (siehe Satz 4.5), ist als weitere, oft terminierende Strategie die Rückwärtsanalyse nach dem in Abbildung 4.10 angegebenen Algorithmus notwendig.

Eingabe: Hybrider Automat $\mathcal{H} = (L, Q^0, X, \Sigma, I, D, E)$
mit der kontinuierlichen Semantik $[\![\mathcal{H}]\!]_C = (L \times Val(X), Q^0, \Sigma \cup \mathbb{R}_+, \rightarrow)$,
Menge von Konfigurationen Q_F
Ausgabe: $true$ gdw. $Reach([\![\mathcal{H}]\!]_C) \cap Q_F \neq \emptyset$

$R := Q_F$
do
$R_{prev} := R$
$R := R \cup \{q \in L \times Val(X) \mid \exists q' : q' \in R \wedge q \rightarrow q'\}$
while $R \not\subseteq R_{prev}$
return $R \cap Q^0 \neq \emptyset$

Abbildung 4.10: Algorithmus für die Rückwärts-Analyse.

Da diese Algorithmen für lineare hybride Automaten nicht in jedem Fall terminieren, werden sie in der Literatur oft auch als ”Semi-Entscheidungsprozeduren” bezeichnet [ACHH93].

Zur effizienten Gestaltung der Erreichbarkeitsanalyse werden einige Abweichungen von den in den Abbildungen 4.9 und 4.10 gezeigten allgemeinen Algorithmen vorgenommen. Neben einigen technischen Details, wie Sonderfallbehandlungen zur Effizienzsteigerung, wird bei der Berechnung der Erreichbarkeitsmenge neben der Menge der bisher erreichten Konfigurationen eine Menge der zuletzt erreichten Konfigurationen gespei-

Eingabe: Hybrider Automat $\mathcal{H} = (L, Q^0, X, \Sigma, I, D, E)$
mit der kontinuierlichen Semantik $[\![\mathcal{H}]\!]_C = (L \times Val(X), Q^0, \Sigma \cup \mathbb{R}_+, \rightarrow)$,
Menge von Konfigurationen Q_F
Ausgabe: $true$ gdw. $Reach([\![\mathcal{H}]\!]_C) \cap Q_F \neq \emptyset$

$R_{reached} := \emptyset$
$R_{new} := Q^0$
while $R_{new} \not\subseteq R_{reached}$ **do**
  if $R_{new} \cap Q_F \neq \emptyset$ **then return** $true$
  $R_{reached} := R_{reached} \cup R_{new}$
  $R_{new} := \{q' \in L \times Val(X) \mid \exists q : q \in R_{new} \wedge q \rightarrow q'\}$
od
return $false$

Abbildung 4.11: Verbesserter Erreichbarkeitsalgorithmus für hybride Automaten.

chert, um nicht immer wieder bereits vorhandene Konfigurationen neu zu berechnen (siehe auch [ACHH93]). Ein Algorithmus, der sowohl diese Technik als auch den Abbruch der Analyse beim Erreichen einer Zielkonfiguration berücksichtigt, ist in Abbildung 4.11 dargestellt.

Eingabe: Hybrider Automat $\mathcal{H} = (L, Q^0, X, \Sigma, I, D, E)$
mit der kontinuierlichen Semantik $[\![\mathcal{H}]\!]_C = (L \times Val(X), Q^0, \Sigma \cup \mathbb{R}_+, \rightarrow)$,
Menge von Konfigurationen Q_F
Ausgabe: $true$ gdw. $Reach([\![\mathcal{H}]\!]_C) \cap Q_F \neq \emptyset$

$R_{reached} := \emptyset$
$R_{new} := Q^0$
while $R_{new} \neq \emptyset$ **do**
  if $R_{new} \cap Q_F \neq \emptyset$ **then return** $true$
  $R_{new} := \{q' \in L \times Val(X) \mid \exists q : q \in R_{new} \wedge q \rightarrow q'\}$
  $R_{added} := R_{new} \setminus R_{reached}$
  $R_{reached} := R_{reached} \cup R_{new}$
  $R_{new} := R_{added}$
od
return $false$

Abbildung 4.12: Weiter verbesserter Erreichbarkeitsalgorithmus.

Das konsequente Anwenden der Idee, die Berechnung bereits vorhandener Konfigurationen zu vermeiden, führt zum in Abbildung 4.12 dargestellten Algorithmus. Hier werden von der Menge der in einem Schritt berechneten Konfigurationen (R_{new}) zur Vorbereitung auf den nächsten Schritt diejenigen Konfigurationen abgezogen, die bereits in der

Gesamtmenge der berechneten Konfigurationen ($R_{reached}$) enthalten waren. Damit wird R_{new} reduziert auf R_{added}.

4.3.2 DDM-Repräsentation

Um Konfigurationsmengen für lineare hybride Automaten speichern und verarbeiten zu können, muß eine neue Repräsentationsmethode eingeführt werden, die Double Description Method. Dieses Verfahren basiert auf einer Matrizendarstellung konvexer Mengen von Variablenbelegungen. Zustände werden bei der Standardrepräsentation explizit dargestellt. Im folgenden werden nur lineare hybride Automaten betrachtet und abkürzend als hybride Automaten bezeichnet.

Definition 4.23 *Für einen hybriden Automaten $\mathcal{H}$ ist die Menge aller Konfigurationen definiert als die Menge $Q = \mathcal{H}.L \times Val(\mathcal{H}.X)$. Ein Element $q = (l, v) \in Q$ heißt* **Konfiguration**. *Eine Teilmenge $R \subseteq Q$ heißt* **Region**. *Eine Region R heißt* **Zustandsregion**, *falls für ein $l \in \mathcal{H}.L$ und ein $V \subseteq Val(\mathcal{H}.X)$ gilt: $R = \{(l, v) \mid v \in V\}$, und* **konvex**, *falls für ein $l \in \mathcal{H}.L$ gilt: $R = \{(l, v) \mid v \in [\![\varphi]\!]\}$ und φ ist eine Konjunktion von Ungleichungen (nach Definition 4.3).*

Die einem Zustand in einer Zustandsregion zugeordnete Belegungsmenge ist durch ein *konvexes Polyeder* repräsentierbar. Aus dem Bereich der Mathematik sind effiziente Polyeder-Algorithmen zur Berechnung der Menge der Lösungen eines Gleichungssystems bekannt [Che65, Che68]. Diese Algorithmen wurden für die Verifikation wiederentdeckt und von Fukuda und Prodon in der Effizienz verbessert [FP96]. Für die Analyse von Verzögerungen in synchronen Programmen wurde diese Methode zuerst von Halbwachs eingesetzt und in einer allgemein verwendbaren Polyederbibliothek implementiert [Hal93, HPR97]. Im folgenden werden die Basisprinzipien der zugrundeliegenden Datenstruktur vorgestellt.

4.3.2.1 Datenstruktur

Belegungsmengen über die Variablen hybrider Automaten werden durch Polyeder repräsentiert. Ist die Menge konvex, genügt ein konvexes Polyeder, sonst wird die Menge durch eine implizite Vereinigung mehrerer konvexer Polyeder dargestellt. Jedes d-dimensionale konvexe Polyeder $P \subseteq \mathbb{R}^d$ hat zwei Repräsentationen:

- Das Polyeder P ist die Menge der Lösungen eines **Ungleichungssystems** $(A, \bar{b})$ mit $P = \{\bar{x} \in \mathbb{R}^n \mid A \cdot \bar{x} \geq \bar{b}\}$, wobei A eine $m \times d$-Matrix und $\bar{b}$ ein Vektor der Dimension m ist.
- Das Polyeder P ist die konvexe Hülle eines **Erzeugendensystems** (V, R, L), wobei die drei Komponenten endliche Mengen von Vektoren der Dimension d für Punkte (V), Strahlen (R) und Geraden (L) sind, so daß
$$P = \left\{ \sum_{v_i \in V} \lambda_i \cdot v_i + \sum_{r_j \in R} \mu_j \cdot r_j + \sum_{l_k \in L} \nu_k \cdot l_k \;\middle|\; \lambda_i \geq 0, \mu_j \geq 0, \sum_i \lambda_i = 1 \right\}.$$

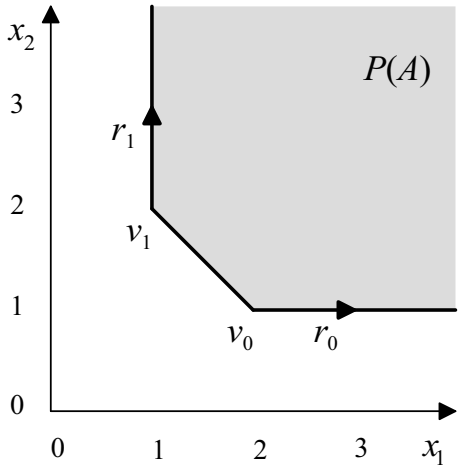


Abbildung 4.13: Polyederdarstellung der Menge $x_1 \geq 1 \wedge x_2 \geq 1 \wedge x_1 + x_2 \geq 3$.

Somit wird das in Abbildung 4.13 dargestellte Polyeder sowohl durch die Lösungen eines Ungleichungssystems

$$P = \left\{ (x_1, x_2) \left| \begin{pmatrix} x_1 & & \geq & 1 \\ & x_2 & \geq & 1 \\ x_1 & + \; x_2 & \geq & 3 \end{pmatrix} \right. \right\} \text{ als auch durch ein Erzeugendensystem}$$

(V, R, L) mit $V = \left\{ v_0 = \binom{2}{1}, v_1 = \binom{1}{2} \right\}, R = \left\{ r_0 = \binom{1}{0}, r_1 = \binom{0}{1} \right\}, L = \emptyset$ repräsentiert.

Diese beiden Repräsentationen sind gegenseitig eindeutig bestimmt. Es existieren effiziente Algorithmen für die Transformation von einer Repräsentation in die andere. Diese Algorithmen minimieren gleichzeitig die Repräsentation. Durch die Darstellung eines Objektes durch zwei verschiedene Repräsentationen erhielt diese Methode von Motzkin et al. den Namen **Double Description Method (DDM)** [MRTT53].

In der Rabbit-Implementierung der DDM-Repräsentation wird eine einfachere Darstellung der beiden Repräsentationen mittels **homogener Koordinaten** verwendet. Durch die Hinzunahme einer zusätzlichen Dimension z_h für die Konstanten und Definition des Polyeders als Schnittmenge eines Kegels[5] mit der zusätzlichen Ebene $z_h = 1$ entsteht die im folgenden beschriebene vereinfachte Repräsentation. Im Ungleichungssystem steht so auf der rechten Seite immer der Nullvektor und im Erzeugendensystem sind nur Strahlen notwendig [FP96]. Die im o. g. Beispiel angeführte Region (mit dem in Abbildung 4.13 gezeigten Polyeder) führt zu dem in der Abbildung 4.14 auf der nächsten Seite aufgeführten Kegel. Durch den Schnitt mit der Ebene $z_h = 1$ entsteht die in Abbildung 4.15 auf der nächsten Seite dargestellte Schnittfläche.

[5]Der Begriff "Kegel" bezeichnet hier nicht einen runden Kreiskegel, sondern eine geometrische Figur, ähnlich einer unregelmäßigen Pyramide, die von dessen Spitze im Koordinatenursprung ausgehend durch Strahlen begrenzt wird (unendliche Ausdehnung, ohne Grundfläche).

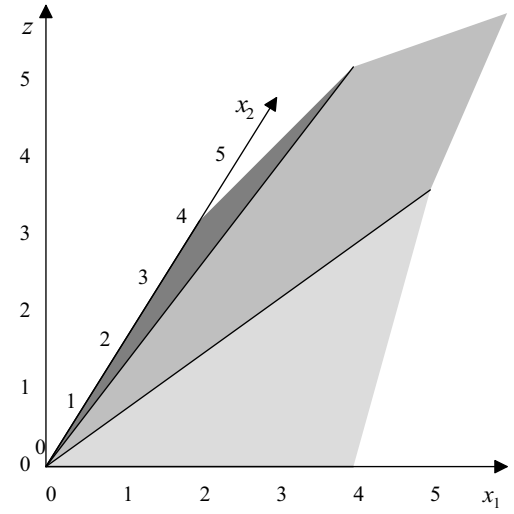


Abbildung 4.14: Kegel zur Repräsentation der Menge aus Abbildung 4.13.

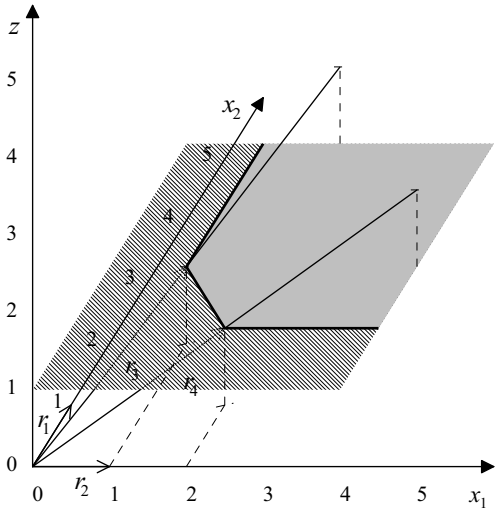


Abbildung 4.15: Schnitt des Kegels mit der zusätzlichen Ebene $z = 1$ führt zum gewünschten Polyeder.

Definition 4.24 *Ein Paar (A, R) reellwertiger Matrizen A und R wird* **Double Description Pair (kurz: DD-Paar)** *genannt, falls gilt:*

$$A \cdot \bar{x} \geq \bar{0} \quad \textit{gdw.} \quad \bar{x} = R \cdot \bar{\lambda} \textit{ mit } \bar{\lambda} \geq \bar{0}$$

Für ein d-dimensionales Polyeder ist A eine $m \times d$-Matrix und R eine $d \times n$-Matrix, $\bar{x} \in \mathbb{R}^d, \bar{\lambda} \in \mathbb{R}^n, \bar{\lambda} \geq \bar{0}$ gdw. $\forall i \in \{1, \ldots, n\} : \lambda_i \geq 0$. Das Polyeder $P \subseteq \mathbb{R}^d$ wird einerseits von A repräsentiert als

$$\left\{ \bar{x} \in \mathbb{R}^d \middle| \, A \cdot \bar{x} \geq \bar{0} \right\}$$

und andererseits von R als

$$\left\{ \bar{x} \in \mathbb{R}^d \middle| \, \bar{x} = R \cdot \bar{\lambda}, \bar{\lambda} \geq \bar{0} \right\}.$$

Die Matrix A wird **Repräsentationsmatrix**, die Matrix R wird **Erzeugendenmatrix** genannt. Diese Namen spiegeln die Intuition wider, daß mit der Matrix A geprüft wird, ob sich ein Vektor innerhalb des Polyeders befindet, d. h. vom Polyeder repräsentiert wird, oder daß mit der Matrix R ein beliebiger Vektor erzeugt werden kann, bzw. durch das Hinzunehmen eines Vektors das Polyeder erweitert werden kann.

Jeder Spaltenvektor der Erzeugendenmatrix von P liegt im Polyeder P und jeder Vektor in P ist eine nicht-negative Linearkombination der Spaltenvektoren von R.

Der folgende Satz von Minkowski stellt die Existenzbedingung für das DD-Paar sicher.

Satz 4.10 (Minkowski) *Für jede reelle $m \times d$-Matrix A existiert eine reelle $d \times n$-Matrix R, so daß (A, R) ein DD-Paar ist. Das durch A repräsentierte Polyeder ist endlich erzeugt von R.*

Beweis: Der Beweis erfolgt durch Angabe eines terminierenden Algorithmus [FP96]. Die Schwierigkeit an dem Beweis ist, daß die Spaltenanzahl n von R eine endliche Zahl sein muß; sonst könnte eine triviale Erzeugendenmatrix einfach alle Vektoren des Polyeders enthalten. □

Ein Satz bzw. ein Algorithmus für die Rückrichtung wird (obwohl vorhanden) nicht benötigt, da nach dem folgenden Lemma gilt [Zie95, Far02]:

Lemma 4.11 (Farkas) *(A, R) ist ein DD-Paar gdw. (R^T, A^T) ein DD-Paar ist.*

Das Paar (R^T, A^T) wird die zu (A, R) **duale Repräsentation** genannt. Die Rückrichtung des Satzes, d. h. die Berechnung einer Repräsentationsmatrix A zu einer gegebenen Erzeugendenmatrix R, kann gelöst werden, indem für die transponierte Erzeugendenmatrix R^T die entsprechende Erzeugendenmatrix A^T berechnet wird. Durch Transponieren entsteht die gewünschte Repräsentationsmatrix. Da die Spaltenvektoren in der Erzeugendenmatrix genau den sogenannten extremen Strahlen entsprechen, ist dieses Problem auch bekannt unter dem Namen **Extrem Ray Enumeration Problem** bzw. **Konstruktion eines minimalen Erzeugendensystems**.

4.3.2.2 Beispiel

Im folgenden wird die Kegelrepräsentation der Menge $P = \{(x_1, x_2) \mid x_1 \geq 1 \wedge x_2 \geq 1 \wedge x_1 + x_2 \geq 3\}$ illustriert (Polyederdarstellung in Abbildung 4.13 auf Seite 132). Durch Hinzunahme einer zusätzlichen Ebene wird für diese konvexe Menge der Kegel in den Abbildungen 4.14 und 4.15 einerseits durch die Repräsentationsmatrix

$$A = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & -3 \end{pmatrix}$$

definiert, wobei die Zeilenvektoren gerade die Normalenvektoren der den Kegel begrenzenden Flächen sind[6]. Die im Koordinatenursprung (der Kegelspitze) beginnenden Strahlen entlang der den Kegel begrenzenden Schnittgeraden bilden das zugehörige Erzeugendensystem. Somit wird der Kegel andererseits auch durch die Erzeugendenmatrix

$$R = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

repräsentiert, wobei die Spaltenvektoren die den Kegel begrenzenden Strahlen sind.

[6]Der Vektor $(0, 0, 1)$ müßte hier hinzugenommen werden, um den Kegel nach unten hin abzugrenzen, da jedoch nur die Schnittfläche mit $z_h = 1$ interessiert, wird dieser Vektor stillschweigend vorausgesetzt.

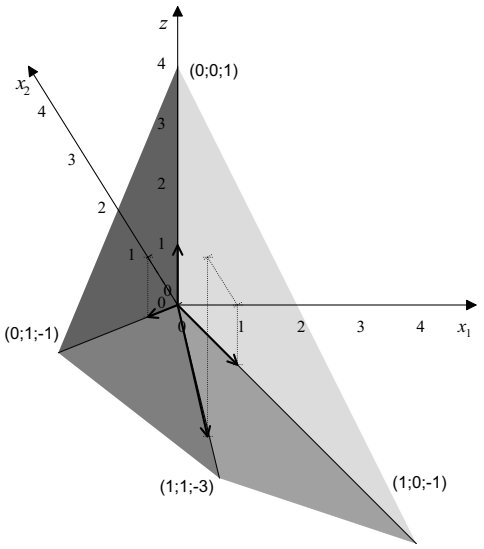


Abbildung 4.16: Kegel zur Repräsentation der zu Abbildung 4.14 dualen Menge in 60°-Projektion.

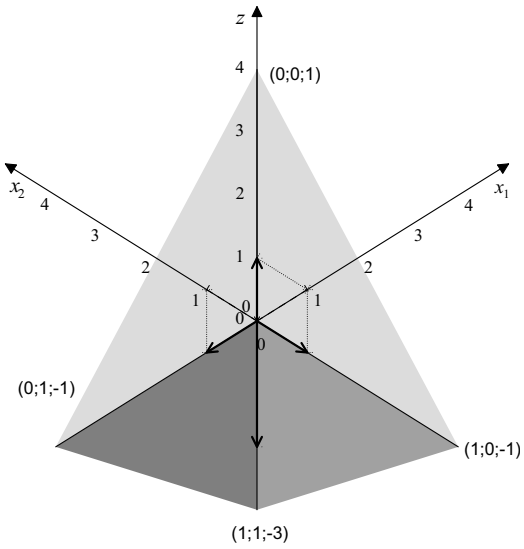


Abbildung 4.17: Kegel aus Abbildung 4.16 in isometrischer Projektion.

Der zu P duale Kegel wird durch die Repräsentationsmatrix

$$A_D = R^T = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix}$$

definiert, wobei die Spaltenvektoren der Matrix R hier als Normalenvektoren der den dualen Raum begrenzenden Flächen interpretiert werden und somit bei A_D als Zeilenvektoren auftreten ($x_2 \geq 0 \wedge x_1 \geq 0 \wedge x_1 + 2 \cdot x_2 \geq -1 \wedge 2 \cdot x_1 + x_2 \geq -1$). In den Abbildungen 4.16 und 4.17 wird der zu P duale Kegel in zwei verschiedenen Sichtweisen dargestellt[7]. Durch den Algorithmus zur Berechnung der Erzeugendenvektoren kann die Erzeugendenmatrix

$$R_D = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ -1 & -1 & -3 & 1 \end{pmatrix}$$

zur Beschreibung des dualen Raumes mittels begrenzender Strahlen angegeben werden. Durch Transponieren von R_D entsteht wieder die ursprüngliche Repräsentationsmatrix $A = R_D^T$.

Anmerkung: Für eine ausführlichere Beschreibung der graphischen Repräsentation mit einer zusätzlichen Koordinate für eine ”Hyperebene” (homogene Koordinate) siehe [FvDFH94]. Eine Bibliothek, die Datenstrukturen und Algorithmen für die Arbeit mit

[7] Es macht keinen Sinn, sich in der dualen Repräsentation die Ebene $z_h = 1$ vorzustellen, da diese Ebene nichts mit dem ursprünglichen Polyeder zu tun hat.

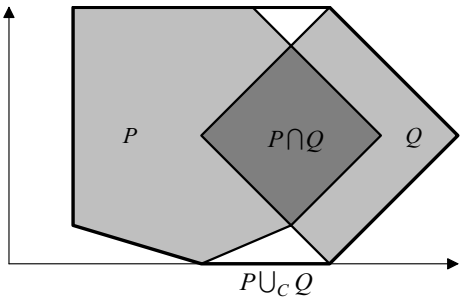


Abbildung 4.18: Durchschnitt und Vereinigung zweier Polyeder.

Polyedern enthält, und eine tiefgreifende Dokumentation der Methode wurde am IRISA-Institut erarbeitet [Wil93].

4.3.2.3 Operationen

Die Double Description Method unterstützt die folgenden Operationen, die für die Erreichbarkeitsanalyse von Bedeutung sind:

Durchschnitt: Der Durchschnitt zweier konvexer Polyeder P und Q ist das konvexe Polyeder $P \cap Q$, das sowohl alle Bedingungen von P als auch alle Bedingungen von Q erfüllt und durch Vereinigung der Ungleichungen, d. h. der Zeilen der Repräsentationsmatrizen, entsteht (siehe Abbildung 4.18).

Konvexe Hülle (konvexe Vereinigung): Die konvexe Hülle $\cup_C$ zweier konvexer Polyeder P und Q ist das kleinste konvexe Polyeder, das sowohl P als auch Q enthält und durch die Vereinigung der Erzeugenden, d. h. der Spalten der Erzeugendenmatrizen, entsteht. Die konvexe Hülle wird als obere Approximation der Vereinigung benutzt, da die Vereinigung konvexer Polyeder im allgemeinen nicht konvex ist (siehe Abbildung 4.18).

Test auf Leerheit: Ein Polyeder ist leer genau dann, wenn im Erzeugendensystem die Menge der Punkte leer ist bzw. kein Strahl existiert, der die zusätzliche Ebene $z_h = 1$ schneidet.

Teilmengenbeziehung und Mengengleichheit: Ein Polyeder P mit der Erzeugendenmatrix R ist enthalten in einem Polyeder Q mit der Repräsentationsmatrix A genau dann, wenn für alle i mit $1 \leq i \leq n$ gilt: $A \cdot (r_{1i}, r_{2i}, \ldots, r_{di}) \geq \bar{0}$, wobei $(r_{1i}, r_{2i}, \ldots, r_{di})$ der i-te Spaltenvektor von R ist. Die Gleichheit muß über gegenseitiges Enthaltensein nachgewiesen werden.

Nachdem ausgeführt worden ist, wie Mengen von Konfigurationen mit der DDM-Repräsentation dargestellt werden, wird im folgenden die Nachfolgerberechnung im Erreichbarkeitsalgorithmus erklärt. Das hier vorgestellte Vorgehen wurde auch in der Werkzeugimplementierung HyTech verwendet [HHWT95a].

Wie bei Timed Automata besteht die Nachfolgerberechnung bei hybriden Automaten aus der Anwendung von Zeit-Transitionen (dem Vergehen von Zeit) und von diskreten Transitionen (Zustandsübergängen):

Zeit-Transitionen. Für die Berechnung der Zeit-Nachfolger wird die Repräsentation mittels Erzeugendenmatrix benutzt. Die Strahlen der Erzeugendenmatrix der Ableitungen definieren die Richtung, in der sich die Werte der Variablen entsprechend der Ableitungen verändern. Diese Strahlen werden zu den Strahlen der Erzeugendenmatrix der Variablenbelegung hinzugefügt[8]. Dadurch wird die Entwicklung der Variablenwerte hin zum Unendlichen berücksichtigt. Da nur bei erfüllter Zustands-Invariante Zeit vergehen darf, wird die Belegungsmenge vor und nach dieser Operation jeweils auf die Invariante beschränkt.

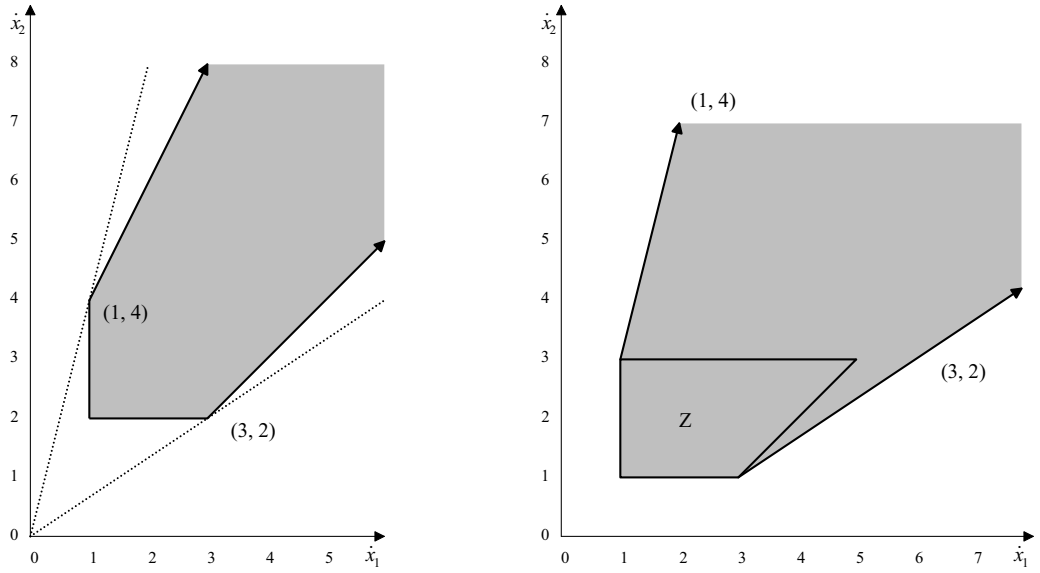


Abbildung 4.19: Berechnung der Zeit-Nachfolger (nach [HHWT95a]).

In Abbildung 4.19 wird als Beispiel ein System mit zwei Variablen x_1 und x_2 betrachtet. Das Ableitungspolyeder sei gegeben durch $\dot{x_1} \geq 1 \wedge \dot{x_2} \geq 2 \wedge \dot{x_2} \geq \dot{x_1} - 1 \wedge \dot{x_2} \leq 2\dot{x_1} + 2$ (grauer Bereich im linken Koordinatensystem). Zur Berechnung der Zeit-Nachfolger zu einem konvexen Polyeder Z werden zum Erzeugendensystem von Z die Vektoren $(1, 4)$ und $(3, 2)$ hinzugenommen, wie der graue Bereich im rechten Koordinatensystem der Abbildung 4.19 verdeutlicht.

Diskrete Transitionen. Für einen Zustandsübergang e ist die Transitionsrelation $\rightarrow_e$ definiert durch das Polyeder $\{v \in Val(X \cup X') \mid v|_X \in guard(e) \wedge \exists u \in update(e)(v|_X) : v|_X \lhd u = v|_{X'}\}$. Um die Nachfolger einer konvexen Region Z zu berechnen, wird zunächst die Spaltenanzahl der Z darstellenden Repräsentationsmatrix verdoppelt für die Variablen aus X'. Der Durchschnitt dieser beiden Mengen wird berechnet, indem alle Zeilen der Repräsentationsmatrix von $\rightarrow_e$ zur Repräsentationsmatrix von Z hinzugefügt werden. Anschließend erfolgt die Existenzquantifizierung der Variablen aus X und eine

[8] Punkte, also Strahlen, die die $z_h = 1$-Ebene schneiden, werden zu Strahlen gemacht, indem die z-Komponente auf den Wert 0 gesetzt wird.

Umbenennung aller Variablen, so daß die entstandene Menge eine Menge von Variablenbelegungen über X ist. Nichtkonvexe Mengen werden mittels Mengen von Polyedern behandelt.

4.3.2.4 Behandlung inaktiver Variablen

In Abschnitt 3.5.6 wurde die Technik der Sonderbehandlung inaktiver Uhren für Timed Automata behandelt. Durch das Eliminieren unnötiger Werte von Uhren, die für das Verhalten eines Modells zeitweise keine Rolle spielen, konnte eine wesentliche Performanceverbesserung erzielt werden. Diese Strategie läßt sich auch auf die DDM-Repräsentation für hybride Modelle übertragen, wobei statt Uhren nun allgemein Variablen und zusätzlich die Ableitungsdefinitionen betrachtet werden müssen.

Definition 4.25 *Eine Variable ist in einem Zustand* **aktiv***, wenn ihr Wert Einfluß auf das zukünftige Verhalten des Automaten hat, d. h.*

- *wenn die Variable in der Invariante des Zustands, dem Wächter eines ausgehenden Zustandsübergangs oder ungestrichen in der Bedingung für die Wertveränderung eines ausgehenden Zustandsübergangs auftritt, oder*
- *wenn die Variable im Zielzustand eines ausgehenden Zustandsübergangs aktiv ist und von diesem Zustandsübergang nicht auf einen expliziten neuen Wert gesetzt wird (die Variable kommt nicht im Definitionsbereich eines Updates vor oder ihr Wert ist abhängig vom Wert vor diesem Übergang).*

Um auch bei der DDM-Repräsentation unnötigen Speicher- und Rechenzeit-Aufwand zu vermeiden, der durch inaktive Variablen verursacht wird, werden folgende Richtlinien für hybride Automatenmodelle festgelegt:

Ableitungen: Ableitungen werden so wenig wie möglich eingeschränkt, damit die Ableitungsmatrizen der Zustände möglichst klein sind. In Zuständen, in denen die Variablenwerte keine Rolle spielen, bleibt die Ableitung der inaktiven Variable beliebig. Dadurch sind nach einer Zeit-Transition alle möglichen Werte dieser Variable erreicht. Die syntaktische Deklaration einer Variable als CLOCK führt z. B. dazu, daß der Modellierer die Ableitungen nicht definieren muß, da diese überall gleich 1 sind. Damit wird aber die Variable in einem Zustand ggf. unnötig eingeschränkt, obwohl die Werte der Variable nicht interessieren.

Wertveränderung: Alle zu einen Zustand mit einer inaktiven Variable führenden Zustandsübergänge setzen diese Variable auf einen festen Wert (z. B. 0), damit in diesem Zustand weniger unterschiedliche Belegungen der Variable in Abhängigkeit der Belegung der Variable im Vorgängerzustand auftreten.

Auch diese Richtlinien können durch statische Analyse und anschließender automatischer Transformation eingehalten werden und zu Performanceverbesserung führen.

Meßergebnisse. Zum Nachweis der Effizienzsteigerung der Technik werden für das hybride Modell von Fischers Protokoll mit 5 Prozessen drei Varianten zur Behandlung

Modell-Variante	CLOCK	ANALOG	ANALOG mit Reset
Verifikationszeit	360	279	183
Speicherbedarf (MB)	94,1	87,2	77,2
Anzahl Iterationen bis Fixpunkt	29	27	27
Anzahl geschalteter diskreter Transitionen	6565	5805	5805
Anzahl Polyeder in der Erreichbarkeitsmenge	15496	12001	12001

Tabelle 4.1: Performanceverbesserung durch Sonderbehandlung inaktiver Variablen.

der Uhren untersucht. Die Meßergebnisse werden in Tabelle 4.1 aufgeführt. In der ersten Spalte der Tabelle wird zur Deklaration der Uhr der Datentyp 'CLOCK' verwendet, was bedeutet, daß in allen Zuständen die Ableitung dieser kontinuierlichen Variable auf den Wert 1 festgelegt ist. Diese Festlegung erfolgt implizit durch die Deklaration als Uhr.

Bei der zweiten Variante des Modells wird der Datentyp der Uhr auf 'ANALOG' verändert, wobei in den Zuständen 'Assign' und 'Wait' die Ableitung im Modell explizit auf den Wert 1 gesetzt wird. In den Zuständen 'Uncritical' und 'Critical', in denen der Wert der Uhr keine Rolle spielt, wird die Ableitung nicht festgelegt, d. h. es sind alle Werte für die Ableitung gültig (volle Menge) und nach der ersten Zeit-Transition beinhaltet die dem Zustand zugeordnete Region alle Werte für die Uhr (volle Menge). Dadurch benötigt die Zeit-Transition nicht nur weniger Speicherplatz zur Repräsentation ihrer Matrix (Verifikationszeit in der Tabelle), sondern die Operation kann auch schneller ausgeführt werden. Außerdem werden Kombinationen für die Konfigurationen beim Eintritt in einen solchen Zustand mit inaktiver Uhr durch die Zeit-Transition eliminiert, da diese immer gleich eine volle Region möglicher Variablenwerte berechnet (siehe Anzahl Polyeder in der Erreichbarkeitsmenge in der Tabelle). Dies spiegelt sich in den Experimenten auch in der kleineren Anzahl geschalteter diskreter Transitionen und Iterationen bis zum Erreichen des Fixpunktes wieder.

In der dritten Variante des Modells wird zusätzlich der Wert der Uhren beim Übergang in einen Zustand, in dem er keine Rolle spielt, auf einen festen Wert gesetzt. Dadurch wird die Operation zur Berechnung der diskreten Nachfolger einfacher: Zur Speicherung der dualen Repräsentation der Wertveränderung einer diskreten Transition ist für einen Ausdruck der Form $x' = 5$ weniger Speicherplatz erforderlich als für $x' = x$. Damit sind auch die Operationen bei der Nachfolgerberechnung weniger aufwendig und Rechenzeit sowie Speicherbedarf reduzieren sich nochmals.

4.4 Zusammenfassung

Lange Zeit wurden Systeme als reine reaktive Systeme modelliert, d. h. von der Zeitabhängigkeit der Reaktionen wurde abstrahiert. Mit der Einführung von Timed Automata und anderer Formalismen zur Zeit-Modellierung (z. B. SDL, Statecharts) wurde dazu übergegangen, die Zeitaspekte bei der Modellierung und der Verifikation zu berücksichtigen. Dies machte die formalen Methoden für die Praxis interessanter (siehe z. B. Protokoll-Engineering). Der nächste Schritt besteht darin, noch weniger zu abstrahieren und auch die hybriden Aspekte der Systeme bei der Modellierung zu betrachten.

Der Schwerpunkt dieses Kapitels liegt in der Realisierung einer Modellierungsnotation, die den ersten Schritt zur Bewältigung großer *hybrider* Systeme darstellt. Dazu wurde der CTA-Formalismus um hybride Automaten erweitert. Weiterhin werden Techniken zur Darstellung von Konfigurationsmengen und zur Verifikation hybrider Automaten beschrieben, um die Repräsentation in das im nächsten Kapitel behandelte Werkzeug integrieren zu können. Damit wurde die Forderung 4 auf Seite 19 erfüllt, die Konzepte zur strukturierten Modellierung auf hybride Systeme zu übertragen. Es wird gezeigt, daß die Integration einer zusätzlichen Repräsentation in das Verifikationsframework Rabbit problemlos möglich ist.

Offenes Problem. Ansätze bzw. Ideen zur effizienten Analyse hybrider Automaten bestehen noch nicht. Der Schwerpunkt der Untersuchungen liegt derzeit noch bei der Verifikation von Realzeit-Systemen. Auch in diesem Kapitel wurde kein neues Verfahren für die Verifikation hybrider Automaten eingeführt, da der Zweck der Betrachtung hybrider Systeme in dieser Arbeit in der Validierung der Modellierungskonzepte und der Überprüfung der Integrationsmöglichkeiten des Frameworks liegt. Auf Verbesserungspotential der hier verwendeten Repräsentation hybrider Automaten wird in Abschnitt 5.4.1.5 hingewiesen.

Kapitel 5

Validierung: Verifikationsframework Rabbit

Die in den Kapiteln 2 bis 4 erarbeiteten theoretischen Konzepte werden einer praktischen Validierung unterzogen. Dazu erfolgt eine Werkzeugimplementierung, in die alle eingeführten Konzepte integriert werden. Dieses Werkzeug dient vor allem dazu, Erfahrungen mit den in Kapitel 2 vorgeschlagenen Modellierungskonzepten zu sammeln und die verbesserte Performance der in Kapitel 3 dargestellten Techniken zur effizienten Verifikation nachzuweisen [Bey01c]. Weiterhin wird für dieses Werkzeug eine Architektur nach den Prinzipien eines Frameworks entworfen, um als Analyseplattform für verschiedene Modell-Repräsentationen zu dienen. Dadurch wird es möglich, die in Kapitel 4 vorgestellten Datenstrukturen zur Repräsentation von hybriden Automaten in das gleiche Werkzeug einzubetten.

Dieses Kapitel beschreibt die Architektur des Werkzeugs, die Eingabesprachen für Modellierung und Verifikation, sowie empirische Ergebnisse. Bei der Vorstellung der Architektur soll ein Eindruck davon vermittelt werden, wie bei der Entwicklung der Software vorgegangen wurde und wie die Repräsentationen aus Kapitel 3 und 4 implementiert und integriert worden sind. Ein Überblick über die syntaktischen Möglichkeiten und über den Umfang der Analyseoperationen folgt in den zwei Abschnitten zur Eingabesprache.

Im Abschnitt über die empirischen Ergebnisse wird die Modellierung und Verifikation von Realzeit- und hybriden Systemen an einigen Fallbeispielen aus der Literatur vorgestellt. Für die Beispiele aus dem Bereich der hybriden Systeme werden die CTA-Modelle und Analyseresultate beschrieben. Darüberhinaus wird für die fünf Beispiele aus dem Bereich der Realzeit-Modellierung jeweils ein Performance-Vergleich mit anderen bekannten Werkzeugen für denselben Anwendungsbereich durchgeführt.

5.1 Architektur

In diesem Abschnitt wird der Aufbau des Werkzeugs und das Zusammenspiel der Komponenten und Bibliotheken sowie die Funktionalität der wichtigsten Komponenten des Gesamtsystems vorgestellt: Scanner, Parser, Kontextprüfung, Repräsentationsschnittstelle,

BDD-Repräsentation, DDM-Repräsentation, Transformation des Modell-Syntaxbaumes in eine der Repräsentationen, repräsentationsunabhängiger Analyseinterpreter. Dabei spiegelt sich der Framework-Gedanke in der Konzeption von Rabbit wider.

Die Werkzeugimplementierung Rabbit wurde in der Programmiersprache C++ erstellt. Dabei wurde für den flexiblen Entwurf insbesondere von den Möglichkeiten einer objektorientierten Programmiersprache Gebrauch gemacht. Das Gesamtsystem besteht aus den folgenden Komponenten:

- Ein Compiler-Frontend, um die Modellbeschreibung in der textuellen Notation einzulesen und eine Hierarchie von CTA-Modulen im Speicher zu erzeugen (abstrakter Syntaxbaum).

- Je eine Repräsentationsbibliothek, welche die Datenstrukturen und Algorithmen bereitstellt, die für die jeweilige symbolische Repräsentation erforderlich sind. In der derzeit verfügbaren Version von Rabbit werden Repräsentationen mittels Binary Decision Diagrams und Double Description Method angeboten.

- Ein Interpreter für die Verifikationsnotation, die verschiedene Analyseanweisungen für die Erreichbarkeits- und Verfeinerungsanalyse anbietet. Der Interpreter steuert die Transformation der für die Analyse benötigten Modellteile in die erforderliche Repräsentation und delegiert die erforderlichen Verifikationsberechnungen an die entsprechenden Repräsentationsbibliotheken.

Der Ablauf bei der Benutzung des Werkzeugs wird in Abbildung 5.1 auf der nächsten Seite dargestellt. Für die Verifikationsaufgabe gibt es zwei Eingaben: eine Beschreibung des Modells als CTA-Module und eine Spezifikation der zu überprüfenden Sicherheitseigenschaften als Analyseanweisungen. Vor der eigentlichen Analyse überprüft das Werkzeug, ob das Modell die Kontextbedingungen sowie die in Abschnitt 2.3 geforderten Bedingungen für die Kompatibilität von Modulen in der Modulstruktur erfüllt.

Nachdem sichergestellt ist, daß der abstrakte Syntaxbaum ein konsistentes Systemmodell darstellt, wird die Transformation in die gewünschte Repräsentation vorgenommen. Für die Berechnungen werden entsprechend der semantischen Grundlagen Produktautomaten verwendet. Um diese zu erzeugen, wird die hierarchische Struktur kommunizierender Module transformiert in eine flache Menge von Automaten (’flat model’), deren Semantik durch das Transitionssystem für den Produktautomaten gegeben ist. Diese Menge von Automaten beinhaltet keine Gültigkeitsbereiche, spezielle Datentypen für Variablen, Zugriffstypen für Variablen und Signale und Abstraktionsebenen mehr. Die durch zusätzliche notationellen Elemente gegebenen Eigenschaften wurden in der weiter oben erwähnten Konsistenzüberprüfung benötigt und sind auf der Ebene der Transitionssysteme nicht mehr erforderlich. Nach diesem Schritt kann der abstrakte Syntaxbaum der für die Verifikation benötigten Modellteile in die Repräsentation überführt werden.

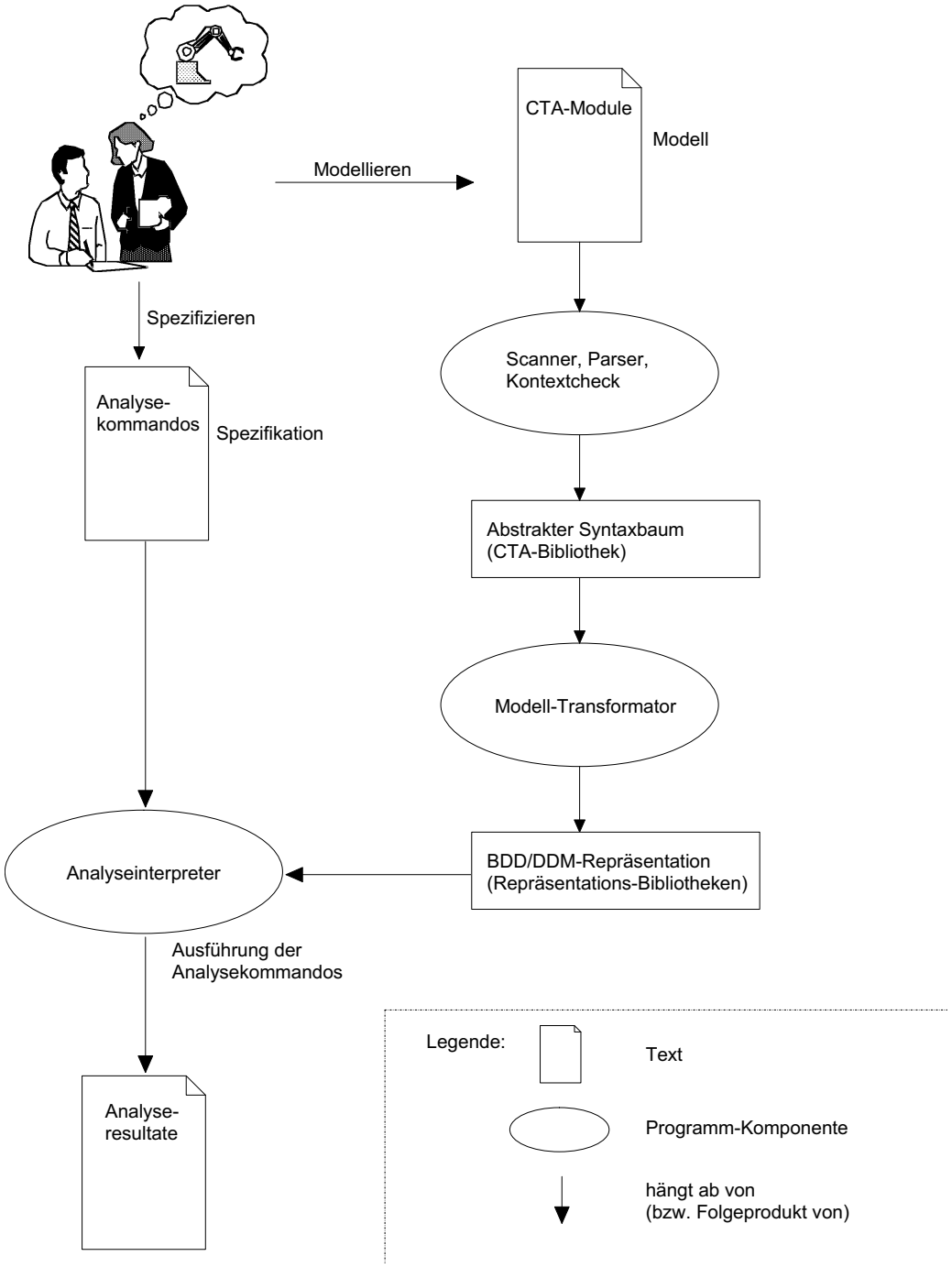


Abbildung 5.1: CTA-Analysewerkzeug.

Die Repräsentation wird zur Laufzeit festgelegt, entweder explizit mittels Analyseanweisungen oder durch eine Analyse der im Modell enthaltenen Automaten, d. h. es wird die für das Modell am besten geeignete Repräsentation ausgewählt. Mit dieser Repräsentation können alle zur Analyse erforderlichen Berechnungen durchgeführt werden. Der Analyseinterpreter führt die einzelnen Analyseaufgaben nacheinander in der vorgegebenen Reihenfolge aus, indem er entsprechend der Repräsentation (durch Polymorphie) die jeweils passenden Algorithmen aufruft.

5.1.1 Architekturentscheidungen und Entwurfsprinzipien

In diesem Abschnitt werden einige der für die Entwicklungsarbeiten wichtigen Entscheidungen begründet. Insgesamt haben sich die einzelnen Entscheidungen und Prinzipien als sinnvoll erwiesen, da die Werkzeugimplementierung während der gesamten Projektlaufzeit von vier Jahren immer eine stabile Entwicklungsbasis darstellte und Weiterentwicklungen aufgrund der flexiblen Architektur problemlos möglich waren. Dieses Entwicklungsprojektes umfaßt Arbeitsergebnisse der folgenden Quantität: 118 Klassen, 167 Dateien Source-Code C++ (31.852 Zeilen) und 382 Dateien mit CTA-Beispielmodellen als Testbasis (178.402 Zeilen).

Werkzeuge und Bibliotheken. Um die *Portierbarkeit* des Systems auf verschiedene Plattformen zu gewährleisten, wurde zu Beginn des Projektes entschieden, für die Entwicklungsarbeiten die objektorientierte Programmiersprache C++ [Str98] und ausschließlich die in der GNU-Distribution verfügbaren Compiler [Sta95b] und Bibliotheken sowie zusätzlichen Programme zu verwenden. Wegen Kompatibilitätsproblemen der Programmierumgebungen von Borland und Microsoft wurden diese nicht verwendet, sondern auf die herkömmlichen Programmierwerkzeuge Make [SM95], Emacs [Sta95a], etc. zurückgegriffen.

Bei der Entwicklung des *Compiler-Frontends* wurde mit den GNU-Versionen der Compiler-Werkzeuge Lex und Yacc gearbeitet. Für die Generierung des Scanners wurde Flex [Pax95] und für die Generierung des Parsers Bison [DS95] eingesetzt. Für allgemeine Datenstrukturen und Algorithmen wie Strings, Vektoren, etc. wurde am Anfang des Projekts mit der Standard-C++-Bibliothek [Lea92] gearbeitet. Als jedoch im Jahr 1998 die *Standard Template Library* [SL95] in ausgereiftem und stabilem Zustand vorlag, wurden sämtliche Benutzungen auf die neue Bibliothek übertragen.

Entwurfsmuster. Um beim Entwurf des objektorientierten Klassensystems für optimale Flexibilität und Wartbarkeit zu sorgen, wurde häufig auf die in den Entwurfsmustern [GHJV93, GHJV94] verankerten Erfahrungen zurückgegriffen. So wurde bei den Klassen des Syntaxbaumes (Modell und Analyseanweisungen) in mehrfacher Hinsicht das Muster *Composite* eingesetzt, sowohl bei der Konstruktion der Repräsentation und bei der Ausgabe des Modells, als auch beim Abarbeiten aller Analyseanweisungen. Bei der Wiederverwendung von bestehenden Klassen wurde jeweils eine *Adapter*-Klasse zwischengelegt, um eine Anpassung zu ermöglichen (Klassen-Adapter: String, Vector, Object; Objekt-Adapter: BDD). Für den flexiblen Umgang mit verschiedenen Repräsentationen wurde die Basis-Idee einer *Abstract Factory* verwendet: dem Analyseinterpreter ist nicht bekannt, auf welcher konkreten Repräsentation die Analysen zur Laufzeit tatsächlich ab-

laufen. Das Erzeugen der DDM- bzw. BDD-Repräsentation wird von *Builder*-Klassen vorgenommen. Im abstrakten Syntaxbaum wird oft das Muster der *Bridge* angewendet, wenn für eine Regel in der Grammatik mehrere 'echte' (nicht-ϵ) Alternativen beim Ersetzen des Metasymbols in Frage kommen. Damit wird über eine abstrakte Schnittstelle Polymorphie ermöglicht.

Eigenentwicklung versus Verwendung fremder Software. Bei der Planung der Entwicklungsarbeiten für die Repräsentationen stellte sich zunächst die Frage, ob zur Reduzierung des Programmieraufwands Teile aus bestehenden Softwarepaketen anderer Forschungsgruppen verwendet werden könnten, da bei dem Entwicklungsprojekt die Personalressourcen auf einen wissenschaftlichen Mitarbeiter und Studenten beschränkt waren. Um alle Ziele erreichen zu können, kommt eine Nutzung von Fremdwerkzeugen nicht in Frage, und zwar aus den folgenden Gründen:

- Für die Arbeit mit Polyedern liegen nur prototypische, in der Programmiersprache C geschriebene, Implementierungen vor, z. B. die von HyTech genutzte. Es ist unklar, in welchem Maße die Qualität dieser Implementierungen sichergestellt ist. Der Quellcode der Software liegt zwar meistens vor, ist aber mangels Dokumentation der Schnittstellen nicht wiederverwendbar. Es erweist sich als schwierig, aus einer solchen Implementierung die Repräsentation vom Rest des Programmes (Compiler-Frontend, Analyseinterpreter) zu trennen. Auch um eine einheitliche Architektur verwirklichen zu können, ist es vorteilhaft, die entscheidenden Komponenten selbst zu entwickeln.

- Die Eigenentwicklung der Bibliotheken für die Datenstrukturen und Algorithmen ermöglicht einen Überblick über die Funktionsweise und eine Einsicht in die technischen Details der Bibliothek. Dabei kommt es zu einem tieferen Verständnis des Verhaltens der Algorithmen. Der Entwickler bekommt bereits bei der Programmierung Hinweise zur Performanceverbesserung.

- Für die Arbeit mit BDDs stehen mehrere qualitativ hochwertige Softwarepakete zur Verfügung. Bei einer Eigenimplementierung kann jedoch unnötiger Overhead einer solchen Bibliothek mit dem Zweck der Allgemeinverwendbarkeit vermieden werden. In den verfügbaren Bibliotheken sind bestimmte erweiterte Techniken zur Performanceverbesserung integriert. Soll bei einer Performanceanalyse die Wirkungsweise einer bestimmten in dieser Arbeit behandelten Technik untersucht werden, müssen diese Techniken ausgeschaltet werden können, damit es nicht zur Überlagerung von verschiedenen Parametern kommt, die die Performance beeinflussen.

- Durch das Vertrautsein mit der benutzten eigenen, kleinen BDD-Bibliothek ist es möglich, in die Details der Bibliothek eingreifende Veränderungen vorzunehmen, z. B. um ein besseres Verständnis des Verhaltens von BDDs bei bestimmten Operationen zu erlangen oder den Einfluß einiger Performanceparameter zu beobachten (Monitoring).

- Durch die strikte Trennung der Repräsentation der Konfigurationsmengen und Transitionssysteme von den anderen Komponenten des Werkzeugs wird die Integration weiterer Repräsentationen ermöglicht und das Werkzeug kann in Folgeprojekten als Analyseframework für automatenbasierte Modelle fungieren; somit wird auch eine vergleichende Untersuchung verschiedener Repräsentationen ermöglicht.

5.1.2 Repräsentation von Konfigurationsmengen und Transitionssystemen

In Abschnitt 2.1 wurde bereits auf verschiedene Darstellungsformen der Beschreibung von Belegungsmengen hingewiesen. Im Zusammenhang mit dem Analyseframework sind insbesondere drei Repräsentationen von Interesse, für die auf den folgenden Seiten Entwurf und Implementierung vorgestellt werden:

Variablenbedingung: Diese *logischen Prädikate in allgemeiner Form*, jedoch ohne die Operatoren Existenz- und Allquantor, treten bei der syntaktischen Definition von Belegungsmengen für Invarianten, Ableitungen, Wächter und Wertveränderungen auf. Durch das Compiler-Frontend werden diese Variablenbedingungen eingelesen und im abstrakten Syntaxbaum wiedergegeben. Mit dieser Darstellung können jedoch keine effizienten Berechnungen durchgeführt werden, deshalb werden die folgenden beiden Repräsentationen von den Verifikationsalgorithmen benutzt.

Menge von Matrizen: Eine Belegungsmenge, die durch eine *Variablenbedingung in disjunktiver Normalform (DNF)* gegeben ist, läßt sich durch eine Menge von Matrizen repräsentieren, indem jede Konjunktion durch eine Matrix dargestellt wird.

Menge von BDD-Knoten: Eine *Variablenbedingung in If-Then-Else-Normalform (ITE)* beschreibt eine Menge von Belegungen als Shannon-Entwicklung. Die Datenstruktur zur Speicherung einer solchen ITE-Formel ist das Binary Decision Diagram, und die Shannon-Entwicklung einer Variable entspricht einem BDD-Knoten [Bry92].

Benötigte Datenstrukturen. Um die Erreichbarkeit von Konfigurationen überprüfen zu können, werden im wesentlichen zwei Datenstrukturen benötigt: eine zur Repräsentation von Transitionssystemen (in der Implementierung 'reprAutomaton' genannt) und eine zur Repräsentation von Konfigurationsmengen (in der Implementierung 'reprConfig' genannt). Entsprechend Abbildung 4.9 auf Seite 129 sind die folgenden elementaren Operationen für die Erreichbarkeitsanalyse zu unterstützen:
Transitionssystem:

- **Nachfolgerberechnung:** Für eine gegebene Menge von Konfigurationen wird von dieser Operation eine Menge von Folgekonfigurationen berechnet. Es gibt zwei verschiedene solcher Operationen: eine berechnet die Menge aller von der gegebenen Menge aus erreichbaren Konfigurationen, d. h. es wird eine Fixpunktiteration durchgeführt; die andere Operation berechnet die Menge der in einer gegebenen Anzahl von Schritten erreichbaren Konfigurationen, z. B. ein Schritt für die direkten Nachfolger.

- **Test auf Erreichbarkeit:** Für zwei gegebene Mengen von Konfigurationen (Startkonfigurationen und Zielkonfigurationen) berechnet diese Operation, ob ausgehend von der Menge der Startkonfigurationen durch Nachfolgerbildung eine Konfiguration aus der Menge der Zielkonfigurationen erreichbar ist. Bei dieser Operation muß nicht unbedingt die gesamte Menge erreichbarer Konfigurationen durchmustert werden, denn sobald eine Zielkonfiguration erreicht worden ist, steht das Ergebnis der Operation fest. Weiterhin ist es nicht erforderlich, die gesamte Menge der erreichbaren Konfigurationen im Speicher zu halten, da sie von der Operation nicht als Ergebnis zurückgegeben werden muß. Damit sind die Voraussetzungen für die sogenannte On-the-fly-Analyse erfüllt (siehe Abschnitt 3.5.4).

Konfigurationsmenge:

- **Vereinigung:** Für zwei gegebene Mengen von Konfigurationen wird die Menge berechnet, die alle Konfigurationen enthält, die in der einen oder der anderen gegebenen Menge enthalten sind.
- **Durchschnitt:** Für zwei gegebene Mengen von Konfigurationen wird die Menge berechnet, die alle Konfigurationen enthält, die in beiden gegebenen Mengen enthalten sind.
- **Differenzbildung:** Durch diese Operation wird die Menge der Konfigurationen berechnet, die in der ersten, nicht jedoch in der zweiten gegebenen Menge enthalten sind.
- **Komplementbildung:** Für eine gegebene Menge von Konfigurationen wird die Menge aller Konfigurationen berechnet, die nicht in der gegebenen Menge enthalten sind.
- **Existenzquantifizierung:** Für eine Variable und eine Menge von Konfigurationen (hier speziell als Menge von Variablenbelegungen zu interpretieren) wird diejenige Menge berechnet, bei der die gegebene Variable durch das Anwenden des Existenzquantors eliminiert worden ist.
- **Test auf Teilmengenbeziehung:** Für zwei gegebene Mengen von Konfigurationen wird überprüft, ob alle in der zweiten Menge enthaltenen Konfigurationen auch in der ersten gegebenen Menge enthalten sind.
- **Test auf Leerheit:** Diese Operation überprüft, ob es sich bei einer gegebenen Menge von Konfigurationen um die leere Menge handelt.

Entwurf. Im folgenden wird ein Eindruck vom objektorientierten Grob-Entwurf vermittelt. Der Analyseinterpreter läßt sich vom Modelltransformator (Repräsentationen-Konstrukteur ctaAutomaton) entsprechend seiner Erfordernisse eine Repräsentation konstruieren. Diese Repräsentation stellt sämtliche Operationen des Transitionssystems für die Erreichbarkeitsanalyse (siehe auch Schnittstelle von reprAutomaton) und für das Rechnen mit Konfigurationsmengen (siehe auch Schnittstelle von reprConfig) zur Verfügung.

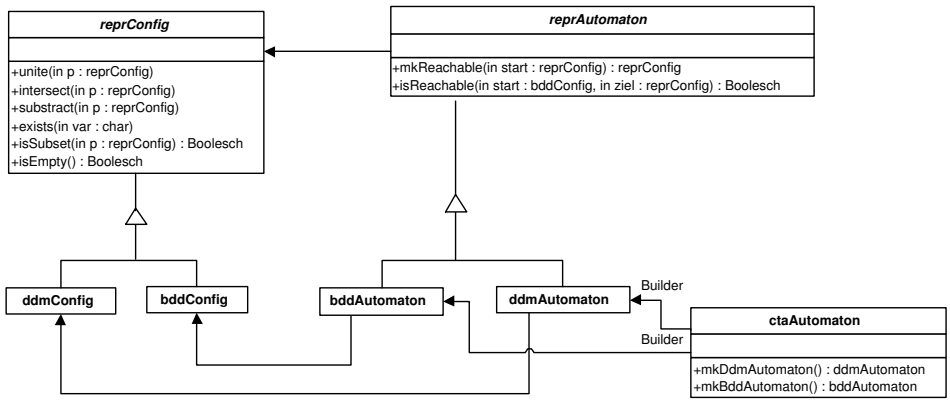


Abbildung 5.2: Design für die Abstraktion von der konkreten Repräsentation.

Das Transitionssystem (reprAutomaton), mit dem der Analyseinterpreter umgeht, ist entsprechend Abbildung 5.2 entweder ein bddAutomaton (falls der Analyseinterpreter eine BDD-Repräsentation gewählt hat) oder ein ddmAutomaton (falls der Analyseinterpreter eine DDM-Repräsentation gewählt hat). Da das Transitionssystem durch den Produktautomaten definiert wird, wird hier der Name ”Automaton” für die Transitionssysteme verwendet.

Analog zu den Repräsentationsautomaten gibt es eine solche Struktur für Konfigurationsmengen, d. h. eine Schnittstellendefinition reprConfig wird durch die Unterklassen bddConfig und ddmConfig implementiert. Durch die beiden Oberklassen für die Repräsentationsschnittstellen wurde der Analyseinterpreter von der verwendeten Datenstruktur entkoppelt, d. h. dem Analyseinterpreter ist nicht bekannt, welche Datenstruktur verwendet wird; die konkreten Operationen werden durch polymorphe Aufrufe ausgewählt.

Für alle wichtigen syntaktischen Elemente eines Automaten gibt es eine entsprechende Klasse nach dem Muster von ctaAutomaton (siehe Abbildung 5.3). Diese Klassen stellen einerseits die Konstrukte für den abstrakten Syntaxbaum zur Verfügung (entsprechend dem Composite-Muster) und dienen andererseits als Konstrukteure für die Repräsentation (entsprechend dem Builder-Muster). Jede Syntaxbaum-Klasse enthält für jede Repräsentation eine Methode, um die dem syntaktischen Konstrukt entsprechende semantische Repräsentation zu erzeugen. Die rekursive Hierarchie des Syntaxbaumes und die dynamische Auswahl der adäquaten Transformations-Methode werden durch Instanzierung des Composite-Musters für die Klassen ctaRestriction und ctaExpression mit ihren Unterklassen realisiert.

Integration einer weiteren Repräsentation. Das Analyseframework kann um eine neue Datenstruktur zur Repräsentation erweitert werden, indem entsprechende Unterklassen von reprAutomaton und reprConfig abgeleitet werden. Die Syntaxbaum-Klassen werden um die zur Konstruktion der semantischen Repräsentation notwendigen Erzeugungsmethoden ergänzt, und es werden entsprechende Änderungen des Modelltransformators zur Auswahl der neuen Repräsentation vorgenommen.

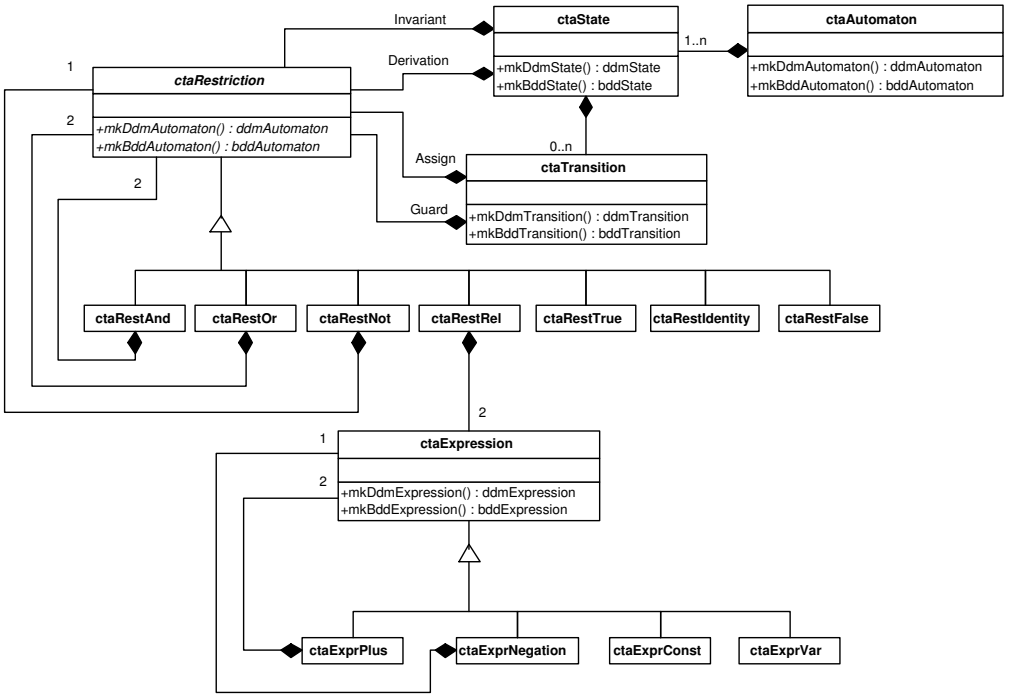


Abbildung 5.3: Klassendiagramm für den CTA-Syntaxbaum.

5.1.3 DDM-Bibliothek

Dieser Abschnitt erklärt die Implementierung der Bibliothek für die DDM-basierte Modell-Repräsentation und Verifikation (siehe auch [BR00b, BR00a]). Bei der in diesem Abschnitt beschriebenen Repräsentationsform handelt es sich um eine Datenstruktur zur symbolischen Darstellung des kontinuierlichen Zustandsraumes. Dazu wird die in Abschnitt 4.3.2 beschriebene Matrix-basierte Datenstruktur angewendet. Die diskreten Zustände der Automaten werden explizit repräsentiert. Daraus ergibt sich die wesentliche Schwachstelle aller Matrix-basierten Datenstrukturen: mit der Anzahl der Zustände des Produktautomaten wächst auch die Anzahl der im Speicher vorhandenen Matrizen exponentiell in Abhängigkeit von der Anzahl der Automaten im Modell (Zustandsraumexplosion).

Repräsentation der Konfigurationsmengen. Die zentrale Datenstruktur in der DDM-Repräsentation ist das doppelt repräsentierte Polyeder. Eine aus einer Menge von Vektoren bestehende Matrix (ddmMatrix) dient sowohl den ddmConstraints als auch den ddmRays als Grundlage (siehe Abbildung 5.4). Diese beiden Matrizen repräsentieren das Polyeder einmal durch ein Ungleichungssystem und einmal durch ein Erzeugendensystem (vgl. Abschnitt 4.3.2). Zum Vervollständigen der Datenstruktur zur Darstellung beliebiger Ungleichungssysteme werden zwei zusätzliche Dimensionen eingeführt, die zum ”eXtended Polyhedron” (ddmXPoly) führen: eine Dimension für die Information, ob es sich um

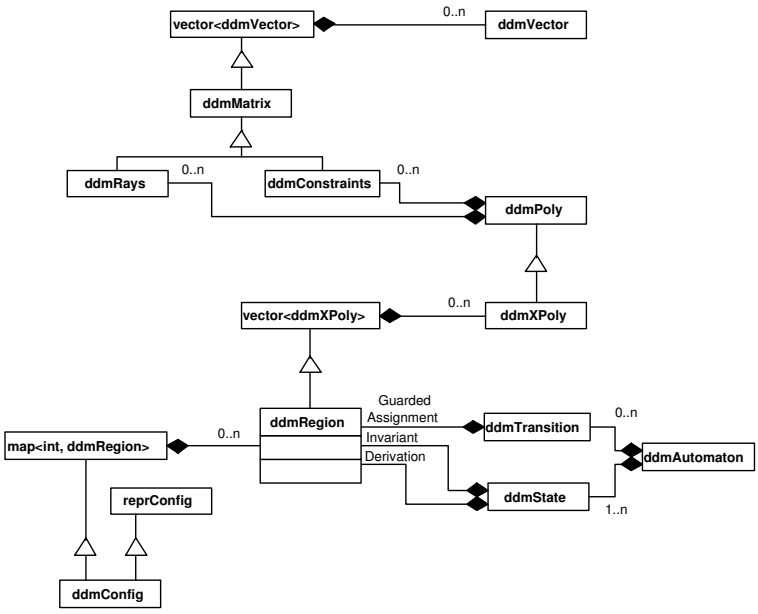


Abbildung 5.4: Klassendiagramm für die DDM-Repräsentation.

eine strikte oder nicht-strikte Ungleichung handelt, und eine weitere Dimension für die Konstanten auf der rechten Seite der Ungleichungen.

Da durch ein Polyeder nur konvexe Mengen dargestellt werden können, wird durch die Sammlung von Polyedern als ddmRegion eine Datenstruktur für die Regionen geschaffen. Zur Repräsentation einer Konfigurationsmenge dient schließlich ddmConfig, wobei jeweils Paare (Zustand, Region) abgespeichert werden. Es wird die STL-Datenstruktur "Abbildung" (map) benutzt, um jedem Zustand eine ddmRegion zuzuordnen. Das Verwenden einer map ist notwendig, um keinen Speicherplatz für Zustände mit leerer Region zu verschwenden (bei der Benutzung eines Arrays) und um einen effizienten Zugriff auf die einem Zustand zugeordnete Region zu gewährleisten.

Repräsentation des Transitionssystems. Bei der DDM-Repräsentation wird das Transitionssystem durch den Produktautomaten repräsentiert, d. h. sowohl die Zustände als auch die Zustandsübergänge werden explizit repräsentiert. Dadurch besteht ein ddmAutomaton aus einer Menge von Zuständen (ddmState) und einer Menge von Zustandsübergängen (ddmTransition), die wiederum Regionen (ddmRegion) für Invarianten und Ableitungen sowie für Wächter und Wertveränderungen (letztere kombiniert zu einer Region).

5.1.4 BDD-Bibliothek

Die Repräsentation der Konfigurationsmengen und der Transitionsrelation mittels Binary Decision Diagrams ist vielversprechend, weil sie die in der Forderung 2 auf Seite 14 genannten drei Kriterien erfüllt. Durch diese Eigenschaften der BDD-Repräsentation kann

der Zustandsraum oft stark komprimiert werden. Die auf einer BDD-Datenstruktur basierende Version des Werkzeugs Rabbit wurde in [Bey01c] vorgestellt.

Repräsentation der Konfigurationsmengen und des Transitionssystems. Sowohl die Konfigurationsmengen als auch die Transitionsrelation des Transitionssystems werden mittels BDDs direkt dargestellt. Bei der Transitionsrelation ist zu beachten, daß diese aus Effizienzgründen nicht monolithisch, sondern als implizite Vereinigung repräsentiert wird. Die einzelnen Transitionen werden in je einem BDD gespeichert und bei der Berechnung der Nachfolger nacheinander angewendet. Details wurden bereits im Abschnitt 3.2.6 ausgeführt.

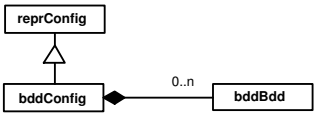


Abbildung 5.5: Klassendiagramm für die BDD-Repräsentation.

Wie in Abbildung 5.5 für die Konfigurationsrepräsentation angedeutet wurde die BDD-Bibliothek analog zur DDM-Bibliothek in das Analyseframework eingebettet. Von der Klasse reprConfig wurde eine Klasse bddConfig abgeleitet, die im wesentlichen einen BDD der Klasse bddBDD aggregiert. Zusätzlich werden Informationen für die Variablenverwaltung geführt (Symboltabelle). Die Klasse bddConfig implementiert alle für das Rechnen mit Konfigurationsmengen erforderlichen und von der BDD-Repräsentation unterstützten Operationen, auf die im folgenden separat eingegangen wird. Für das Transitionssystem wird von der Klasse reprAutomaton eine Klasse bddAutomaton abgeleitet, wobei die Algorithmen für die Erreichbarkeitsanalyse entsprechend der Datenstruktur implementiert werden. In Abschnitt 3.2.4 wurden die Operationen des BDD-Pakets erläutert.

Entsprechend der gängigen Definition von Alur wurde im Abschnitt 2.2.2 auf von mehreren Automaten gemeinsam benutzte Uhren verzichtet und für die Zustandsübergänge nur das Rücksetzen erlaubt. In der Werkzeugimplementierung wurden diese beiden Einschränkungen aufgehoben. Die Zustandsäquivalenz der ganzzahligen Semantik bei der Erreichbarkeitsanalyse bleibt davon unberührt. Ein Beweis dieser Behauptung würde für jede gemeinsam benutzte Uhr einen zusätzlichen Automaten einführen, dem diese Uhr zugeordnet ist. Die Kommunikation mit den Automaten, die ursprünglich auf die gemeinsame Uhr zugegriffen haben, würde über Synchronisationsmarken abgewickelt werden.

5.2 Modellierungssprache

Um eine Übersicht zu den Modellierungsmöglichkeiten zu geben, wird die Notation für die Systembeschreibung kurz vorgestellt. Die syntaktischen Konstrukte können grob in die folgenden Kategorien unterteilt werden:

Elemente zur Strukturbeschreibung: Ein wesentlicher Vorzug des CTA-Formalismus ist die Möglichkeit zur modularen Strukturierung des Modells. Einzelne Kompo-

nenten des System-Modells werden in Module eingekapselt und können über einen Schablonen-/Instanziierungsmechanismus in einer kompositionellen Hierarchie angeordnet werden. Diese Mechanismen, die zur Unterstützung des Modellierungsprozesses eingeführt wurden, sind gleichzeitig für die Verifikation von großem Nutzen.

Elemente zur Verhaltensbeschreibung: Das Verhalten des Systems wird innerhalb eines Moduls durch (hybride) Automaten definiert. Ein Automat besteht dabei aus Kontrollzuständen, Zustandsübergängen und einer Initialisierung der Variablen.

Syntaktischer Zucker: Neben den für die Verifikation notwendigen syntaktischen Elementen existieren Konstrukte, die dem Modellierer bei einer möglichst fehlerfreien Modellerstellung helfen sollen. Durch fest definierte Gültigkeitsbereiche der Bezeichner werden das versehentliche Mehrfachbenutzen eines Bezeichners oder unzulässige Zugriffe vermieden. Durch die Deklaration eines Bezeichners entsprechend den vorgegebenen Daten- und Zugriffstypen wird ebenfalls eine nicht vorgesehene Verwendung unterbunden.

5.2.1 Module

Ein Modul (Schlüsselwort MODULE) enthält vier syntaktische Komponenten: die Deklaration der Variablen und Signale, die Definition der Initialkonfigurationen, eine Menge von Modul-Instanziierungen und die Definition des Automaten. Das in Abschnitt 4.1.7 vorgestellte Beispiel-System der Bahnschranke wird hier aufgegriffen und in mehreren Abbildungen in der textuellen Repräsentation dargestellt, um einige syntaktische Konstrukte darzustellen.

```
1   MODULE RailRoadCrossing {
2     LOCAL
3       app  : SYNC;
4       exit : SYNC;
5       lower: SYNC;
6       raise: SYNC;
7     INST Process_Train FROM Train WITH {
8       app AS app;
9       exit AS exit;
10    }
11    INST Process_Gate FROM Gate WITH {
12      lower AS lower;
13      raise AS raise;
14    }
15    INST Process_Controller FROM Controller WITH {
16      app AS app;
17      exit AS exit;
18      lower AS lower;
19      raise AS raise;
20    }
21  }
```

Abbildung 5.6: CTA-Modul für das Gesamt-Modell des Bahnübergangs.

Eines der in der Systembeschreibung definierten Module ist ein sogenanntes 'Top-Modul', welches das gesamte System repräsentiert und Instanziierungen anderer Module beinhaltet. Im Top-Modul des Beispiels (RailRoadCrossing in Abbildung 5.6) wurden alle Variablen als LOCAL deklariert, da es sich um ein geschlossenes System handelt.

```
1   MODULE Train {
2     OUTPUT
3       app: SYNC;
4       exit: SYNC;
5     LOCAL
6       x: ANALOG;    // Distance of train.
7     INITIAL
8       STATE(Train) = Far AND x >= 1000;
9     AUTOMATON Train {
10      STATE Far  { INV     x >= 1000;
11                   DERIV   DER(x) >= -50 AND DER(x) <= -40;
12                   TRANS { GUARD   x = 1000;
13                           SYNC   !app;
14                           GOTO   Near; }
15                 }
16      STATE Near { INV     x >= 0;
17                   DERIV   DER(x) >= -50 AND DER(x) <= -30;
18                   TRANS { GUARD  x = 0;
19                           GOTO   Past; }
20                 }
21
22      STATE Past { INV     x <= 100;
23                   DERIV   DER(x) >= 30 AND DER(x) <= 50;
24                   TRANS { GUARD  x = 100;
25                           SYNC   !exit;
26                           DO     x' = 1000;
27                           GOTO   Far; }
28                 }
29    }
30  }
```

Abbildung 5.7: CTA-Modul für das Zug-Modell.

(Offene Systeme haben mindestens eine Variable oder ein Signal vom Zugriffstyp INPUT, OUTPUT oder MULTREST.) Dieses Modul für das Gesamtsystem besteht aus dem Zug-Modell (Abbildung 5.7), der Schranke (Abbildung 5.8 auf der nächsten Seite) und der Steuerung (Abbildung 5.9 auf Seite 155).

5.2.2 Schnittstellen

Die Modellierungsnotation des CTA-Formalismus beinhaltet zwei orthogonale Klassifikationstypen für Bezeichner der Variablen und Signale. Zunächst werden die vier verschiedenen Zugriffsmodi erklärt:

- **LOCAL:** Lokale Variablen oder Signale sind außerhalb des Moduls, in dem sie deklariert wurden, nicht sichtbar. Sie werden innerhalb eines Moduls zur Kommunikation zwischen dem Automaten und den enthaltenen Untermodulen genutzt.

- **INPUT:** Eingabe-Variablen oder -Signale dürfen innerhalb des Moduls nicht eingeschränkt werden. Damit ist für Signale gemeint, daß das Modul (bzw. ein im Modul enthaltener Automat) in jeder Konfiguration in der Lage sein muß, auf ein solches Signal zu reagieren.

- **OUTPUT:** Ausgabe-Variablen oder -Signale dürfen nur in dem Modul, in dem sie deklariert wurden, eingeschränkt werden, d. h. in der Umgebung des Moduls darf nur lesend zugegriffen werden. Damit ist für Signale gemeint, daß die Entscheidung, wann die Marke auftritt, einzig und allein innerhalb dieses Moduls getroffen wird.

```
1   MODULE Gate {
2     LOCAL
3       g: ANALOG;    // Degree of gate.
4     INPUT
5       lower: SYNC;  // Lower gate.
6       raise: SYNC;  // Raise gate.
7     INITIAL
8       STATE(Gate) = Open AND g = 90;
9     AUTOMATON Gate {
10      STATE Open {   INV   g = 90;
11                     DERIV  DER(g) = 0;
12                     TRANS { SYNC ?raise;
13                             GOTO Open; }
14                     TRANS { SYNC ?lower;
15                             GOTO Down; }
16                }
17      STATE Up   {   INV   g <= 90;
18                     DERIV  DER(g) = 9;
19                     TRANS { SYNC ?raise;
20                             GOTO Up; }
21                     TRANS { SYNC ?lower;
22                             GOTO Down; }
23                     TRANS { GUARD   g = 90;
24                             GOTO Open; }
25                }
26      STATE Down {   INV   g >= 0;
27                     DERIV  DER(g) = -9;
28                     TRANS { GUARD   g= 0;  GOTO Closed; }
29                     TRANS { SYNC ?raise;
30                             GOTO Up; }
31                     TRANS { SYNC ?lower;
32                             GOTO Down; }
33                  }
34      STATE Closed { INV   g = 0;
35                     DERIV  DER(g) = 0;
36                     TRANS { SYNC ?raise;
37                             GOTO Up; }
38                     TRANS { SYNC ?lower;
39                             GOTO Closed; }
40                  }
41    }
42  }
```

Abbildung 5.8: CTA-Modul für das Schrankenmodell.

- **MULTREST:** Mehrfach einschränkbare Variablen und Signale dürfen sowohl innerhalb des Moduls als auch außerhalb des Moduls eingeschränkt werden (ähnlich den normalen globalen Variablen).

Die Steuerung in Abbildung 5.9 auf der nächsten Seite hat zum Beispiel zwei Eingabe-Signale, um mit dem Zug-Modell zu kommunizieren. Diese müssen als INPUT deklariert werden, weil die Steuerung gezwungen sein soll, auf die Signale vom Zug zu reagieren. Damit muß ein in der Steuerung enthaltener Automat jederzeit in der Lage sein, mit diesen Signalen zu synchronisieren. Für die Kommunikation mit der Schranke existieren zwei Ausgabe-Signale, die das Verhalten der Schranke steuern. Diese müssen als OUTPUT deklariert sein, um das Reagieren der Umgebung (in diesem Fall nur die Schranke) auf diese Signale sicherzustellen. Die Uhr t wird zum Messen der Reaktionszeit der Steuerung benötigt, da diese innerhalb einer bestimmten Toleranzgrenze liegen muß.

Sind in einem Automaten nicht alle notwendigen Zustandsübergänge für die Reaktion auf ein Eingabe-Signal modelliert worden, fügt das Werkzeug die fehlenden Übergänge hinzu, jedoch führen diese in einen speziellen Fehlerzustand INPUT_ERROR. Wurde ein solcher Zustand generiert, kann per Erreichbarkeitsanalyse überprüft werden, ob dieser Zustand erreichbar ist. Die Erreichbarkeit dieses Zustands deutet auf einen Modellierungsfehler hin – es wurde eine Situation modelliert, in der ein Eingabe-Signal eingeschränkt wird, was zu einem Deadlock führen kann und deshalb vermieden werden muß.

```
 1  MODULE Controller {
 2    LOCAL
 3      t: CLOCK;    // Timer.
 4    INPUT
 5      app: SYNC;
 6      exit: SYNC;
 7    OUTPUT
 8      lower: SYNC;  // Lower gate.
 9      raise: SYNC;  // Raise gate.
10    INITIAL
11      STATE(Controller) = Idle;
12    AUTOMATON Controller {
13      STATE Idle    { TRANS { SYNC ?app;
14                              DO   t' = 0;
15                              GOTO ToLower;}
16                      TRANS { SYNC ?exit;
17                              DO   t' = 0;
18                              GOTO ToRaise;}
19                    }
20      STATE ToLower { INV     t <= 5;
21                      TRANS { SYNC ?app;
22                              GOTO ToLower; }
23                      TRANS { SYNC !lower;
24                              GOTO Idle; }
25                      TRANS { SYNC ?exit;
26                              DO   t' = 0;
27                              GOTO ToRaise; }
28                    }
29      STATE ToRaise { INV     t <= 5;
30                      TRANS { SYNC ?exit;
31                              GOTO ToRaise; }
32                      TRANS { SYNC !raise;
33                              GOTO Idle; }
34                      TRANS { SYNC ?app;
35                              DO   t' = 0;
36                              GOTO ToLower; }
37                    }
38    }
39  }
```

Abbildung 5.9: CTA-Modul für das Steuerungsmodell.

Weiterhin werden fünf verschiedene (Daten-) Typen der Bezeichner unterschieden (vgl. HyTech [HHWT95b]):

CONST: Ein Bezeichner, dem ein fester Wert zugeordnet ist. Dies hat die gleiche Wirkung wie eine Variable, deren Ableitung immer 0 ist und die für alle Module Eingabe-Variable ist.

DISCRETE: Eine diskrete Variable dient dem Speichern eines Wertes. Die Ableitung einer diskreten Variable hat immer den Wert 0.

CLOCK: Eine Uhr ist eine kontinuierliche Variable mit der festen Ableitung 1.

STOPWATCH: Eine Stoppuhr kann im Gegensatz zur Uhr auch angehalten werden; deshalb hat die Ableitung immer den Wert 0 oder den Wert 1.

ANALOG: Bei einer analogen Variable unterliegt die Ableitungsfestlegung keinerlei Einschränkung. Die Ableitung kann durch ein beliebiges lineares Prädikat definiert werden.

SYNC: Ein Signal (bzw. Synchronisationsmarke) dient der Synchronisation der Automaten durch Parallelschaltung von Zustandsübergängen.

Im betrachteten Beispiel besitzt das Steuerungsmodell des Bahnübergangs eine Uhr als kontinuierliche Variable und vier Signale zur Synchronisation.

5.2.3 Instanziierungen

Um ein Modell für das Gesamtsystem zu definieren, enthält das Modul RailRoadCrossing drei Instanziierungen anderer Module (Schlüsselwort INST). Jede Schnittstellenvariable und jedes Schnittstellensignal eines Schablonenmoduls wird mit einer der Variablen oder einem der Signale des enthaltenden Moduls identifiziert (AS). Lokale Variablen oder Signale können nicht mit Variablen oder Signalen des enthaltenden Moduls identifiziert werden.

5.2.4 Automaten

Kontrollzustände und Zustandsübergänge. Ein Automat (AUTOMATON) besteht aus einer Menge von Zuständen. Für jeden Zustand (STATE) existiert eine Variablenbedingung für die Menge der möglichen Ableitungen nach der Zeit aller Variablen (DERIV) und eine weitere Variablenbedingung für die Definition der Invariante (INV), die erfüllt sein muß, solange der Automat in dem Zustand verbleibt.

Die Zustandsübergänge (TRANS) sind syntaktisch ihrem Anfangszustand zugeordnet. Ein Zustandsübergang enthält eine Variablenbedingung für den Wächter (GUARD), ein Signal (SYNC), eine Variablenbedingung (DO) für die Wertveränderung (um die Werte der Variablen nach dem Übergang festzulegen) und den Folgezustand des Übergangs (GOTO). Ein Übergang kann nur ausgewählt werden, wenn die aktuelle Konfiguration die Variablenbedingung des Wächters erfüllt und alle anderen Automaten, die das Signal des Übergangs kennen, ebenfalls einen Übergang mit demselben Signal vollziehen. Die Signale werden in Transitionen durch einen Präfix '?', '!' oder '#' für die Benutzung als Eingabe-, Ausgabe- oder mehrfach einschränkbares Signal deklariert.

Variablenbedingungen. Variablenbedingungen sind das syntaktische Element zum Festlegen der Belegungsmengen bei der Definition von Invarianten, Wächtern, etc. Für alle in Definition 4.1 erlaubten Bedingungen existieren syntaktische Entsprechungen in der Modellierungssprache.

Eine Variable x kann in den folgenden drei Formen in Variablenbedingungen auftreten:

- **x** bezeichnet den Wert der Variable x. In der Variablenbedingung für eine Wertveränderung meint x den Wert der Variable vor dem Zustandsübergang.
- **x'** kommt nur in Bedingungen für Wertveränderungen vor. x' bezeichnet den Wert der Variable nach der Ausführung des Übergangs.
- **DER(x)** wird nur bei der Festlegung der Ableitung in Zuständen benutzt und adressiert die erste Ableitung nach der Zeit für eine Variable.

Initialisierung. Der Anfangszustand eines Automaten und die Anfangswerte der Variablen werden durch eine Initialisierungsklausel (INITIAL) des Moduls festgelegt. Dabei handelt es sich um eine Kombination von Variablenbedingungen und Zustandsbedingungen (STATE(automaton_x)=state_y) für Automaten zur Festlegung von Konfigurationsmengen.

5.2.5 Gültigkeitsbereiche

Jedes Modul hat seinen eigenen Namensbereich. Alle Namen von Variablen und Signalen sind auf den Bereich des Moduls beschränkt und außerhalb der Modulgrenze nicht bekannt. Um Kommunikation zwischen verschiedenen Modulen zu erlauben, können die Schnittstellenbezeichner enthaltener Module mit Bezeichern des diese Module enthaltenden Moduls identifiziert werden, entsprechend der Konsistenz- und Kompatibilitätseinschränkungen. Als LOCAL deklarierte Bezeichner können nicht bei der Identifikation von Schnittstellenbezeichnern verwendet werden.

Ein Modul besitzt mehrere verschiedene Namensbereiche für Automaten, Kontrollzustände, Variablen und Signale sowie für Modulinstanzen. Für die Modulnamen existiert ein globaler Namensbereich.

5.3 Verifikationssprache

Die vom Analyseinterpreter unterstützten Operationen werden in diesem Abschnitt erklärt. Das Werkzeug stellt die Erreichbarkeitsanalyse in der Weise zur Verfügung gestellt, daß die gewünschten Analysen mittels Analyseanweisungen programmiert werden. Der Analyseinterpreter ist nicht beschränkt auf Fragen der Form ”Erfüllt Modell A die Formel B?”. Es können sehr allgemeine Verifikationsaufgaben formuliert und auch schrittweise Berechnungen durchgeführt werden.

5.3.1 Erreichbarkeitsanalyse

Die Sprache für die Erreichbarkeitsanalyse lehnt sich an die im Werkzeug HyTech implementierten Prinzipien an [HHWT95b], wurde jedoch um die Handhabung der durch die hierarchische Modulstruktur gegebenen verschiedenen Gültigkeitsbereiche erweitert. Eine Erweiterung auf die Prüfung eines Modells bzgl. einer temporallogischen Formel wäre denkbar, würde jedoch keine neuen Erkenntnisse für die in dieser Arbeit verfolgten Ziele bringen. Für die Verfeinerungsanalyse wurde die Analysesprache um eine entsprechende Anweisung weiterentwickelt.

Auch hier wird zur Illustration das Beispiel des Bahnübergangs verwendet. Der Analyseteil der Eingabedatei für die Verifikation des Bahnübergang-Modells wird in Abbildung 5.10 auf der nächsten Seite gezeigt.

Grundlegender Gedanke bei der Erreichbarkeitsanalyse ist das Rechnen mit Regionen (Konfigurationsmengen). Bestimmte Operationen werden aufgerufen und Regionen-Variablen können genutzt werden, um Zwischenergebnisse von Berechnungen für die spätere Verwendung im Verifikationsprozeß zu speichern. Da sich die Erreichbarkeitsanalyse auf verschiedene Module beziehen kann, wird in der ersten Zeile einer Analysespezifikation der Name des Moduls angegeben, welches das zu analysierende Gesamtsystem modelliert (Top-Modul des zu analysierenden Modells).

Deklaration von Regionen-Variablen. Wie in Abbildung 5.10 auf der nächsten Seite gezeigt, werden für die Verifikation des Bahnübergang-Modells drei Regionen-Variablen

```
REACHABILITY CHECK RailRoadCrossing
{
    VAR
      initial, error, reached : REGION;

    COMMANDS
      // Initial regions from modules.
      initial := INITIALREGION;

      error   :=   COMPLEMENT( STATE(Process_Gate.Gate) = Closed )
                INTERSECT
                  Process_Train.x < 250;

      reached := REACH FROM initial FORWARD;

      IF( EMPTY(error INTERSECT reached) )
      {
        PRINT " Safety requirement satisfied. ";
      }
      ELSE
      {
        PRINT " Safety requirement violated. ";
        PRINT " The resulting region: " reached;
      }

}
```

Abbildung 5.10: Zum Modell des Bahnübergangs gehörende Analysespezifikation.

benutzt: initial speichert die Menge der Initialkonfigurationen. reached ist für die Menge aller von initial aus erreichbaren Konfigurationen vorgesehen und die Regionen-Variable error speichert die Menge aller Konfigurationen, die in einem korrekten Modell nicht erreicht werden dürfen.

Regionen-Ausdrücke. Ein Regionen-Ausdruck dient zur Konstruktion von Regionen, jede von ihnen wird durch eine Regionen-Konstante, Variablenbedingung, Zustandsbedingung, Regionen-Variable oder durch einen Berechnungsoperator festgelegt.

Regionen-Konstanten: Eine Regionen-Konstante definiert eine fest durch ein Schlüsselwort vorgegebene Region. Es existieren die Regionen-Konstanten FALSE (für die leere Konfigurationsmenge) und TRUE (für die Menge aller Konfigurationen des Modells).

Variablen- und Zustandsbedingungen: Zur Definition von Regionen können Variablenbedingungen über sämtliche Variablen des Modells benutzt werden. Process_Train.x $<$ 250 zum Beispiel definiert die Region aller Konfigurationen, die die Bedingung erfüllen, daß die Variable x der Modulinstanz Process_Train kleiner als 250 ist. Andererseits können Regionen über Zustandsbedingungen definiert werden, d. h. der Zustand eines Automaten wird festgelegt. Zum Beispiel definiert STATE(Process_Gate.Gate) = Process_Gate.Closed die Menge aller Konfigurationen, bei denen sich der Automat Gate in der Modulinstanz Process_Gate im Zustand Closed befindet.

Erreichbarkeitsoperationen: Für das Berechnen von Konfigurationen, die mit diskreten oder Zeit-Transitionen von einer gegebenen Region aus erreicht werden können, stehen vier verschiedene Operatoren zur Verfügung: POST(<region>) und PRE(<region>) für die Berechnung der direkten, in einem Schritt erreichbaren Nachfolger, und REACH FROM <region> FORWARD und REACH FROM

<region> BACKWARD für das Berechnen des Fixpunktes durch iterative Anwendung der Operatoren POST oder PRE. Durch die Angabe einer Maximalanzahl von Schritten kann die Fixpunktiteration nach der vorgegebenen Anzahl von Erreichbarkeitsschritten abgebrochen werden. So liefert z. B. der Ausdruck REACH FROM <region> FORWARD IN <n> STEPS die Menge der in n Vorwärtsschritten erreichbaren Konfigurationen.

Mengenoperationen: Da es sich bei Regionen um Mengen handelt, stehen auch die Mengenoperationen für die Berechnung zur Verfügung. In der Analysesprache existieren drei binäre und ein unärer Operator: INTERSECT für den Durchschnitt zweier Regionen, UNION für die Vereinigung zweier Regionen, DIFFERENCE für die Mengendifferenz zweier Regionen und COMPLEMENT für das Mengen-Komplement einer Menge. Für INTERSECT, UNION und COMPLEMENT können analog AND, OR und NOT verwendet werden, da sich die Operatoren semantisch entsprechen.

Regionen-Variable: Eine Region kann auch durch die Angabe einer Variable adressiert werden. Dann ist die Ergebnis-Region eine Kopie der bereits berechneten und der Variable zugeordneten Konfigurationsmenge.

Initial-Region: Mit dem Schlüsselwort INITIALREGION kann auf die Menge der Initialkonfigurationen des Modells zugegriffen werden, so wie sie in den Initialisierungsklauseln innerhalb der Module definiert sind.

Konventionen für den Komponentenzugriff. Aufgrund der verschiedenen Gültigkeitsbereiche wird die Punkt-Notation verwendet, um in einem Analyseabschnitt eine bestimmte Komponente innerhalb einer Variablen- oder Zustandsbedingung zu adressieren. Im Analyseabschnitt der Eingabedatei können die Bezeichner aller Variablen und Kontrollzustände benutzt werden, die Kapselung durch die Gültigkeitsbereiche dient nur der strukturierten Modellierung. Als aktueller (Referenz-) Namensbereich dient der des Top-Moduls des zu analysierenden Modells. Um die Variable x des Moduls Process_Train zu adressieren, wird deren absoluter Name Process_Train.x benutzt. Der Zustand Closed des Automaten Gate im Modul Process_Gate wird mit Process_Gate.Closed bezeichnet.

Boolesche Ausdrücke. In einer bedingten Anweisung werden Boolesche Ausdrücke als Bedingung für die Ausführung eines bestimmten Anweisungsblocks verwendet. Ein Boolescher Ausdruck ist ein Prädikat über Regionen und besteht aus Vergleichen und Tests von Konfigurationsmengen sowie aus Booleschen Kombinationen solcher Ausdrücke.

Boolesche Konstanten: Es werden die beiden Booleschen Konstanten FALSE und TRUE in ihren gewöhnlichen Bedeutungen angeboten.

Vergleiche und Tests: Zur Auswertung einer Mengenrelation zwischen zwei Regionen stehen die Operatoren '=' und CONTAINS für den Test auf Gleichheit und Teilmengenbeziehung zur Verfügung. Mit dem Schlüsselwort EMPTY kann geprüft werden, ob eine Region leer ist. Da bei der On-the-fly-Analyse nicht die Menge aller erreichbaren Konfigurationen berechnet wird, steht für diese Analyse ein besonderer Boolescher Ausdruck zur Verfügung. Der Ausdruck ISREACHABLE FROM initial

TO error FORWARD liefert genau dann den Wert $true$ zurück, wenn ausgehend von der in der Regionenvariablen initial gespeicherten Menge der Initialkonfigurationen eine Konfiguration aus der in der Regionenvariablen error gespeicherten Menge von Konfigurationen erreichbar ist.

Boolesche Kombinationen: Boolesche Ausdrücke können durch das Anwenden gewöhnlicher Boolescher Operatoren kombiniert werden: AND, OR und NOT in den bekannten Bedeutungen.

Anweisungen. Die Verifikationsaufgaben werden durch die sequentielle Aneinanderreihung der folgenden Anweisungen formuliert.

Zuweisungen: Mit einer Zuweisung kann einer (vorher deklarierten) Regionen-Variable das Ergebnis eines Regionen-Ausdrucks für spätere Benutzung zugewiesen werden, z. B. wertet der Analyseinterpreter bei der Anweisung reached := REACH FROM initial FORWARD zunächst den Regionen-Ausdruck auf der rechten Seite der Zuweisung aus und ordnet das Ergebnis der Variablen reached zu.

Bedingte Anweisung: Mit den Schlüsselwörtern IF, THEN und ELSE kann eine bedingte Anweisung konstruiert werden. Dabei wird der auf IF folgende in runde Klammern gesetzte Boolesche Ausdruck ausgewertet und dem Resultat entsprechend der auf THEN oder der auf ELSE folgende Anweisungsblock in geschweiften Klammern ausgeführt.

Schleifen: Um einen iterativen Verifikationsalgorithmus zu beschreiben, wird durch das Schlüsselwort WHILE eine Schleife bereitgestellt. Bei jeder Iteration wird der nach dem Schlüsselwort in runde Klammern gesetzte Boolesche Ausdruck ausgewertet und dem Resultat entsprechend eine erneute Iteration durchgeführt (Ausführen des Anweisungsblocks in geschweiften Klammern und erneute Prüfung der Bedingung) oder zur dem Anweisungsblock folgenden Anweisung übergegangen.

Ausgaben: Für die Rückmeldungen der Ergebnisse an den Benutzer ist die PRINT-Anweisung vorgesehen. Das Argument der Anweisung kann entweder eine Zeichenkette, ein Regionen-Ausdruck oder ein Regionen-Ausdruck zwischen zwei Zeichenketten sein. Eine Region wird in einem für die jeweilig benutzte Repräsentation angemessenen und übersichtlichen Format ausgegeben. Die Ausgabe-Anweisungen für Automaten, für Größenangaben zur Repräsentation einer Region und zur graphischen Visualisierung von BDDs werden hier nicht erläutert.

Die Anweisungen im in Abbildung 5.10 auf Seite 158 gezeigten Beispiel definierten drei Regionen: die Menge der Startkonfigurationen, die Menge der verbotenen Konfigurationen (die vom Modell vermieden werden müssen) und die Menge der von den Startkonfigurationen erreichbaren Konfigurationen. In der bedingten Anweisung wird ausgewertet, ob der Schnitt der Erreichbarkeitsmenge mit der Fehlermenge leer ist und dementsprechend das Analyseresultat ausgegeben. Die Sicherheitsbedingung kann durch das vorgeschlagene Modell nicht erfüllt werden, wenn die Reaktionszeit der Steuerung oder der Sicherheitsabstand erhöht werden.

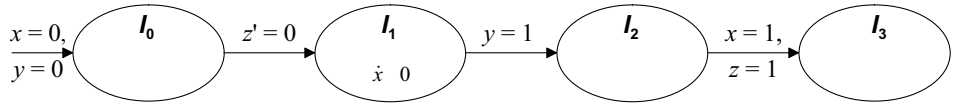


Abbildung 5.11: Nicht diskret interpretierbarer Automat mit Stoppuhr.

Einschränkungen für abgeschlossene Timed Automata. Folgende Möglichkeiten, die von der DDM-Repräsentation angeboten werden und im Werkzeug als Analyseanweisungen verankert sind, sind bei der BDD-Repräsentation nicht verfügbar, da sie bei der Berechnung zu falschen Ergebnissen führen können:

Test auf Teilmengenbeziehung und Gleichheit: Die BDD-Darstellung beruht auf einer Diskretisierung des kontinuierlichen Zustandsraumes, wobei einzelne diskrete Repräsentanten eine Menge von unendlich vielen kontinuierlichen Werten repräsentieren können. Werden bei den Analyseanweisungen Operationen wie der Test auf Teilmengenbeziehung angewendet, muß sich der Benutzer darüber im klaren sein, daß die mit Operationen in der diskreten Repräsentation errechneten Ergebnisse nicht kontinuierlich interpretiert werden dürfen. Zum Beispiel würde bei der BDD-Repräsentation durch die diskrete Darstellung $(x = 2 \vee x = 3) \supseteq (x \geq 2 \wedge x \leq 3)$ gelten. Analoges gilt für die Gleichheit.

Differenz- und Komplementbildung: Aus demselben Grund ist auch die Differenz- und Komplementbildung nicht erlaubt. Sowohl die Menge COMPLEMENT$(x \leq 2 \wedge x \geq 3)$ als auch die Menge $(x \geq 2 \wedge x \leq 3)$ DIFFERENCE $(x = 2 \vee x = 3)$ wären bei der diskreten Interpretation leer.

Stoppuhr- und analoge Variablen: In dem in Abbildung 5.11 dargestellten Automaten mit der Stoppuhr x und den Uhren y und z ist der Zustand l_3 bei kontinuierlicher Semantik erreichbar, bei ganzzahliger Semantik jedoch nicht, da die diskreten Zustandsübergänge zu den teilweise nicht ganzzahligen Zeitpunkten $0.5, 1.0, 1.5$ schalten müssen.

Verwendung der strikten Relationsoperatoren Aufgrund der abgeschlossenen ganzzahligen Semantik dürfen die Operatoren $<$, $>$ und $\neq$ bei Ungleichungen nicht verwendet werden. Dies ist damit begründet, daß die diskreten Repräsentanten einer Menge immer in der Menge eingeschlossen sein müssen.

5.3.2 Verfeinerungsanalyse

Zur Illustration der praktischen Wirkungsweise der Verfeinerungsanalyse und der Notation des Werkzeugs wurde ein kleines Kommunikationsprotokoll entworfen, welches auf einer hierarchischen (Kommunikations-) Struktur basiert. Ein ähnliches Protokoll ist auf der Titelseite von [dRLP98] dargestellt (siehe Abbildung 5.12 auf der nächsten Seite); es verdeutlicht die Funktionsweise des hier beschriebenen Protokolls sehr gut. Das Beispiel

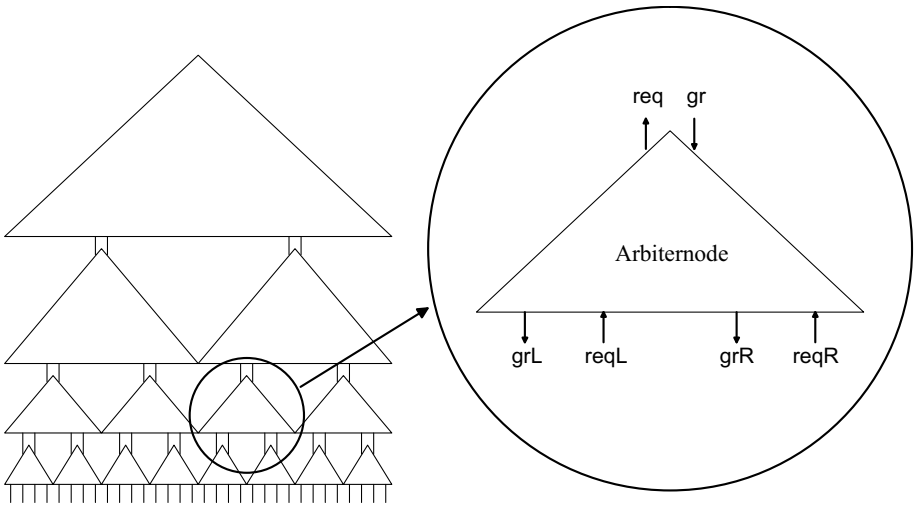


Abbildung 5.12: Stark kompositionelle Struktur (Graphik entnommen der Titelseite von ”Compositionality: The Significant Difference” [dRLP98]).

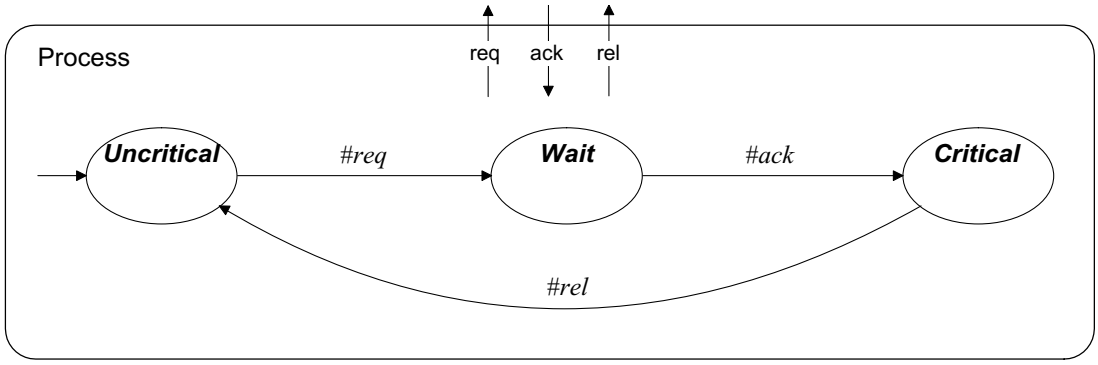


Abbildung 5.13: Prozeß-Modul.

hat den Vorteil, daß ein induktiver Beweis geführt werden kann, wobei der Induktionsschritt durch die Verfeinerungsanalyse sichergestellt wird. Die modulare Beweistechnik wird in Kapitel 6 auf ein größeres Modell einer Fertigungsanlage angewendet, um die praktische Relevanz der Methode zu verdeutlichen.

Jeder Prozeß (siehe Abbildung 5.13) hat drei Zustände: *Uncritical*, *Wait* und *Critical*. Beim Übergang von *Uncritical* nach *Wait* wird vom Prozeß das Signal req zum Scheduler gesendet. Ein übergeordneter Scheduler muß entscheiden, ob der Prozeß die exklusive Ressource benutzen darf oder nicht. Falls ja, sendet er das Signal ack zum Prozeß (identifiziert durch die lokale diskrete Variable no), wodurch dieser in seinem ’kritischen Abschnitt’ auf die Ressource zugreifen kann. Der Prozeß gibt die Ressource wieder frei, indem er das Signal rel sendet. Jeder Scheduler (siehe Abbildung 5.14) ist für zwei (von ihm eingekapselte) Prozesse zuständig, und er selbst verhält sich exakt so zu seiner Um-

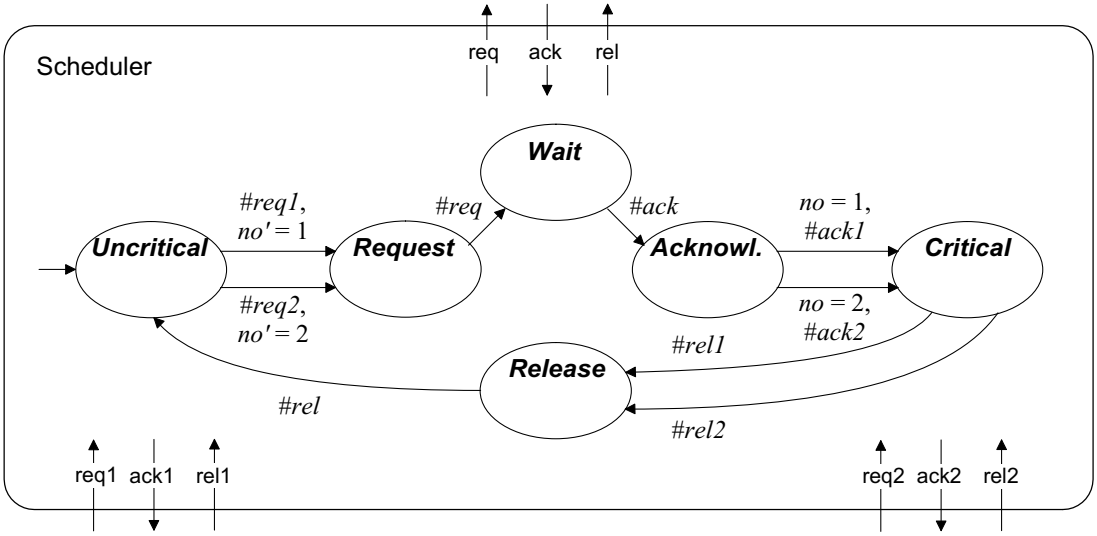


Abbildung 5.14: Scheduler-Modul für zwei Prozesse oder weitere Scheduler.

gebung (zu seinem übergeordneten Scheduler) wie ein Prozeß. Somit kann ein Baum bestehend aus Schedulern (Knoten) und Prozessen (Blätter) konstruiert werden.

Für die Verifikation der Eigenschaft des gegenseitigen Ausschlusses könnte die Erreichbarkeitsanalyse der gesamten 'geflatteten' Komposition genutzt werden; aufgrund der Komplexität ist dies bei sehr vielen Prozessen nicht möglich. Bei der Anwendung eines induktiven Beweises wird der gegenseitige Ausschluß zunächst für eine initiale Anzahl von Prozessen mittels Erreichbarkeitsanalyse nachgewiesen (Induktionsanfang). Für einen Prozeß ist dies trivial. Weiterhin muß nachgewiesen werden, daß, falls die Eigenschaft für n Prozesse gilt (Induktionshypothese), diese Eigenschaft auch für $2n$ Prozesse gilt (Induktionsschritt). Für den Induktionsschritt wird jetzt die Verfeinerungsanalyse verwendet: Die Existenz einer Simulationsrelation zwischen dem Scheduler und dem Prozeß wird überprüft, d. h. ob der Scheduler einen Prozeß verfeinert. Die Verifikation der Behauptung Scheduler refines Process benötigt weniger als 0.5 Sekunden Berechnungszeit.

Der Analyseabschnitt der Eingabedatei besteht aus einer Anweisung. Den Schlüsselwörtern REFINEMENT und CHECK folgt der Name des Moduls, welches eine genauere Spezifikation der Verfeinerungsanalyse enthält. In der Abbildung 5.15 auf der nächsten Seite sind das Prozeß-Modul, das Refinement-Modul und der Analyseabschnitt aufgeführt. Abbildung 5.16 auf Seite 165 zeigt den Aufbau eines Scheduler-Moduls mit zwei Modul-Instanzen für Prozesse. Im Modul Refinement befinden sich zwei Modulinstanzen P und Q, zwischen denen die Simulationsrelation in der Form P refines Q bestehen soll. P steht somit für die Instanz der Verfeinerung und Q für das abstraktere Modul. In diesem Modul werden alle gemeinsamen Synchronisationsmarken deklariert. Die beiden Modulinstanzen sind notwendig, damit gegebenenfalls Umbenennungen vorgenommen werden können, falls die Marken der Module P und Q unterschiedliche Namen aufweisen. Das Werkzeug gibt nach erfolgter Analyse eine Meldung darüber aus, ob eine Simulationsrelation besteht oder nicht.

```
 1 MODULE Process {
 2   MULTREST
 3     announce   : SYNC;
 4     acknowledge: SYNC;
 5     release    : SYNC;
 6   INITIAL   STATE(Mutex) = Uncritical;
 7   AUTOMATON Mutex {
 8     STATE Uncritical    { TRANS { SYNC #announce;
 9                                   GOTO Wait;         }  }
10     STATE Wait          { TRANS { SYNC #acknowledge;
11                                   GOTO Critical;     }  }
12     STATE Critical      { TRANS { SYNC #release;
13                                   GOTO Uncritical;   }  }
14   }
15 }
16
17 MODULE Refinement {
18   LOCAL
19     announce   : SYNC;
20     acknowledge: SYNC;
21     release    : SYNC;
22   INST Q FROM Process WITH        {
23     announce      AS announce;
24     acknowledge   AS acknowledge;
25     release       AS release;     }
26   INST P FROM Scheduler2 WITH     {
27     announce      AS announce;
28     acknowledge   AS acknowledge;
29     release       AS release;     }
30 }
31
32 REFINEMENT CHECK Refinement;
```

Abbildung 5.15: Prozeß-Modul und Refinement-Modul für die Verfeinerungsanalyse des einfachen Mutex-Protokolls.

5.4 Empirische Ergebnisse

In diesem Abschnitt werden einige in der Literatur gebräuchliche hybride Modelle vorgestellt, um einen Einblick in die Modellierungsmöglichkeiten mit hybriden Automaten zu geben. Danach werden verschiedene Modelle für Realzeit-Systeme und deren Verifikation erläutert. Dabei wird die Performance der Werkzeugimplementierung Rabbit der von vergleichbaren Werkzeugen anderer Forschungsgruppen gegenübergestellt.

5.4.1 Hybride Beispiele

Im folgenden werden einige hybride Systeme mit dem CTA-Formalismus modelliert und analysiert, wobei zur Modellierung des Verhaltens hybride Automaten zur Anwendung kommen.

5.4.1.1 Undichter Gasbrenner – Rückwärtsanalyse

In diesem Beispiel wird das Verhalten eines Gasbrenners modelliert, bei dem aus einem Leck Gas austritt. Es handelt es sich um ein hybrides System, da die Ausflußmenge je Zeiteinheit über die Laufzeit des Systems nicht konstant ist. Es bestehen die Annahmen, daß (1) jeder Defekt innerhalb 1 s entdeckt und beseitigt werden kann und (2) der Gasbrenner innerhalb der auf eine Reparatur folgenden 30 s dicht bleibt. Die mittels Verifikation nachzuweisende Sicherheitseigenschaft ist, daß der Gasbrenner nach einer beliebigen Zeit von wenigstens 60 s höchstens $\frac{1}{20}$ der Zeit undicht gewesen sein darf.

```
 1 MODULE Scheduler {
 2   MULTREST
 3     announce   : SYNC;
 4     acknowledge: SYNC;
 5     release    : SYNC;
 6   LOCAL
 7     announce1   : SYNC;
 8     acknowledge1: SYNC;
 9     release1    : SYNC;
10     announce2   : SYNC;
11     acknowledge2: SYNC;
12     release2    : SYNC;
13     pNr         : DISCRETE(3);
14   INITIAL   STATE(Mutex) = Uncritical;
15   AUTOMATON Mutex {
16     ...
17   }
18   INST Proc1 FROM Process WITH     {
19     announce      AS announce1;
20     acknowledge   AS acknowledge1;
21     release       AS release1;     }
22   INST Proc2 FROM Process WITH     {
23     announce      AS announce2;
24     acknowledge   AS acknowledge2;
25     release       AS release2;     }
26 }
```

Abbildung 5.16: Scheduler-Modul für das einfache Mutex-Protokoll.

Der hybride Automat für dieses Modell ist in Abbildung 5.17 dargestellt. Er hat zwei Kontrollzustände: Im Zustand *Leaking* leckt der Gasbrenner und im Zustand *NonLeaking* nicht. Die Stoppuhr z mißt die kumulierte Leckzeit, d. h. die Gesamtzeit, die der Automat im Zustand *Leaking* verbracht hat. Die Uhr x mißt die in einem Zustand verbrachte Zeit und wird bei jedem Zustandswechsel auf 0 zurückgesetzt, mit dieser Uhr werden die Annahmen (1) und (2) im Modell verankert. Die insgesamt verstrichene Systemzeit wird von der Uhr y gemessen.

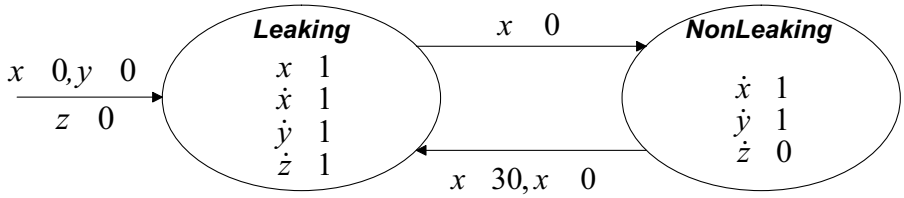


Abbildung 5.17: Hybrides Modell eines undichten Gasbrenners (nach [ACH+95]).

Ziel der Erreichbarkeitsanalyse ist der Nachweis, daß keine Konfiguration erreichbar ist, in der gilt $y \geq 60 \wedge 20 \cdot z \geq y$. In Abbildung 5.18 auf der nächsten Seite ist die Eingabedatei für das CTA-Modell und die Analyseanweisungen aufgeführt. Dieses Modell gehört zu der Klasse hybrider Automaten, für die die vorwärts gerichtete Fixpunktiteration der Erreichbarkeitsanalyse nicht terminiert. Durch zusätzliche Invarianten in den Zuständen können die Werte sämtlicher (Stopp-) Uhren nach unten begrenzt werden (d. h. für jede Variable x gilt: $x \geq 0$), wobei sich das modellierte Verhalten nicht verändert. Ausgehend von der Menge der Fehlerkonfigurationen können alle mittels Rückwärtsschritten erreichbaren Konfigurationen berechnet werden. Da die Werte aller Uhren nach unten begrenzt sind, keine negativen Ableitungen im Modell vorkommen und Zeno-Verhalten auch ausgeschlossen ist, muß die Rückwärtsanalyse terminieren. Die berechnete Erreichbarkeitsmenge wird mit der Menge der Initialkonfigurationen geschnitten. Ist diese Schnittmenge

```
MODULE GasBurner
{
  LOCAL
    x : CLOCK;     // Time elapsed in current state.
    y : CLOCK;     // Overall time.
    z : STOPWATCH; // Overall time leaking.
  INITIAL
    STATE(Auto) = Leaking AND x = 0 AND y = 0 AND z = 0;
  AUTOMATON Auto {
    STATE Leaking    { INV   x <= 1 AND x >= 0 AND y >= 0 AND z >= 0;
                       DERIV DER(z) = 1;
                       TRANS { DO x' = 0;
                               GOTO NonLeaking; }
                     }
    STATE NonLeaking { INV x >= 0 AND y >= 0 AND z >= 0;
                       DERIV DER(z) = 0;
                       TRANS { GUARD x >= 30;
                               DO x' = 0;
                               GOTO Leaking; }
                     }
  }
}
REACHABILITY CHECK GasBurner
{
  VAR
    initial, error, reached: REGION;
  COMMANDS
    initial := INITIALREGION;      // Initial configurations from module.
    error := (y >= 60) AND (20 * z >= y);
    reached := REACH FROM error BACKWARD;
    IF( EMPTY( reached INTERSECT initial ) )
    {
      PRINT " Requirements satisfied. ";
    }
    ELSE
    {
      PRINT " Invalid configuration reachable. ";
    }
}
```

Abbildung 5.18: CTA-Eingabedatei für die Verifikation des Gasbrenner-Modells.

leer, so ist keine der Fehlerkonfigurationen von einer Initialkonfiguration aus erreichbar, d. h. das Modell erfüllt die Sicherheitsanforderungen.

Die Verifikation dieses Beispielmodells benötigt Bruchteile einer Sekunde (160 ms). Dabei werden zwölfmal Nachfolger berechnet, d. h. die Fixpunktiteration terminiert nach 12 Iterationen.

5.4.1.2 Füllstandssteuerung – Invariantentest

Im Modell für eine Füllstandssteuerung soll für den in Abbildung 5.19 auf der nächsten Seite dargestellten Tank die Regelung des Wasserstands übernommen werden. Es handelt sich um ein System, bei dem einem Wassertank ständig eine Wassermenge von 2 Einheiten entnommen wird; der Wasserstand sinkt um 2 Einheiten pro Zeiteinheit. Entsprechend des Wasserstands muß die Steuerung dafür sorgen, daß ein Ventil rechtzeitig vor Erreichen eines minimalen Wasserstands geöffnet wird, damit genügend neues Wasser in den Tank strömen kann. Durch das Ventil strömen je Zeiteinheit 3 Einheiten Wasser in den Tank, d. h. der Wasserstand steigt um 1 Einheit je Zeiteinheit.

Der Wasserstand verändert sich mit der Zeit nach einer stückweise linearen Funktion, im Modell (Abbildung 5.20 auf Seite 168 zeigt den hybriden Automaten) durch die analoge Variable y wiedergegeben. Die Ableitung dieser Variable nach der Zeit beträgt 1, falls das Ventil geöffnet ist und der Füllstand steigt, und -2, falls das Ventil geschlossen ist und somit nur Wasser abfließt. Im Initialzustand des Systems ist das Ventil geöffnet und der

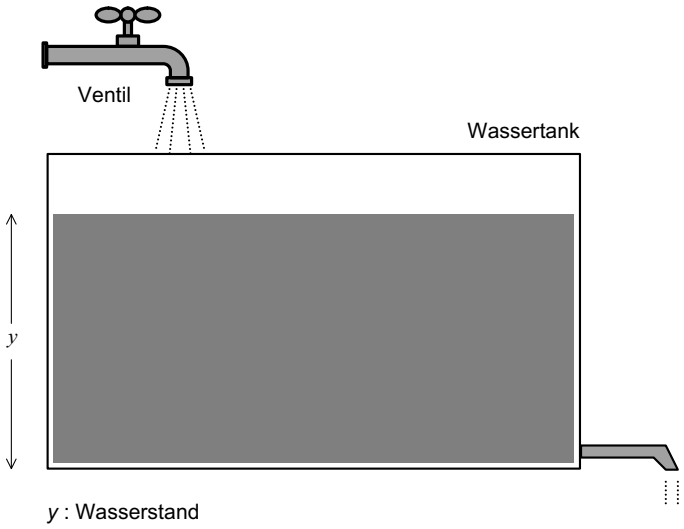


Abbildung 5.19: System für die Füllstandsregelung.

Wasserstand beträgt 1 Einheit. Durch die Steuerung muß der Wasserstand zwischen 1 und 12 Einheiten gehalten werden. Das Öffnen und Schließen des Ventils benötigen jeweils 2 Zeiteinheiten. Dadurch muß mit dem Öffnen beim Wasserstand 5 begonnen werden und mit dem Schließen beim Wasserstand 10.

Der hybride Automat hat vier Zustände. Die Uhr x mißt in den Zuständen *Ventil_schließt* und *Ventil_öffnet* die Dauer des Stellvorgangs. Nach 2 Zeiteinheiten geht der Automat in den nächsten Zustand über. In den Zuständen *Ventil_schließt* und *Ventil_zu* wird somit ein geschlossenes Ventil angestrebt und in den Zuständen *Ventil_öffnet* und *Ventil_offen* entsprechend ein geöffnetes Ventil.

Wie in Abbildung 5.21 auf der nächsten Seite dargestellt, wird in diesem Beispiel die Erreichbarkeitsanalyse in Form eines Invariantentests formuliert. Dabei wird nicht wie im vorhergehenden Beispiel überprüft, ob unerwünschte Konfigurationen erreichbar sind, sondern vielmehr, ob eine Bedingung (Invariante) für alle erreichbaren Konfigurationen gilt. Diese alternative Sichtweise der Analyse ändert an der Verifikation prinzipiell nichts. Daß die Teilmengenbeziehung effizienter analysiert werden kann als ein Durchschnitt (zuzüglich des Tests auf Leerheit mit konstantem Zeitaufwand), ist praktisch nicht von Interesse, da der Hauptaufwand der Analyse bei der Berechnung der erreichbaren Konfigurationen auftritt.

Auch in diesem Beispiel liegt die Ausführungszeit für die gesamte Analyse unter einer Sekunde (110 ms). Für die Berechnung aller erreichbarer Konfigurationen sind 8 Nachfolgerberechnungen notwendig.

5.4.1.3 Fischers Protokoll – abweichende Uhren

Für das in Abschnitt 3.3.4 vorgestellte Protokoll für gegenseitigen Ausschluß (vgl. Abbildung 3.11 auf Seite 74) wird ein hybrides Modell vorgestellt. Um das Beispiel interessanter zu machen, wird in der Literatur eine Modellierung vorgeschlagen, bei der die

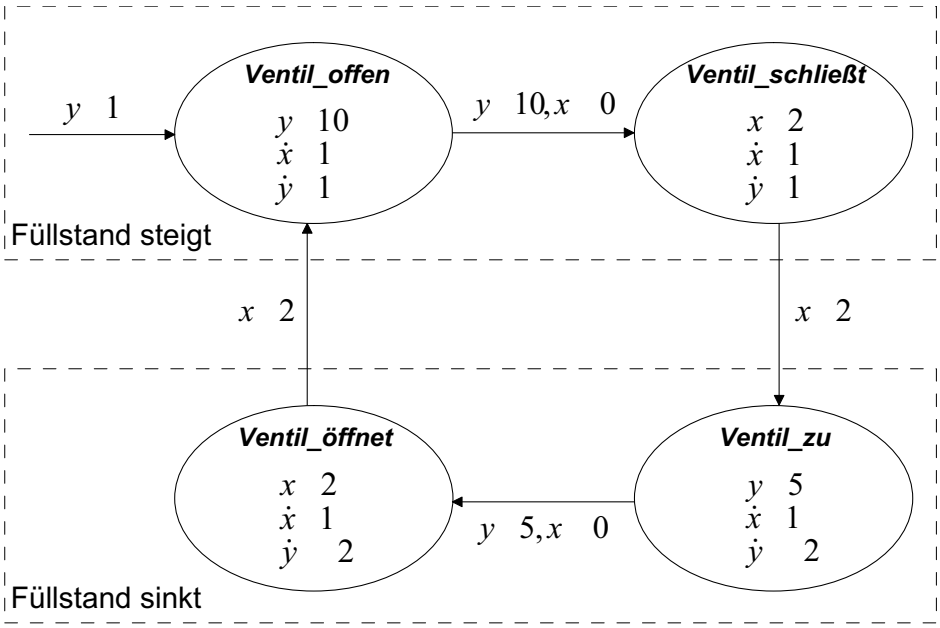


Abbildung 5.20: Hybrides Modell einer Füllstandssteuerung (nach [ACH+95]).

```
 1 REACHABILITY CHECK Waterlevel
 2 {
 3   VAR
 4     initial, error, property, reachable: REGION;
 5   COMMANDS
 6     initial   := STATE(WaterLevel) = Ventil_offen AND y = 1;
 7     property  := (y >= 1) INTERSECT (y <= 12);
 8     reachable := REACH FROM initial FORWARD;
 9     IF(property CONTAINS reachable)
10     {
11       PRINT " Invariant satisfied. ";
12     }
13     ELSE
14     {
15       PRINT " Invariant violated. ";
16     }
17 }
```

Abbildung 5.21: Invariantentest der Füllstandssteuerung.

lokalen Uhren der Prozesse voneinander abweichen [ACH+95]. So könnte z. B. die Uhr des ersten Prozesses etwas langsamer und die Uhr der zweiten Prozesses etwas schneller als eine Referenz-Uhr laufen. Durch die Nutzung hybrider Automaten ist dieser Sachverhalt leicht durch die analogen Variablen x_i mit Ableitungen ungleich 1 zu modellieren (siehe Abbildung 5.22 auf der nächsten Seite).

Die Sicherheitseigenschaft des gegenseitigen Ausschlusses ist in diesem Modell nur dann gewährleistet, wenn der Prozeß mit der schnellsten Uhr im Zustand *Wait* solange wartet, bis der Prozeß mit der langsamsten Uhr aus dem Zustand *Assign* in den Zustand *Wait* übergegangen ist.

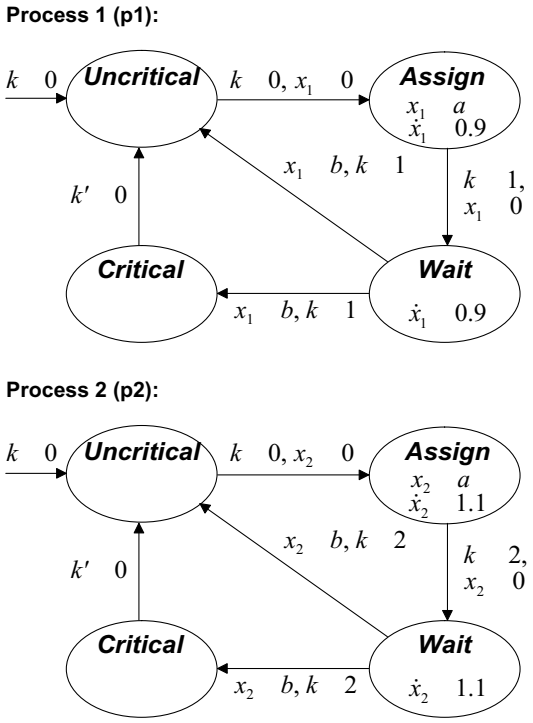


Abbildung 5.22: Hybrides Modell für Fischers Protokoll für gegenseitigen Ausschluß.

Anzahl Prozesse	2	3	4	5	6
Verifikationszeit (Sekunden)	0,06	0,49	6,77	183	17041
Speicherbedarf (MB)	1,2	4,1	10,1	53,9	541
Anzahl Zustände Produktautomat	16	64	256	1024	4096
Anzahl Transitionen Produktautomat	40	240	1280	6400	30720
Anzahl Iterationen bis Fixpunkt	9	15	21	27	33
Anzahl geschalteter diskreter Transitionen	50	291	1376	5805	22812

Tabelle 5.1: Analyseaufwand beim hybriden Modell von Fischers Protokoll.

In Tabelle 5.1 werden die Verifikationszeiten und der Speicherbedarf[1] für den Nachweis der Ausschlußeigenschaft beim Protokoll für 2 bis 6 Prozesse[2] mit gleichlaufenden Uhren angegeben (Linux-Rechner mit Prozessor AMD Athlon, 1 GHz Taktfrequenz, 1.2 GB Hauptspeicher). Der Analyseaufwand für dieses Modell wächst stärker als exponentiell in Abhängigkeit von der Anzahl der Prozesse (bzw. Automaten). Dies resultiert aus der expliziten Speicherung der diskreten Kontrollzustände in der Erreichbarkeitsmenge. Um den gegenseitigen Ausschluß zu gewährleisten, muß die Konstante b einen größeren Wert erhalten als die Konstante a.

[1]Der in der Tabelle angegebene Speicherbedarf wurde mit dem Programm 'memtime v1.2' von Johan Bengtsson gemessen, dem an dieser Stelle für dieses kleine nützliche Werkzeug gedankt sei.

[2]Das Werkzeug HyTech ist in der Lage, 6 Prozesse mit kürzerer Laufzeit zu analysieren. Ein Vergleich ist jedoch nicht möglich, da HyTech auf einer wesentlich vereinfachten Semantik beruht, die hier nicht verwendet wird.

```
MODULE Process {
  INPUT
    processNo:  CONST;   // Identification number of proccess.
                         // Constants for time bounds.
    a:          CONST;   // Maximal time the modeled assignment to k needs.
    b:          CONST;   // Minimal time waiting for assignments of other processes.
  MULTREST
    k:  DISCRETE;        // Shared variable for announcement.
  LOCAL
    x:  CLOCK;
  INITIAL STATE(Fischer) = uncritical AND k = 0;
  AUTOMATON Fischer {
    STATE uncritical { TRANS { GUARD k = 0;
                               DO x' = 0;
                               GOTO assign; }
                     }
    STATE assign     { INV x <= a;
                       TRANS { DO x' = 0 AND k' = processNo;
                               GOTO wait; }
                     }
    STATE wait       { TRANS { GUARD x >= b AND k != processNo;
                               DO x' = 0;
                               GOTO uncritical; }
                       TRANS { GUARD x > b AND k = processNo;
                               DO x' = 0;
                               GOTO critical; }
                     }
    STATE critical   { TRANS { DO k' = 0 AND x' = 0;
                               GOTO uncritical; }
                     }
  }
}
MODULE System {
  LOCAL
    a = 3 : CONST;
    b = 3 : CONST;
    pNo1 = 1: CONST;
    pNo2 = 2: CONST;
  LOCAL
    k:     DISCRETE;
  INST Proc1 FROM Process WITH {
    a         AS a;
    b         AS b;
    processNo AS pNo1;
    k         AS k;
  }
  INST Proc2 FROM Process WITH {
    a         AS a;
    b         AS b;
    processNo AS pNo2;
    k         AS k;
  }
}
```

Abbildung 5.23: CTA-Quelltext des Modells für Fischers Protokoll (hybride Version).

Die textuelle Repräsentation des Fischer-Modells wird in Abbildung 5.23 aufgeführt. Nicht explizit definierte Invarianten und Wächter werden als $true$ verstanden. In diesem Modell kommt das Instanziierungskonzept der CTA-Modellierungsnotation zur Anwendung, um auf einfache Weise mehrere sich nur in der Prozeß-Identifikationsnummer unterscheidende Prozesse zu instanziieren. Dieses Konzept ermöglicht auch bei größeren Modellen ein effizientes Modellieren, worauf im nächsten Kapitel genauer eingegangen wird.

5.4.1.4 Steuerung Bahnübergang

Der Vollständigkeit halber wird hier das Beispiel der Steuerung eines Bahnübergangs erwähnt, wofür das Modell in den Abschnitten 4.1.2 und 4.1.7 beschrieben ist. Die Erklärung des zugehörigen Analyseabschnitts folgte in Abschnitt 5.3.1 als Beispiel zur Verifikationsnotation. Da dieses System aus wenigen Komponenten besteht, wird zur Be-

rechnung aller erreichbaren Konfigurationen weniger als eine Sekunde (100 ms) benötigt, wobei für die Nachfolgerberechnung 14 diskrete Transitionen notwendig sind.

5.4.1.5 Performanceprobleme bei hybriden Systemen

Die Effizienz der verwendeten Datenstruktur für hybride Automaten könnte in einigen Punkten verbessert werden: Einerseits könnte die Technik eingesetzt werden, daß nur die Variablen in einer Transition berücksichtigt werden, die an der Wertveränderung beteiligt sind. Andererseits müßte untersucht werden, inwieweit eine echte On-the-fly-Analyse (ohne Speicherung der vollen Erreichbarkeitsmenge) einen Effizienzgewinn bringt. Die Minimierung des Transitionssystems bzw. des Produktautomaten wurde als weitere Technik zur Effizienzsteigerung in der Literatur ausgiebig untersucht, löst jedoch die grundsätzlichen Probleme der Analyse hybrider Modelle nicht. Zur Speicherplatzersparnis kann auf die Konstruktion des Produktautomaten verzichtet werden, indem der Produktautomat implizit im Speicher gehalten wird; es muß dann jedoch nachgewiesen werden, daß der Mehraufwand bei den Analysealgorithmen den Vorteil nicht aufwiegt.

Ein Nachteil der vorliegenden Implementierung entsteht durch die verwendete CTA-Semantik: Zustände mit falscher Invariante können nicht weggestrichen werden (wie dies z. B. bei HyTech der Fall ist). Diese Semantik ist wichtig, um das im Abschnitt 4.2.1 besprochene Problem der von anderen Automaten erzwungenen 'Deadlocks' zu lösen.

Solange die diskreten Kontrollzustände in der Erreichbarkeitsmenge explizit abgespeichert werden, kann für hybride Automaten keine effiziente Erreichbarkeitsanalyse erwartet werden, da die Menge der erreichbaren Konfigurationen mit exponentiellem Platzaufwand gespeichert wird.

5.4.2 Realzeit-Beispiele und Performance-Vergleich

Die Performance-Ergebnisse in diesem Abschnitt resultieren aus eigenen Experimenten auf einem Linux-Rechner, der mit einem Prozessor AMD Athlon mit 1 GHz Taktfrequenz und 1,2 GB Hauptspeicher ausgestattet ist. Für die Werkzeugimplementierung Rabbit wurde der verfügbare Arbeitsspeicher für die BDD-basierten Berechnungen auf maximal 400 MB beschränkt. Die in Tabellen und Diagrammen dargestellten Meßergebnisse für Fremdwerkzeuge stammen ebenfalls aus eigenen Experimenten (ohne Speicherbegrenzung).

Die Verifikationszeiten der Matrix-basierten Werkzeugen hängen stark von der Wahl bestimmter Parameter ab, zu der viel Erfahrung im Umgang mit dem Werkzeug und Kenntnisse der Implementierung notwendig sind. Deshalb werden hier nicht selbst ermittelte, sondern die von den Programmierern der entsprechenden Werkzeuge veröffentlichten Parameter für die Experimente genutzt. Teilweise (z. B. bei der BDD-basierten Version von Kronos) sind die Versionen der Werkzeuge mit den besten Performance-Resultaten nicht freigegeben und somit für externe Experimente nicht verfügbar; dann wurden die besten veröffentlichten Resultate zum Vergleich verwendet.

Auch wurden in diesem Abschnitt nur solche Beispielmodelle herangezogen, die die Entwickler der Werkzeuge anderer Forschungsgruppen in ihren Artikeln für Benchmark-

# Prozesse	4	5	6	7	8	10	12	14	16	32	64	128
Kronos	0.83	52.6										
Uppaal	0.06	1.44	181	32488								
Kronos Appr.	0.31	0.90	4.80	28.5	172							
Uppaal Appr.	0.02	0.07	0.33	1.45	6.24	158						
RED, Wang	1.64	6.78	21.7	60.7	168	1400						
Rabbit	0.04	0.08	0.15	0.26	0.50	1.35	1.61	3.81	6.50	61.4	559	5200

Tabelle 5.2: Berechnungszeiten zur Verifikation von Fischers Protokoll (in Sekunden)*†.

Vergleiche benutzen und mit denen ihre Werkzeuge gute Ergebnisse erzielen. Auf Modelle, die bei anderen Werkzeugen pathologisch wirken, wird beim Vergleich verzichtet.

Für die Experimente mit Uppaal wurde die im Januar 2002 verfügbare Version verwendet: Uppaal2k 3.2.4. Um bestmögliche Performance zu erzielen, wurden die auf der Uppaal-Internetseite von den Entwicklern angegebenen Parameter verwendet[3]. Die aktuellen Entwicklungen bei Uppaal wurden ausführlich dokumentiert [ABB+01].

Die Werkzeuge Uppaal und Kronos bieten eine Über-Approximation an, die z. B. bei Fischers Protokoll zur Verifikation der Mutex-Eigenschaft anwendbar ist. Bei der Erreichbarkeitsanalyse mit Über-Approximation werden mehr Konfigurationen berechnet, als in Wirklichkeit erreichbar sind. Sind bei Über-Approximation bestimmte Konfigurationen nicht erreichbar, dann können diese bei der normalen Analyse auch nicht erreichbar sein. Für eine Beschreibung der Approximation durch konvexe Hüllen wird auf Aufsätze anderer Forschungsgruppen verwiesen [Hal93, WT94, Bal96a, DT98].

5.4.2.1 Fischers Protokoll für gegenseitigen Ausschluß – Realzeit-Version

In diesem Abschnitt werden die Meßergebnisse von Vergleichsexperimenten vorgestellt, bei denen das Protokoll für den gegenseitigen Ausschluß von mehreren um eine gemeinsame Ressource konkurrierenden Prozessen verifiziert wird [Lam87]. Der Timed Automaton für einen solchen Prozeß wurde bereits in Abschnitt 3.3.4 vorgestellt.

Tabelle 5.2 gibt die Meßergebnisse für die Verifikation der Eigenschaft des gegenseitigen Ausschlusses für verschiedene Anzahlen von Prozessen (erste Zeile der Tabelle) für mehrere Werkzeuge wieder. Performance-Ergebnisse aus der Literatur, die nicht durch eigene Messungen nachvollzogen werden konnten, werden im Text wiedergegeben.

In der zweiten und dritten Zeile werden die Berechnungszeiten der Werkzeuge *Kronos* und *Uppaal* angegeben. Die Berechnungszeiten für diese Werkzeuge sind stärker als exponentiell von der Anzahl der Prozesse abhängig. Die besten veröffentlichten Performancemessungen von Uppaal beruhen auf einer erweiterten Version, die auf kompositionellen und symbolischen Techniken basiert [LPY95a, LPY95b]. Die Ergebnisse sind besser als die in der Tabelle aufgeführten, jedoch auch nicht polynomiell: 18,8 Sekunden für 6 Prozesse, 145,0 Sekunden für 7 Prozesse und 1107 Sekunden für 8 Prozesse. Uppaal

[3] http://www.docs.uu.se/docs/rtmv/uppaal/benchmarks/

*Bei keiner Angabe benötigte die Verifikation mehr als 400 MB Speicher oder länger als 7200 s.

†Rabbit kann 128 Prozesse mit nur 200 MB Speicher bei einer Berechnungszeit von 5367 s verifizieren.

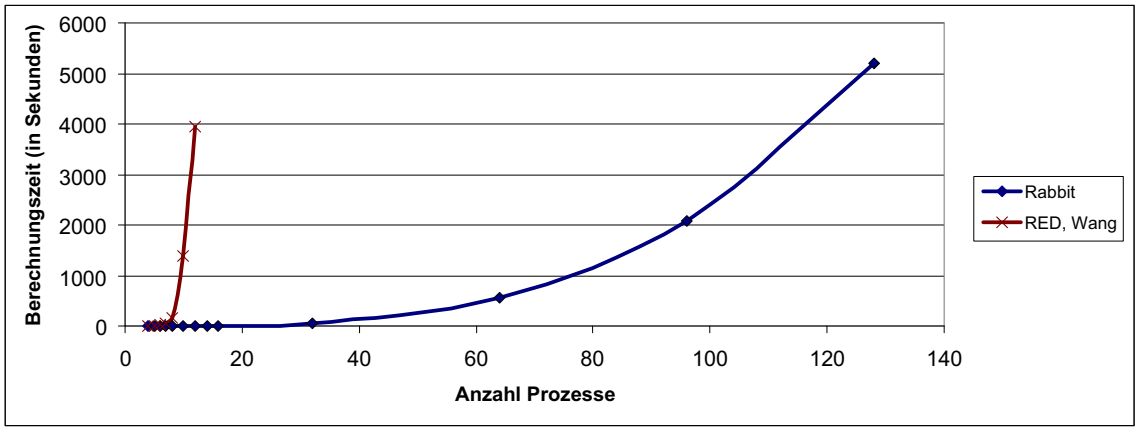


Abbildung 5.24: Laufzeiten in Abhängigkeit von der Anzahl der Fischer-Prozesse.

ist diesen Quellen zufolge in der Lage, die Eigenschaft des gegenseitigen Ausschlusses für bis zu 9 Prozesse des Fischer-Protokolls zu verifizieren (ohne Zeitangabe).

Die Verifikation mit den Werkzeugen Uppaal und Kronos wird interessanter, wenn die für dieses Modell zulässige Approximation (konvexe Hülle) benutzt wird. In der Tabelle 5.2 auf der vorherigen Seite werden die Berechnungszeiten für beide Werkzeuge angegeben (in der Tabelle abgekürzt mit 'Appr.'). Jedoch ist schon bei wenigen Prozessen die Grenze der praktischen Nutzbarkeit erreicht. Uppaal konnte mit Approximation die Verifikation von 11 Fischer-Prozessen in 3727 Sekunden durchführen (auf einer Sun Ultra-Sparc IIi mit 300 MHz unter Nutzung von 710 MB Speicher).

Farn Wang implementierte das Werkzeug *RED* (Version 3.1) für eine BDD-ähnliche Datenstruktur, die von ihm CRD (Clock Restriction Diagram) genannt wird [Wan01]. Er selbst berichtet über die Verifikation von 17 Prozessen in 15.330 Sekunden auf einem Pentium II mit 366 MHz und 256 MB Speicher [Wan00]. Dieses Ergebnis konnte mit der frei verfügbaren Version des Werkzeugs auf einem schnelleren Rechner mit 1 GHz jedoch nicht nachvollzogen werden. Die in eigenen Experimenten nachgewiesenen Berechnungszeiten sind in der vorletzten Zeile der Tabelle aufgeführt und wurden in der Abbildung 5.24 zur Gegenüberstellung mit Rabbit genutzt. Bei der Arbeit von Farn Wang ist die Einschränkung zu beachten, daß ein System als skalierbare Menge symmetrischer Prozesse aufgefaßt werden muß, was einen sehr speziellen Sonderfall darstellt [Wan00]. Modelle, die aus einer skalierbaren Anzahl von Prozessen bestehen, weisen meist eine weniger komplexe Struktur auf.

Für das Werkzeug *Rabbit* wurde bei diesem Protokoll ein polynomieller Aufwand ermittelt (letzte Zeile der Tabelle). Dadurch kann Rabbit das Modell von Fischers Protokoll fast so effizient analysieren wie das auf rein kompositionelle Beweistechniken beruhende Werkzeug *CMC* von Kristoffersen et al. [KLL+97]. Dieses Werkzeug kann den gegenseitigen Ausschluß für 50 Prozesse von Fischers Protokoll in 172 Sekunden beweisen.

Die *BDD-basierte Version von Kronos* ist in der Lage, 14 Prozesse zu verifizieren (auf einer SUN Ultra-Sparc 1, siehe [BMPY97]), was allerdings auch exponentiellen Aufwand vermuten läßt.

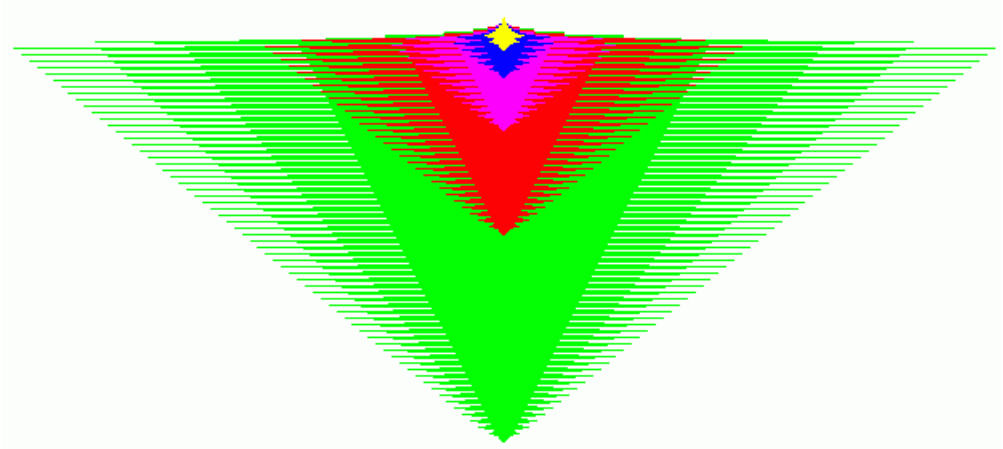


Abbildung 5.25: BDD-Gestalt in Abhängigkeit von der Anzahl der Fischer-Prozesse (Maximalbreite des größten BDDs: 1058 BDD-Knoten).

Das Werkzeug *Timed COSPAN* von Alur und Kurshan kann bis zu 10 Prozesse in 720 s verifizieren, wobei als Grundlage eine BDD-basierte Repräsentation des Regionenautomaten dient [AK96].

Durch die BDD-Visualisierung in Abbildung 5.25 wird der polynomiell mit der Anzahl der Prozesse ansteigende Speicherbedarf der Erreichbarkeitsmenge bei Rabbit veranschaulicht. Dargestellt werden die Gestalten für 4, 8, 16, 32 und 64 Prozesse in wechselnden Graustufen (für eine Erklärung der Visualisierung siehe Abbildung 3.15 auf Seite 78).

Wie aus der Tabelle 5.2 und der Abbildung 5.24 ersichtlich ist, kann die Sicherheitseigenschaft mittels Model-Checking nur vom Werkzeug Rabbit in akzeptabler Zeit überprüft werden. Dieser Erfolg liegt in erster Linie in der komprimierten Repräsentation der Transitionsrelation und der Erreichbarkeitsmenge begründet, die wiederum auf das Ordnen der Variablen innerhalb des repräsentierenden BDDs zurückzuführen ist (siehe Abschnitt 3.3.5 für den Vergleich verschiedener Variablenordnungen bei Fischers Protokoll).

5.4.2.2 Modell eines AND-Schaltkreises

In den Abschnitten 2.2.1 und 2.3.2 wurde zum Veranschaulichen der Modellierungsmöglichkeiten das CTA-Modell eines MOS-Schaltkreises für die AND-Funktion beschrieben. In diesem Abschnitt werden Performance-Ergebnisse für dieses Modell vorgestellt und verglichen. Dieses Modell weist eine weniger regelmäßige Struktur auf und hat einen stärker verbundenen Kommunikationsgraphen als das Modell für Fischers Protokoll. Um das Beispiel skalieren zu können, wurden Modelle mit unterschiedlicher Anzahl von Eingängen des AND-Gatters erarbeitet.

Eine zu analysierende Eigenschaft eines solchen Schaltungsmodells ist, ob ein Kurzschluß auftreten kann, d. h. ob es eine Schaltsituation gibt, bei der zwei dual arbeitende Transistoren gleichzeitig eine geöffnete Kollektor-Emitter-Strecke haben. Andererseits kann die maximale Anzahl gleichzeitig schaltender Transistoren von Interesse sein, da die maximale Stromaufnahme einer solchen Schaltung proportional von der Anzahl der

gleichzeitigen Umschaltvorgänge abhängt. Für diese Analysen muß die Menge aller erreichbaren Konfigurationen berechnet werden, was gleichzeitig die aufwendigste Operation der ganzen Analyse ist. Die konkrete Ausprägung einer bestimmten Verifikation besteht in der Formulierung einer Menge der interessierenden Konfigurationen und die Berechnung des Durchschnitts dieser Menge mit der Erreichbarkeitsmenge.

Anzahl der Eingänge	2	4	8	16
Berechnungszeit, GTR	0.34	5.34	107	1305
Berechnungszeit, TVTR + THTR	0.21	2.44	38.2	426

Tabelle 5.3: Laufzeiten in Sekunden für die Berechnung aller erreichbaren Konfigurationen für das AND-Modell mit dem Werkzeug Rabbit.

In der Tabelle 5.3 werden die Zeiten zum Berechnen aller erreichbaren Konfigurationen mit Rabbit aufgeführt, wobei Modelle mit unterschiedlicher Anzahl von Eingängen für den Schaltkreis benutzt wurden (erste Zeile in der Tabelle). In der zweiten Zeile der Tabelle werden die Rechenzeiten unter Benutzung der On-the-fly-Analyse mit der Standard-Transitionsrelation angegeben. Wird für die Berechnung der Repräsentation der Transitionsrelation die transitive Hülle (siehe Abschnitt 3.5.2.1) der teilweise vereinigten Transitionsrelation (siehe Abschnitt 3.5.2.2) verwendet, können die in der dritten Zeile dargestellten Ergebnisse erzielt werden.

Dieses Modell wurde auch von Bozga et al. als Performance-Beispiel genutzt [BMPY97]. Dabei benötigt die BDD-basierte Version von Kronos benötigt für ein AND-Modell mit 4 Eingängen eine Berechnungszeit von 324.7 Sekunden (Prozessor SUN Ultra-Sparc 1). Diese Aufgabe wird von Rabbit in nur 2.44 bzw. 5.34 Sekunden bewältigt.

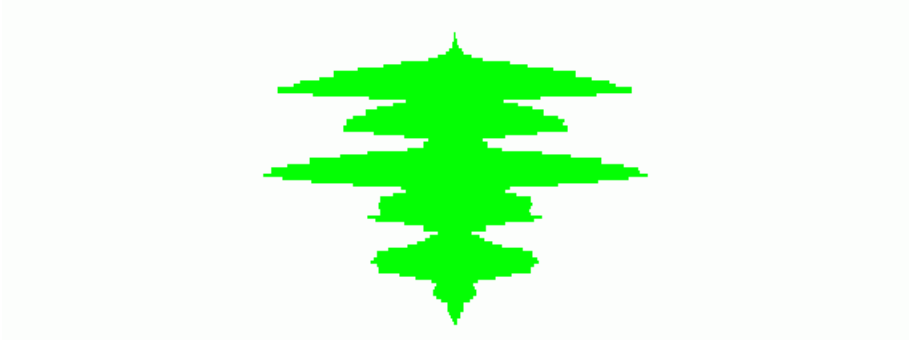


Abbildung 5.26: Gestalt des BDDs der Erreichbarkeitsmenge beim AND-Modell mit 4 Eingängen (Maximalbreite des BDD: 464 Knoten, Gesamtgröße des BDD: 15267 Knoten).

In Abbildung 5.26 wird die Gestalt des BDDs der Erreichbarkeitsmenge für das Modell der AND-Schaltung mit 4 Eingängen dargestellt. In diesem Modell existieren fünf Bereiche mit starker Kommunikation zwischen den Komponenten, was zu einem starken lokalen Breitenzuwachs dieser Bereiche des BDDs führt.

5.4.2.3 Mini-Automaten mit lokalen Uhren

Das folgende Beispiel dient als Benchmark zur Überprüfung, wie gut ein Werkzeug mit mehreren voneinander unabhängigen Realzeit-Komponenten umgehen kann. Der Automat für eine solche Komponente wurde bereits in der Abbildung 3.23 auf Seite 94 vorgestellt.

Anzahl Komp.	4	6	8	16	32	64	128
Anzahl Konfig.	$3.3 \cdot 10^5$	$1.9 \cdot 10^8$	$1.1 \cdot 10^{11}$	$1.2 \cdot 10^{22}$	$1.5 \cdot 10^{44}$	$2.2 \cdot 10^{88}$	$4.8 \cdot 10^{176}$
Mocha	4.74	361	>7200				
Rabbit	0.06	0.10	0.19	1.23	6.11	26.8	123

Tabelle 5.4: Laufzeit zur Berechnung aller erreichbarer Konfigurationen für das Modell mit zwei Zuständen und einer Uhr je Komponente.

In der Tabelle 5.4 werden die Daten von Experimenten mit Rabbit und Mocha, ebenfalls ein BDD-basiertes Werkzeug für Model-Checking, aufgeführt. Hier wird die Version c-Mocha verwendet, da diese Realzeit-Modelle unterstützt. In der ersten Zeile wird die Anzahl der Komponenten im Modell, in der zweiten die Anzahl der Konfigurationen in der Erreichbarkeitsmenge angegeben. Die Berechnungszeiten von Mocha sind in der dritten Zeile und die Laufzeiten der On-the-fly-Berechnung der Erreichbarkeitsmenge von Rabbit in der vierten Zeile wiedergegeben (vgl. Tabelle 3.4 auf Seite 94 für die Berechnungszeit von Rabbit bei 256 Komponenten).

Die BDD-basierte Version von Kronos benötigt 20.000 Sekunden für die Berechnung der Erreichbarkeitsmenge für 9 Komponenten [BMPY97].

Die BDD-Repräsentation für dieses Modell hat eine interessante Eigenschaft: Die Menge aller erreichbaren Konfigurationen hat für dieses Modell eine lineare Speicherkomplexität (in Übereinstimmung mit der Größenschätzung, weil die Menge $\mathit{Comm}_{\mathcal{A}}(i)$ nach Abschnitt 3.3.2 für alle Komponenten i leer ist). Jedoch wachsen die Zwischen-BDDs innerhalb der Fixpunkt-Iteration sehr stark (vgl. Abbildung 3.24 auf Seite 94). Dieses Problem konnte mit der On-the-fly-Analyse aus Abschnitt 3.5.4 gelöst werden. Die Größe des jeweiligen BDDs für die Erreichbarkeitsmenge kann dort der Tabelle 3.4 auf Seite 94 entnommen werden.

5.4.2.4 Token-Ring-FDDI-Protokoll

Fiber Distributed Data Interface (FDDI) ist ein Hochgeschwindigkeitsprotokoll für lokale Netzwerke (LAN), die auf Glasfaser-Optik beruhen und nach dem Konzept des Token-Rings arbeiten [Jai94, SR94]. Ein solches FDDI-Netzwerk besteht aus n identischen Stationen und einem Ring, über den die Stationen miteinander kommunizieren können. Es wird unterschieden zwischen *synchronen* Nachrichten mit hoher Priorität und *asynchronen* Nachrichten mit niedriger Priorität. Bei diesem Protokoll stellt sich als Verifikationsaufgabe z. B. die Frage nach der *begrenzten Zeit für den Zugriff auf den Ring*. D. h. es soll nachgewiesen werden, daß die Zeit zwischen zwei aufeinanderfolgenden Token-Zugriffen einer Station durch eine Konstante begrenzt ist, die von der Anzahl der beteiligten Stationen abhängt.

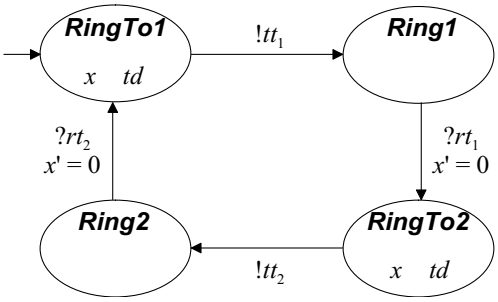


Abbildung 5.27: Modell für den Ring im FDDI-Protokoll für zwei Stationen.

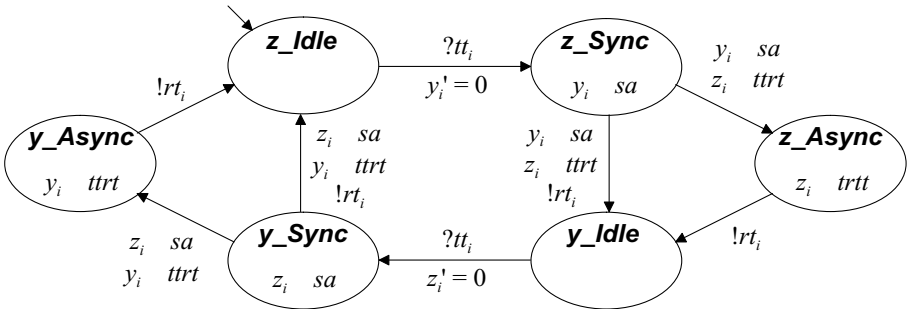


Abbildung 5.28: Modell für die i-te Station im FDDI-Protokoll.

Abbildung 5.27 gibt den Automaten für den Ring wieder. Der Ring funktioniert in der folgenden Weise: Nach einer Verzögerung bei der Token-Weitergabe im Ring von td Zeiteinheiten (*token delay*) wird der nächsten Station i durch das Signal tt_i (*take token*) das Token übergeben. Zum Messen der Verzögerung bedient sich der Automat einer Uhr x. Nachdem die Station i die Datenübertragung in synchronem oder asynchronem Modus beendet hat, wird durch das Signal rt_i (*release token*) das Token an den Ring zurückgegeben. Wiederum nach einer Verzögerung td des Rings bekommt die Station $i + 1 \ mod \ n$ das Token usw. Die Konstante td spielt für die Verifikationsaufgabe eine untergeordnete Rolle.

Der Automat für die i-te Station ist in Abbildung 5.28 dargestellt[4] (siehe auch [Yov98]). Für die Station i existieren zwei Uhren y_i und z_i, die jeweils abwechselnd die Token-Umlaufzeit messen. Der Automat befindet sich in einem der drei Zustände *auf Token wartend*, *im synchronen* oder *im asynchronen Übertragungsmodus*. Diese drei Zustände werden den Rollen der Uhren entsprechend unterteilt in sechs Zustände des Automaten: In den Zuständen *z_Idle*, *z_Sync* und *z_Async* mißt die Uhr z_i die seit dem Auftreten des Signals tt_i in der letzten Runde vergangene Zeit, die Uhr y_i wird beim Auftreten des Signals tt_i auf den Wert 0 gesetzt und mißt im Zustand *z_Sync* die Zeitdauer der synchronen Übertragung von Nachrichten hoher Priorität, die durch die Konstante sa begrenzt ist. Von nun an wird die Uhr y_i in dieser Runde nicht mehr zurückgesetzt, damit sie die Zeit seit dem letzten Auftreten des Signals tt_i für die nächste Runde messen kann. In den Zuständen *y_Idle*, *y_Sync* und *y_Async* tauschen die Uhren y_i und z_i die Rollen: y_i mißt die

[4]Dieses Automaten-Modell des FDDI-Senders orientiert sich an dem für Uppaal entwickelten Modell von Oliver Möller.

Zeit seit dem Auftreten von tt_i in der letzten Runde und z_i mißt die Zeit seit dem Übergang von *y_Idle* nach *y_Sync*.

Eine Station darf nach der synchronen Übertragung in den asynchronen Modus gehen, falls die Umlaufzeit des Tokens ihren Maximalwert $ttrt$ (*target token rotation time*) noch nicht erreicht hat. Hier darf die Station solange Nachrichten niedriger Priorität senden, bis der Maximalwert für die asynchrone Übertragungszeit erreicht ist, oder sie kann das Token vorher an den Ring zurückgeben. (Die Konstanten im Modell wurden mit folgenden Werten belegt: $td = 0, sa = 1, ttrt = 2n + 1$.)

Komplexitätsuntersuchung des Speicheraufwands für die Erreichbarkeitsmenge. Erstmals in der Literatur[5] wird hier eine Komplexitätsabschätzung zur Größe der Erreichbarkeitsmenge argumentiert (in der in Abbildung 3.26 auf Seite 97 eingeführten Form). Dabei wird nach einer ähnlichen Strategie vorgegangen wie in Abschnitt 3.3.2.

Behauptung 5.1 *Die Speicherkomplexität für die BDD-Repräsentation der Erreichbarkeitsmenge für das FDDI-Protokoll liegt in $O(n^4 \log\ n)$, wobei n die Anzahl der beteiligten Stationen ist.*

Begründung: Zunächst wird die Tiefe des BDDs untersucht: Für jede Station sind sechs Werte für die Zustände des Automaten und je $2n + 1$ Werte für die beiden Uhren notwendig (bestimmt durch die größte Konstante im Modell). Damit werden $O(\log n)$ Bits zur Kodierung der Variablen einer Komponenten benötigt. Der Automat für den Ring hat $2n$ Zustände und eine Uhr mit einem konstanten Wertebereich ($O(\log n)$ Bits für den Ring). Da im BDD die Werte für n Stationen zu speichern sind, liegt die Gesamt-Tiefe des BDD in $O(n \log n)$.

Die Variablenordnung des BDDs ergibt sich aus der Präfix-Linearisierung des Modells und der Größenschätzung aus Abschnitt 3.3.2 in der folgenden Reihenfolge: Variablen des Rings, Variablen der Station 1, Variablen der Station 2, ..., Variablen der Station n. Sei der BDD an der Ebene direkt unter der letzten Variablen des Rings betrachtet: hier hat der BDD $O(n)$ verschiedene Kofaktoren, für jeden Zustand des Ring-Automaten einen. Diese Kofaktoren unterschieden sich darin, daß eine einzige Station i nicht in einem *Idle*-Zustand auf das Token wartet (dann ist der Ring im Zustand *Ringi*), oder alle Stationen befinden sich in einem *Idle*-Zustand (dann ist der Ring im Zustand *RingToi* und die aktive Uhr der Station i hat den größten Wert). Die unbeteiligten Stationen, die in der Variablenordnung vor Station i liegen, befinden sich entweder alle im Zustand *z_Idle* oder alle im Zustand *y_Idle*; dasselbe gilt für alle hinter i liegenden Stationen. Die Anzahl dieser Kofaktoren ergibt den zweiten linearen Faktor in der Abschätzung.

Sei im folgenden einer dieser Kofaktoren betrachtet. Dann verbleibt die Untersuchung der Abhängigkeiten der Uhren untereinander in diesem einen Kofaktor. Die dafür relevanten Komponenten einer Konfiguration lassen sich in der folgenden Weise vereinfacht schreiben: Bezeichne $(i, x_1, \ldots, x_n)$ eine Konfiguration, wobei i der Wert der Zustandsvariablen für den Ring ist (*Ringi*) und $x_1, \ldots, x_n$ die Werte der jeweils aktiven Uhren der

[5] Da der Analyseaufwand bei den existierenden Werkzeugen aufgrund der Datenstrukturen und Algorithmen für nahezu alle Modelle exponentiell oder stärker wächst, waren bisher auf bestimmte Modelle bezogene Komplexitätsabschätzungen in der Literatur nicht relevant.

Stationen $1, \ldots, n$ bezeichnen. Für die i-te Station sind jeweils zwei Uhren aktiv: x_i^{alt} mißt die Zeit seit dem Empfangen des Tokens in der letzten Runde und x_i^{neu} mißt die Zeit seit dem Empfangen des Tokens in der aktuellen Übertragungsphase. Dann gilt die folgende Abhängigkeit:
$x_i^{alt} > x_{i+1} > \cdots > x_n > x_1 > \cdots > x_{i-1} > x_i^{neu}$.
D. h. nachdem eine Station das Token empfangen hat, ist die Uhr x_i^{neu} gerade zurückgesetzt worden und hat den kleinsten Wert überhaupt. Außerdem hat die Uhr x_i^{alt}, die die Zeit seit dem Empfangen des Tokens in der letzten Runde mißt, den größten Zeitwert überhaupt. Damit ist eine Abhängigkeit zwischen je zwei benachbarten Stationen gegeben.

Diese Abhängigkeit wirkt sich zweimal auf die BDD-Größe aus: Sei die Anzahl der Kofaktoren im BDD an der Position unmittelbar vor der ersten Variable der Station i betrachtet. Dann muß für jeden Wert der aktiven Uhr der vorherigen Station $i-1$ ein Kofaktor existieren, um die "kleiner-als"-Beziehung zur Vorgänger-Station ausdrücken zu können. Außerdem müssen orthogonal dazu alle Werte der aktiven Uhr der Station 1 in Kofaktoren 'durchgereicht' werden, um die "größer-als"-Beziehung der Station n zur Station 1 auszudrücken.

Damit existieren im BDD in der Ebene vor der ersten Variablen der Station i eine Anzahl von Kofaktoren aus $O(n^3)$. Durch Kombination mit der Tiefe folgt die Behauptung.

Anzahl Sender	2	4	6	8	10	12	14	16
Anz. Konfigurationen	106	2420	48330	903784	$1.6 \cdot 10^7$	$2.9 \cdot 10^8$	$5.0 \cdot 10^9$	$8.5 \cdot 10^{10}$
Uppaal	0.01	0.03	0.16	1.42	18.2	279	4530	>7200
Rabbit, Laufzeit	0.04	0.25	0.99	4.20	11.4	26.9	49.8	142
Rabbit, BDD-Größe	350	4211	18501	64505	151737	305523	501305	993673

Tabelle 5.5: Laufzeit und Speicherbedarf für die Berechnung aller erreichbaren Konfigurationen beim Modell des FDDI-Protokolls.

Dieses Ergebnis konnte mit dem Werkzeug Rabbit experimentell nachgewiesen werden. Tabelle 5.5 gibt die Meßgrößen der Verifikation in Abhängigkeit der Anzahl beteiligter Sender (erste Zeile) an. In der zweiten Zeile ist die Anzahl der in der Erreichbarkeitsmenge enthaltenen Konfigurationen aufgeführt. Die von Uppaal benötigte Analysezeit ist in der dritten Zeile enthalten. Für die Experimente mit Rabbit wurden die Laufzeiten der Full-set-Berechnung unter Verwendung der vereinigten Transitionsrelation in der vierten Zeile aufgelistet. Die Anzahl der BDD-Knoten zur Speicherung der Erreichbarkeitsmenge geht aus der letzten Zeile hervor. In den Meßergebnissen spiegelt sich fast exakt die polynomielle Abhängigkeit vierten Grades wieder: Eine Verdopplung der Anzahl der Sender führt zu einer 16fachen Anzahl an BDD-Knoten in der Erreichbarkeitsmenge. □

5.4.2.5 CSMA/CD-Protokoll

In einem Boadcast-Netzwerk mit einem Multi-Access-Medium stellt sich das Problem, nach welchem Schema die Erlaubnis zum Zugriff auf das Medium erteilt werden soll,

wenn mehrere Sender-Stationen Daten übertragen wollen. Eine Lösung für dieses Problem stellt das Protokoll CSMA/CD (Carrier Sense, Multiple-Access with Collision Detection) dar (vgl. [Tan89]).

Wenn ein Sender Daten übermitteln möchte, fragt er das Medium, ob es gerade von einem anderen Sender für eine Übertragung benutzt wird. Wenn nicht, kann der Sender mit der Übertragung beginnen. Wird festgestellt, daß das Medium gerade von einem anderen Sender benutzt wird, muß der Sender eine gewisse Zeitspanne warten und die Übertragung erneut versuchen. Beginnen mehrere Sender nahezu gleichzeitig mit der Datenübertragung, dann tritt eine Kollision auf. Alle Sender erkennen dies und brechen die Übertragung ab. Nach einer zufällig gewählten Wartezeit kann erneut der Zugriff auf das Medium versucht werden.

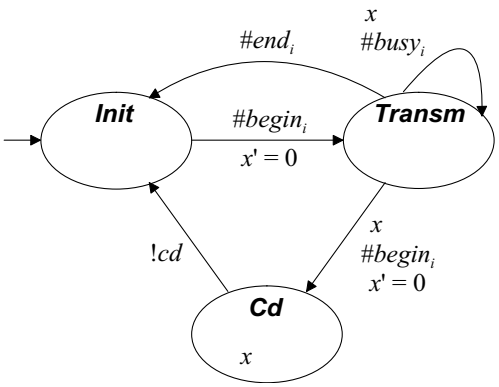


Abbildung 5.29: Modell für das Medium im CSMA/CD-Protokoll (nach [Yov97]).

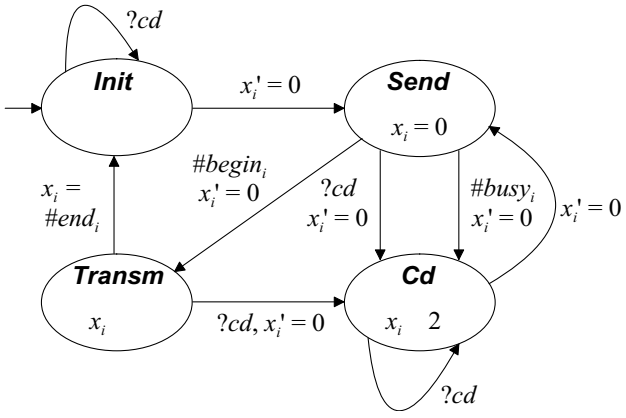


Abbildung 5.30: Modell für die i-te Sender-Station im CSMA/CD-Protokoll.

Das Gesamtmodell für dieses Beispiel besteht aus einem Medium und einer Anzahl von Sendern. Der Automat für das Medium ist in Abbildung 5.29 dargestellt. Im Modell existieren für jeden Sender i die Signale $begin_i$, end_i und $busy_i$ für die oben beschriebene Kommunikation mit dem Medium. Zum Melden einer Kollision existiert ein gemeinsames Signal cd, d. h. cd ist ein globales Signal für alle Sender und das Medium. In der

Anz. Sender	2	4	8	16	32	64	128	256
Anz. Konfig.	71	3471	$4.2 \cdot 10^6$	$3.4 \cdot 10^{12}$	$1.2 \cdot 10^{24}$	$7.9 \cdot 10^{46}$	$1.3 \cdot 10^{92}$	$3.2 \cdot 10^{157}$
Uppaal	0.01	0.03	>7200					
Uppaal, Appr.	0.01	0.02	2.02	>7200				
Kronos, Appr.	0.04	2.65	>7200					
RED, Wang	0.05	0.28	5.88	>7200				
Rabbit, Zeit	0.03	0.09	0.49	2.58	13.1	60.2	349	2160
ReachSet	99	309	729	1569	3249	6609	13329	23460
TransRel	425	1605	6245	24645	97925	390405	1559045	6231045

Tabelle 5.6: Laufzeit zur Berechnung aller erreichbaren Konfigurationen für das Modell des CSMA/CD-Protokolls.

Wartephase befindet sich der Automat im Zustand *Init*. Von dort aus kann der Automat mit der Marke $begin_i$ synchronisieren, falls der Sender i das Medium benutzen möchte. Nach einer gewissen Zeitspanne σ (für das Propagieren über das Netzwerk) kann das Medium die Abfragen anderer Sender durch Synchronisation mit $busy_j$ für den j-ten Sender beantworten. Die Zeit wird gemessen durch die Uhr x. Beginnt der Sender j vor dem Verstreichen dieser Zeitspanne mit der Übertragung, wobei sich das Medium wegen des i-ten Senders bereits im Zustand *Transm* befindet, dann geht das Medium in den Zustand *Cd* über und alle Sender erhalten das Signal cd für eine Kollision.

Jeder Sender ist durch einen Automaten modelliert worden (siehe Abbildung 5.30 auf der vorherigen Seite). Solange sich ein Sender im Zustand *Idle* aufhält, kann er beliebig cd-Signale empfangen, ohne seinen Zustand zu verändern. Möchte der Sender Daten übertragen, wechselt er in den Zustand *Send*. Durch die Invariante des Zustands *Send* wird erzwungen, daß unmittelbar nach Betreten des Zustands in den nächsten Zustand gewechselt wird. Ist das Medium gerade nicht beschäftigt und kann mit dem Signal $begin_i$ synchronisieren, dann wechseln Sender und Medium gemeinsam in der Zustand *Transm*. Hier kann der Sender eine Zeit lang (begrenzt durch die Konstante λ) Daten übertragen. Die Zeit wird von der Uhr x_i gemessen. Mit dem Signal end_i gibt der Sender nach beendigter Übertragung das Medium wieder frei. Betritt jedoch innerhalb einer gewissen Zeitspanne σ (Propagierzeit) ein weiterer Sender den Zustand *Transm* oder war bereits kurz vorher ein anderer Sender in diesem Zustand, dann erhalten alle Sender das cd-Signal und gehen über in den Zustand *Cd*. Von dort aus kann jeweils nach einer Zeit von maximal 2σ Einheiten der Zugriff auf das Medium erneut versucht werden.

Tabelle 5.6 gibt die Meßergebnisse der Experimente mit diesem Modell wieder. In der ersten Zeile sind wieder verschiedene Anzahlen der Sender-Prozesse angegeben, um die das Modell skaliert wurde. Die zweite Zeile beinhaltet die Anzahl der in der Erreichbarkeitsmenge repräsentierten Konfigurationen. Die folgenden drei Zeilen geben die Laufzeiten für die Werkzeuge Uppaal und Kronos an, wobei die Abkürzung 'Appr.' auf die Verwendung der konvexen Hülle hinweist. Uppaal ohne Approximation kann das Modell mit bis zu 7 Sendern in 285 Sekunden verifizieren. Mit Approximation ist der höchste erreichbare Wert 12 Sender in 323 Sekunden (nicht in der Tabelle enthaltene Zwischenwerte für Uppaal mit Approximation: 9 Sender 6.28 s, 10 Sender 19.6 s, 11 Sender 72.7 s,

12 Sender 323 s). Kronos kann mit Approximation bis zu 7 Sender-Prozesse in 1910 s verifizieren.

Für das ebenfalls auf BDD-Techniken basierende Werkzeug RED 3.1 von Farn Wang stehen die Laufzeit-Ergebnisse in der sechsten Zeile. Das größte von diesem Werkzeug handhabbare Modell enthält 13 Sender und kann in einer Zeit von 2390 s verifiziert werden. Selbst diese Meßreihe weist einen exponentiellen Analyseaufwand in Abhängigkeit von der Anzahl der Sender auf.

Für die eigene Implementierung Rabbit wurde die Laufzeit zur Berechnung der Erreichbarkeitsmenge in der siebenten Zeile aufgeführt. In den letzten beiden Zeilen sind die Größen der Erreichbarkeitsmenge (ReachSet) und die Transitionsrelation (TransRel) in BDD-Knoten angegeben.

Bei der Verifikation mit Rabbit tritt folgender Effekt auf: Die Größe des BDDs für die Erreichbarkeitsmenge wächst *linear* mit der Anzahl der beteiligten Sender, weil die Sender voneinander unabhängig sind und nur vom Medium abhängen. Wie aus der vorletzten Zeile der Tabelle ersichtlich, verdoppelt sich die Größe des repräsentierenden BDDs bei Verdopplung der Anzahl der Sender. Dennoch ist der für die Berechnung der Erreichbarkeitsmenge erforderliche *Zeitaufwand nicht linear*. Die Ursache liegt in der Transitionsrelation. In der vorliegenden Implementierung können die Transitionsrelationen für unterschiedliche Signale nicht *während* der Konstruktion der Transitionsrelation vereinigt werden. Da in diesem Modell die Anzahl der Signale linear von der Anzahl der beteiligten Sender abhängt, ist zur Speicherung der Transitionsrelation ein quadratischer Aufwand erforderlich, der sich in der Gesamt-Laufzeit widerspiegelt, und zwar überwiegend bei der Berechnung der Transitionsrelation.

5.5 Zusammenfassung

Der wichtigste Beitrag ist die Effizienzverbesserung bei der Verifikation von Realzeit-Systemen auf der Basis der Timed Automata. Deshalb wird ein Performancevergleich mit den besten verfügbaren Werkzeugen durchgeführt, wobei versucht worden ist, für die Fremdwerkzeuge jeweils die günstigsten Parameter zu finden (z. B. Approximation bei Uppaal und Kronos).

Das Werkzeug Rabbit liefert wesentlich bessere Performance-Resultate als alle vergleichbaren Model-Checking-Werkzeuge für Timed Automata. Dabei wurde nicht nach Modellen gesucht, die für das hier vorgestellte Werkzeug besonders günstig sind. Vielmehr wurden solche Modelle verifiziert, die von den Entwicklern anderer Werkzeuge zur Präsentation ihrer Forschungsergebnisse verwendet wurden. Die Leistung der hier vorgestellten Konzepte liegt nicht in der marginalen Verbesserung der Performance[6], sondern in einer Verbesserung der Analyse-Komplexität einiger Beispiel-Modelle von exponentiell auf polynomiell. Die meisten der behandelten Beispiele können mit polynomiellem Zeit- und Speicherplatz-Aufwand analysiert werden. Damit ist die Implementierung Rabbit der Forderung aus Kapitel 1 nach einem effizienten Werkzeug zur Verifikation auch großer Systeme gerecht geworden (Forderung 3 auf Seite 17).

[6] Beitrag von Frits Vaandrager zur Mailing-Liste CONCURRENCY (Aug. 2000): ”At a recent meeting of the Dutch Association for Theoretical Computer Science, Amir Pnueli gave an overview talk on model checking. He concluded his talk with a transparency stating something like ’The future of model checking is bright’. During the lunch break following Pnueli’s talk, Willem Paul de Roever (always good for some provocative statements) argued that Pnueli was dead wrong. According to Willem Paul model checking is almost dead: ’The key people in the area agree that there are no major breakthroughs to be expected but only minor improvements. Model checking has just not become what people hoped it would be.’ ”

Kapitel 6

Fallstudie Fertigungsanlage

Die bisher in den einzelnen Kapiteln vorgestellten Beispiele sind geeignet, um einige Modellierungsaspekte einfach zu illustrieren. Einige besondere Eigenschaften bei der Analyse können betont und Performance-Vergleiche mit anderen Werkzeug-Implementierungen durchgeführt werden. Diese sogenannten Performance-Benchmarks haben einen entscheidenden Nachteil: Sie sind aus vielen kleinen Komponenten zusammengesetzt, die zum Erreichen der Performancegrenzen notwendige Modellgröße wird dadurch erreicht, daß die Anzahl der beteiligten Prozesse bzw. Komponenten skaliert wird. Dadurch weisen diese Modelle meist eine einfache Struktur auf. Dies wird damit gerechtfertigt, daß einerseits große Modelle mit vielen Variablen und Zuständen benötigt werden und andererseits der Modellierungsaufwand bestimmte Grenzen nicht überschreiten soll.

Aus diesem Grund wird hier die Modellierung und Verifikation einer realen Fertigungsanlage beschrieben, die an Größe und Komplexität die bisher in der Literatur erwähnten Fallstudien im Bereich der auf Timed Automata basierenden Verifikation übertrifft. Die durchgeführte Fallstudie soll belegen, daß die in dieser Arbeit vorgeschlagenen Konzepte und Ergebnisse auch auf große Systeme angewendet werden können, nicht nur auf sogenannte Benchmark-Beispiele. Das Projekt zum Entwickeln des formalen Modells nahm mehrere Monate in Anspruch und erforderte nicht nur das Anwenden der formalen Modellierungstechnik, sondern darüberhinaus auch softwaretechnische Überlegungen, wie es beim ”Programmieren im Großen” erforderlich ist. Die hier vorgestellte Fertigungsanlage ist vom Anlagentyp her der in der KORSO-Fallstudie von Lewerentz und Lindner verwendeten Produktionszelle ähnlich [LL95].

Nach einigen Ausführungen über die Abhängigkeiten zwischen Prozeß-Phasen und Ergebnis-Dokumenten wird die Fertigungsanlage beschrieben. Es wird ausführlich auf die Beschreibung des Systems als CTA-Modell eingegangen. Das Modell wird in physische Umgebung, Steuerung und Ablaufprogramm des Werkstücks gegliedert. In der physischen Umgebung werden die Annahmen über die Hardware fixiert. Steuerung mit Ablaufprogramm sind die Teile, für die in der Praxis ein Software-Programm entwickelt werden würde. Es wird ein Überblick über die Struktur und die verwendeten Modellierungskonzepte gegeben. Da das Modell der Fertigungsanlage in dieser Arbeit wegen des enormen Umfangs nicht im Detail dargestellt werden kann, werden einige wichtige Komponenten exemplarisch genauer erklärt. Die vom Werkzeug Rabbit angebotenen Möglichkeiten für

die Spezifikation und Verifikation werden in einigen Beispiel-Analysen gezeigt. Dabei wird nicht nur auf Erreichbarkeitsanalyse, sondern auch auf Spezifikation mittels abstrakter Modelle und Verfeinerungsanalyse eingegangen.

6.1 Vorbetrachtungen: Dokumente und Prozeß-Phasen

Auch wenn sich diese Arbeit nicht mit der Untersuchung von formale Methoden integrierenden Entwicklungsprozessen beschäftigt, muß auf ein Minimum an Organisation der verschiedenen Arbeitsschritte bei der Erzeugung und Analyse eines so großen Systems zurückgegriffen werden. Ein Prozeßmodell ist auch notwendig, um die Zusammenhänge besser darstellen und Abläufe einordnen zu können. Gründlich untersuchte und verfeinerte Ansätze zur Definition von Prozeßmodellen sind in [SM97] zu finden, [EMD+98] bietet einen Überblick über das KORSYS-Projekt, in dem u. a. formale Methoden unterstützende Prozeßmodelle behandelt worden sind.

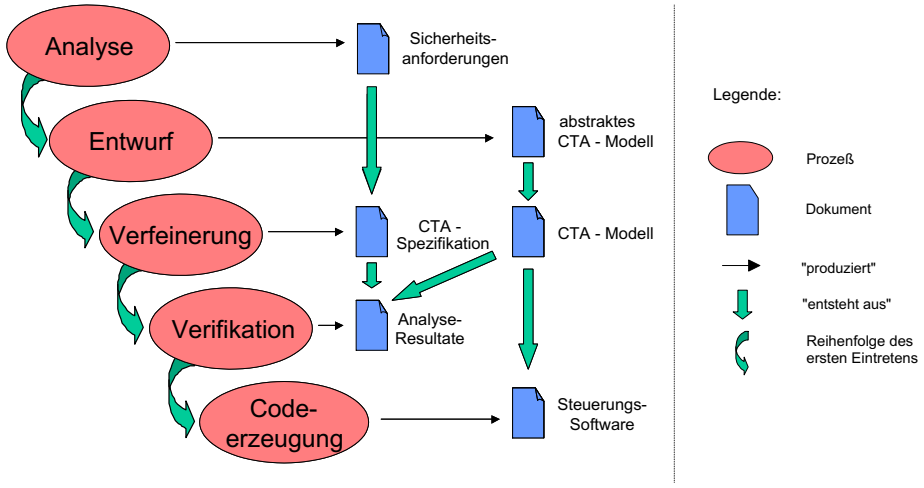


Abbildung 6.1: Relevante Dokumente und ihre Entstehung im Entwicklungsprozeß.

Abbildung 6.1 zeigt einen solchen möglichen Entwicklungsprozeß, der eine grobe Orientierung beim Entwickeln von Steuerungssoftware für eingebettete Realzeit-Systeme mit dem hier beschriebenen Formalismus liefert. Der Prozeß zur Entwicklung einer Steuerung wird zunächst unterteilt in die Schritte Modellbildung, Verifikation, Erzeugung des lauffähigen Programms, wobei die Modellbildung aus Analyse, Entwurf und Verfeinerung besteht. Die Pfeile zwischen den Phasen auf der linken Seite der Abbildung deuten die Reihenfolge des ersten Eintretens in die jeweilige Phase an, d. h. eine Folgephase benötigt Arbeitsergebnisse aus einer vorherigen Phase. Die Arbeitsergebnisse sind durch das Dokumentensymbol horizontal der Phase zugeordnet, in der sie entstehen. Die Pfeile zwischen den Dokumenten zeigen deren Abhängigkeiten untereinander. Dies bedeutet nicht, daß *streng* nach einem Wasserfall-Modell vorgegangen werden muß. Iterationen der früheren Phasen und zyklisches Vorgehen sind notwendig und erwünscht.

In der **Analysephase** werden für ein geplantes System die Anforderungen bzw. Eigenschaften, die während des Entwicklungsprozesses beachtet werden sollen und die letztendlich verifiziert werden müssen, natürlichsprachlich definiert. Dabei darf nur auf die Hardwarekomponenten Bezug genommen werden, da die Architektur der Software-Steuerung noch nicht bekannt ist. Die Eigenschaften beziehen sich dabei also auf die Wechselwirkung der Hardwarekomponenten untereinander, auf die Wirkung der Komponenten auf das zu bearbeitende Werkstück und auf die Sicherheitsanforderungen des Menschen als Bediener der Anlage.

Die **Entwurfsphase** dient der Modellbildung auf hoher Ebene, es wird die Architektur des Modells festgelegt. Die Hauptkomponenten des Systems werden identifiziert und in das Modell abgebildet. Auch die Interaktionen der Komponenten untereinander und die Schnittstellen der Komponenten sowie die grundlegende hierarchische Strukturierung des Modells werden definiert. Meist wird in diesem Stadium des Modells die Zeit noch nicht berücksichtigt, d. h. es wird zunächst ein reaktives System modelliert. Das Arbeitsergebnis dieser Phase ist ein abstraktes CTA-Modell, welches alle Architektur- und Entwurfs-Entscheidungen beinhaltet.

Von zentraler Bedeutung ist die **Verfeinerungsphase**. In dieser Phase werden zwei unterschiedliche Entwicklungsaktivitäten ausgeführt: Zum einen wird das CTA-Modell aus der Entwurfsphase oder aus einer vorherigen Iteration der Verfeinerungsphase weiterentwickelt. Es werden alle noch fehlenden Details eingefügt und bisher noch unwichtige Komponenten implementiert. Falls bisher noch nicht geschehen, wird in dieser Phase das Zeitverhalten der einzelnen Komponenten modelliert. Zum anderen wird die CTA-Spezifikation, eine formale Definition aller zu überprüfenden Eigenschaften des Modells, erstellt. In dieser Version der Anforderungsspezifikation wird Bezug auf die einzelnen Komponenten des CTA-Modells genommen, wobei beliebig tief auf die Details des Modells eingegangen werden kann und auch Komponenten der Steuerung adressiert werden können. Am Ende einer Iteration dieser Phase steht als Arbeitsergebnis ein Paar von CTA-Spezifikation und CTA-Modell zur Verfügung. Es ist also grundsätzlich darauf zu achten, daß nach dieser Phase Spezifikation und Modell zusammen passen. Je nach Modellgröße und Grobheit der Verfeinerungsschritte sind mehrere Iterationen der Verfeinerungsphase notwendig.

Die eigentliche formale Verifikation erfolgt in der **Verifikationsphase**. Das vollautomatische Analyse-Werkzeug überprüft, ob das CTA-Modell alle in der CTA-Spezifikation aufgeführten Eigenschaften erfüllt. Dabei entsteht als Dokument eine Sammlung von Analyseresultaten, die von beliebigen imperativen Analyseanweisungen berechnet werden und daher von der Aussage ”Eigenschaft erfüllt” bzw. ”Eigenschaft verletzt” bis hin zu detaillierten Informationen über die berechneten Konfigurationen reichen können.

Ein wesentliches Problem bei der auf formale Verifikation gestützten Softwareentwicklung besteht darin, daß die Verifikation des Modells wenig über das letztendlich in das eingebettete System eingebaute Steuerungsprogramm aussagt, da bei der Implementierung des Steuerungsprogramms Fehler eingebracht werden können. Diese manuell programmierte Steuerung muß grundsätzlich ausführlich mit herkömmlichen Methoden getestet werden. In der Phase der **Codeerzeugung** wird für den Teil des Modells, der die Steuerung modelliert, ein ausführbares Programm generiert. Ist der Generator fehlerfrei,

der aus dem Steuerungsmodell das Steuerungsprogramm erzeugt, dann kann angenommen werden, daß das generierte Programm auch korrekt bzgl. der Spezifikation ist, falls das Modell korrekt bzgl. der Spezifikation war. Der Einsatz einer solchen Generierung von ausführbarem Code ist in zweierlei Hinsicht lohnenswert. Einerseits benötigt das sorgfältige Sicherstellen der Korrektheit des Übersetzers weniger Zeit als das Sicherstellen der Korrektheit des Steuerungsprogrammes, da dieses viel größer ist als der Codegenerator selbst. Andererseits genügt es, die Korrektheit des Generators einmal zu zeigen, während die Korrektheit des manuell erzeugten Steuerungsprogrammes bei jeder neuen Steuerung nachgewiesen werden muß.

6.2 Von der Idee über das Modell zum Programm

In diesem Abschnitt werden einige in diesem Projekt angenommene Leitgedanken zur Modell-Bildung und zu den Beziehungen der verschiedenen Entwicklungsstadien des Modells zueinander wiedergegeben. Wie im Abschnitt 6.1 unter dem Gesichtspunkt der Prozeß- und Dokumentenabfolge bereits angedeutet, arbeitet der Entwickler mit Beschreibungen des Steuerungsprogrammes auf vielen verschiedenen Ebenen. Abbildung 6.2 auf der nächsten Seite zeigt eine Abfolge von Systembeschreibungen der wichtigsten vier Ebenen.

Aus den Überlegungen zu den Aufgaben des Systems entsteht eine **Anforderungsdefinition**. Diese wird meist von Fachexperten, d. h. von Leuten aus dem Anwendungsbereich des Systems (Kunden), in natürlichsprachlicher Form verfaßt und von den Entwicklern häufig als ”schwammig” bewertet.

In Analysen entwickelt das Softwareentwicklungsteam ein klares Bild davon, was genau vom fertigen System erwartet wird. Als Ergebnis dieser Phase entsteht eine formal verfaßte und damit präzise **Spezifikation** der Eigenschaften des Systems (formalisierte Anforderungsbeschreibung). Dabei wird häufig eine Temporal-Logik verwendet, wobei sich die Formeln nur auf die physische Umgebung der Steuerung beziehen sollten, da die Steuerung selbst noch nicht, auch noch nicht als Modell, existiert. Dieser Prozeß der Formalisierung findet, abhängig davon, wie abstrakt er gehalten wird, entweder in der Analysephase oder in der Verfeinerungsphase statt. Hier wird bereits der erste Vorteil der Anwendung formaler Methoden deutlich: durch das präzise Formulieren entdeckt der Entwickler Lücken in seinem Verständnis oder zu Widersprüchen führende Interpretationsfehler, die bereits jetzt durch Rückfragen an die Fachexperten geklärt werden können und sonst erst in einer späteren Phase explizit geworden wären.

Im nächsten Schritt wird das eigentliche formale **Modell** entwickelt. Das System-Modell enthält ein Teil-Modell zur Definition des Aufbaus und des Verhaltens der Steuerung (Steuerungsmodell) sowie ein Umgebungsmodell, welches alle Annahmen über die physische Umgebung der Steuerung festgelegt. Zur Notation des Modells werden z. B. zustandsbasierte Formalismen wie Petrinetze oder Automaten verwendet (hier natürlich Timed Automata). Die neue, weniger abstrakte Beschreibung des Systems stellt eine Verfeinerung der Spezifikation des Systems dar. Die Korrektheit des Modells bezüglich der

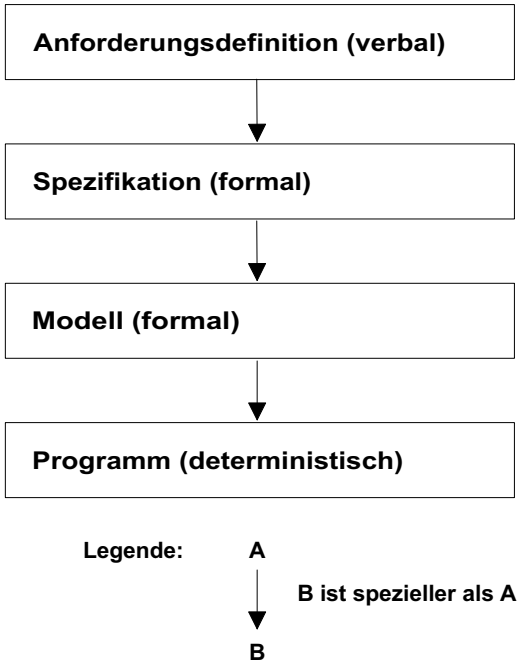


Abbildung 6.2: Systembeschreibungen auf verschiedenen Ebenen.

Spezifikation kann durch Model-Checking nachgewiesen werden, d. h. es werden alle in der Spezifikation festgelegten Eigenschaften am Modell geprüft.

Da ein genügend detailliertes Modell meist nicht in einem Schritt entwickelt werden kann, sind evtl. weitere Verfeinerungsschritte notwendig. Dabei wird gewöhnlich eine der beiden folgenden Vorgehensweisen gewählt:

- Es werden in einem iterativen Prozeß weitere schrittweise Verfeinerungen des Modells vorgenommen. Ist der gewünschte Detaillierungsgrad erreicht, werden mittels Erreichbarkeitsanalyse alle geforderten Eigenschaften nachgewiesen. Der offensichtliche Nachteil dieser Strategie ist das schnell unpraktikabel groß werdende Modell, was zur Folge hat, daß bestimmte Analysen nicht mehr möglich sind.

- Sobald im Modell genügend Details vorhanden sind, um eine Eigenschaft zu verifizieren, wird diese Eigenschaft des Modells bewiesen. Bei allen weiteren Verfeinerungsschritten wird nach Vollendung eines bestimmten Zwischen- bzw. des Endstadiums bewiesen, daß die neue, detailliertere Version des Modells (oder eines Teilmodells für eine Komponente) eine Verfeinerung des korrespondierenden Modells der abstrakteren Version ist. Diese Art der schrittweisen Verfeinerung wird *eigenschaftserhaltende Verfeinerungstransformation* genannt.

Um letztendlich eine ausführbare Steuerung zu erhalten, muß im letzten Schritt aus dem Teil des Systemmodells, welches das Steuerungsprogramm modelliert, ein **Programm** erzeugt werden. Dabei muß das im Modell der Steuerung modellierte und als

korrekt bewiesene Verhalten der Steuerung als ”Software” in einer Programmiersprache implementiert werden. Dabei kommt es nochmals zu einer Verfeinerung durch viele im Modell nicht berücksichtigte technische Details sowie evtl. durch das Auflösen konzeptueller Abstraktionen (z. B. Nicht-Determinismus). Ein zuverlässiger und in der Softwaretechnik weitgehend angestrebter Ansatz ist hier die automatische Codeerzeugung. Dabei wird die manuelle Programmierarbeit der Entwickler auf ein Minimum (z. B. Schnittstellen zu Funktionsaufrufen der Hardwaresteuerung) reduziert.

6.3 Beschreibung der Fertigungsanlage und ihrer Anforderungen

Für Zwecke der Lehre und Forschung wurde am Lehrstuhl Software-Systemtechnik der BTU Cottbus das Fischertechnik-Modell einer Produktionsanlage errichtet (siehe Abbildung 6.3). Im folgenden werden einige für die Modell-Bildung erforderliche Informationen zu diesem Beispiel-System angegeben.

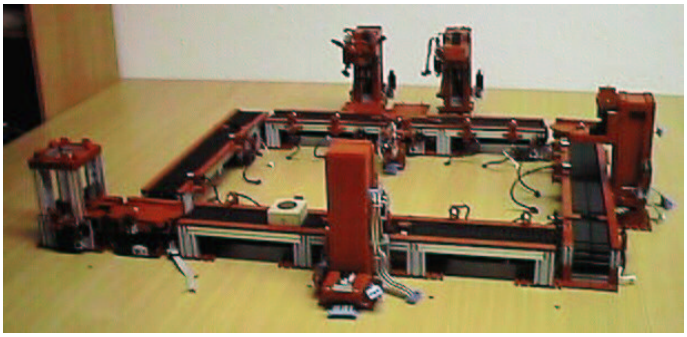


Abbildung 6.3: Fischertechnik-Modell einer Fertigungsanlage.

6.3.1 Überblick

Der schematische Aufbau dieser Anlage wird in Abbildung 6.4 auf der nächsten Seite dargestellt. Als Grundkomponente für den Transport der Werkstücke innerhalb der Anlage stehen Transportbänder zur Verfügung, die ein Werkstück ringförmig an allen Bearbeitungsstationen vorbeiführen können. Soll ein Werkstück bearbeitet werden, muß es vor der entsprechenden Bearbeitungsmaschine positioniert werden.

Für die Bearbeitung mehrerer Werkstücke steht ein Werkstück-Lager bereit, aus dem bei Bedarf neue Werkstücke in den ”Produktionsprozeß” geholt werden können. Das Lager kann die Anlage auch mit mehreren Werkstücken gleichzeitig bestücken, die mit den Bändern beliebig bewegt werden können. Es gibt keine vorgeschriebene Transportrichtung; die Werkstücke können prinzipiell im oder entgegen dem Uhrzeigersinn bewegt werden. Die Bearbeitung der Werkstücke wird von mehreren Maschinen durchgeführt (Presse, Bohrmaschine, Fräse, Multiwerkzeug-Maschine).

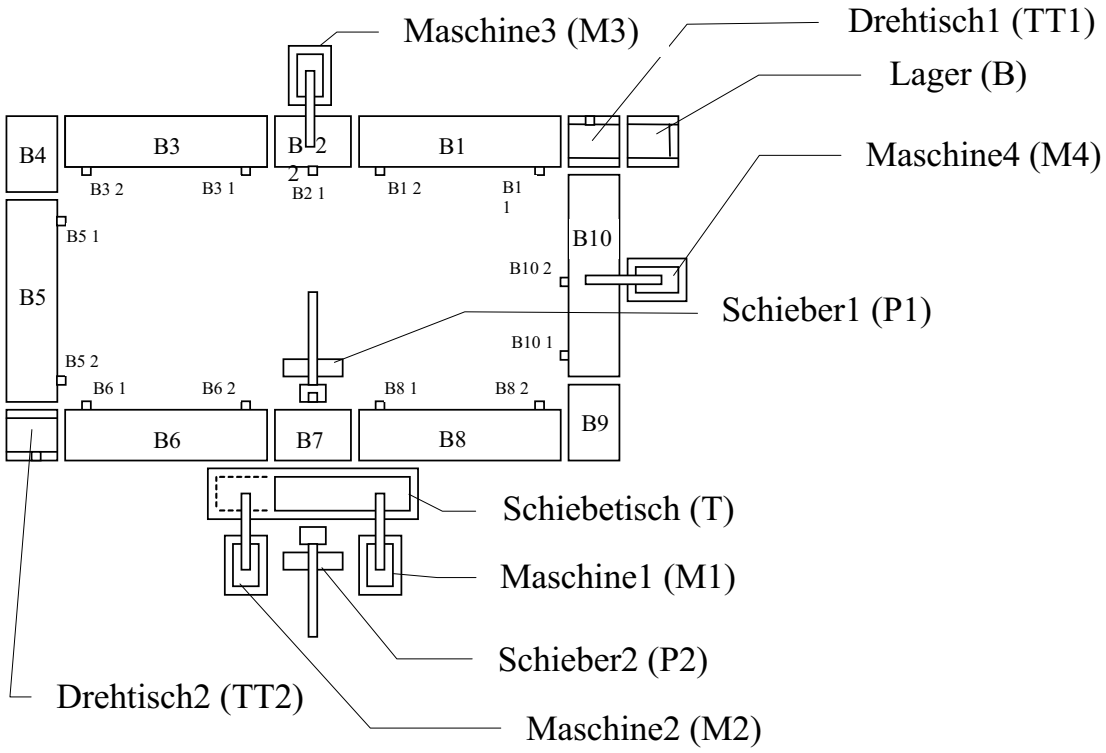


Abbildung 6.4: Schematischer Aufbau der Fertigungsanlage.

Die Sensoren sind über eine entsprechende Pegelanpassung direkt an die Parallelports eines PCs (über I/O-Karten) angeschlossen. Die Ansteuerung der Motoren erfolgt über externe Schaltungen für die Stromverstärkung und Laufrichtungsumkehr (Umpolung). Die Anlage umfaßt 28 Motoren und 44 Sensoren.

6.3.2 Komponenten

Förderbänder. Ein Förderband besteht aus dem eigentlichen Förderband, welches durch einen Motor bewegt werden kann, und aus Sensoren, mit denen die Position eines sich auf dem Band befindlichen Werkstücks festgestellt werden kann. Nach den Aufgaben des Förderbandes und der Sensoranordnung wird unterschieden in

- *Transportbänder* mit jeweils einem Sensor am Anfang und am Ende des Bandes, die es ermöglichen, dem übergebenden Anlagenteil mitzuteilen, daß das Werkstück übernommen worden ist,
- *Bearbeitungsbänder* mit einem Sensor in der Mitte des Bandes, der dazu dient, das Werkstück für die zu bedienende Maschine zu positionieren,
- *kurze Bänder* ohne Sensor für das Weitertransportieren oder das Auswerfen der Werkstücke an den Ecken der Anlage, an denen keine Drehtische vorhanden sind.

Drehtische. Ein Drehtisch ist eine drehbare Plattform, in das ein Förderband integriert ist. Die Plattform kann mit Hilfe eines Motors nach links oder rechts gedreht werden. Mit Hilfe zweier Sensoren kann festgestellt werden, in welcher Endposition sich die Plattform momentan befindet. Das Förderband kann mit Hilfe eines weiteren Motors bewegt werden. Um feststellen zu können, ob sich ein Werkstück auf dem Förderband befindet, steht ein weiterer Sensor in der Mitte der Plattform zur Verfügung.

Bearbeitungsmaschinen. In der Anlage kommen Maschinen der folgenden Typen zum Einsatz: Bohrmaschine, Presse, Fräse. Allen Maschinen ist gemeinsam, daß sie einen Arbeitskopf haben, der mit Hilfe eines Motors gehoben und gesenkt werden kann. In welcher Endposition er sich befindet, kann durch zwei Sensoren der Maschine ermittelt werden. Ein Sensor ist aktiviert, wenn sich der Arbeitskopf in seiner Ausgangsposition (oben) befindet, und der andere, wenn sich der Kopf in seiner untersten Position befindet.

Schieber. Ein Schieber besteht aus einem ausfahrbarem Arm, der das Werkstück von sich wegschieben kann. Dieser Arm wird mit Hilfe eines Motors ausgefahren und eingezogen. Durch zwei Sensoren wird ermittelt, in welcher Position sich der Arm gerade befindet. Ein Sensor ist aktiviert, wenn sich der Arm in seiner Ausgangsposition (eingezogen) befindet, der andere, wenn der Arm vollständig ausgefahren ist. Ein weiterer Sensor am Ende des Armes kann feststellen, ob sich ein Werkstück vor dem Schieber befindet oder nicht.

Schiebetisch. Der Schiebetisch besteht aus einer Plattform, die nach links oder rechts bewegt werden kann. Für die Steuerung steht ein Motor zur Verfügung, der die Plattform nach links bzw. rechts bewegen kann. Zur Kontrolle der Position des Tisches befinden sich am rechten und linken Ende je ein Sensor.

Werkstück-Lager. Vor Beginn der Bearbeitung der Werkstücke durch die Fertigungsanlage befinden sich alle Werkstücke im Lager, wo sie übereinander eingestapelt sind. Durch eine Kette mit einem Mitnehmer kann ein Werkstück aus dem Lager heraus auf den Drehtisch 1 befördert werden.

6.3.3 Typischer Fertigungsablauf

Im folgenden wird ein möglicher, typischer Fertigungsablauf geschildert, bei dem die Werkstücke entgegen dem Uhrzeigersinn durch die Anlage transportiert werden:

- Das Werkstück (Rohteil) wird dem Lager entnommen.
- Sollte sich der (freie) Drehtisch 1 vor dem Band 10 befinden, wird er durch eine 90°-Drehung entgegen dem Uhrzeigersinn vor das Lager gedreht. (Das Lager und das Band 10 sind Konkurrenten bzgl. des Drehtisches 1.)
- Wenn der Drehtisch vor dem Lager steht und frei ist, übernimmt er das Werkstück aus dem Lager.
- Der Drehtisch 1 befördert das Werkstück auf das Band 1, falls dieses frei ist. (Falls der Drehtisch ein Teil aus dem Lager übernommen hat und auf das Band 1 warten muß, blockiert er auch das Band 10.)

- Das Band 1 befördert das Werkstück zum Band 2. Sollte das Band 2 belegt sein, wird am zweiten Sensor des Bandes gewartet.
- Das Band 2 befördert das Werkstück zur Bandmitte und hält dort an.
- Die Maschine 3 bearbeitet das Werkstück.
- Ist die Maschine 3 nach erfolgter Bearbeitung wieder in ihrer Ausgangsposition angekommen, wird das Werkstück vom Band 2 auf das Band 3 transportiert, sobald dieses frei ist.
- Das Band 3 befördert das Werkstück zum Band 4, sollte das Band 4 belegt sein, wird am zweiten Sensor von Band 3 gewartet,
- Das Band 4 befördert das Werkstück nach gleichem Prinzip über Band 5 zum Drehtisch 2. Sollte der Drehtisch 2 belegt sein oder nicht vor dem Band 5 stehen, wird am zweiten Sensor gewartet.
- Ist der Drehtisch leer und steht vor dem Band 6, wird er durch eine 90°-Drehung mit dem Uhrzeigersinn vor das Band 5 gedreht.
- Der Drehtisch 2 nimmt das Werkstück vom Band 5 auf, bewegt es bis zu seinem Bandsensor und vollführt danach eine 90°-Drehung entgegen dem Uhrzeigersinn.
- Wenn das Band 6 frei ist, lädt der Drehtisch das Werkstück dort ab.
- Das Band 6 befördert das Werkstück zum Band 7, sollte jedoch das Band 7 oder der Schiebetisch belegt sein, wird am zweiten Sensor von Band 6 gewartet.
- Sind Schiebetisch und Band 7 frei, befördert das Band 7 das Werkstück vor den Schieber 1 und hält dort an. (Band 7 und der Schiebetisch haben keinen eigenen Sensor zur Erfassung, ob sich ein Teil auf ihnen befindet. Dies kann nur durch den Sensoren von Schieber 1 oder durch den Zustand der Steuerung festgestellt werden.)
- Der Schieber 1 befördert das Werkstück auf den Schiebetisch, wenn dieser sich in der rechten Endlage befindet.
- Sobald der Schieber 1 wieder seine Ausgangsstellung erreicht hat, bewegt sich der Schiebetisch mit dem Werkstück nach links.
- Ist der Schiebetisch mit dem Werkstück am linken Endschalter angekommen, bearbeitet die Maschine 2 das Werkstück.
- Sobald die Maschine 2 wieder die Ausgangsstellung erreicht hat, bewegt sich der Schiebetisch mit dem Werkstück nach rechts.
- Ist der Schiebetisch mit dem Werkstück an der rechten Endlage angekommen, stößt der Schieber 2 das Werkstück wieder auf das Band 7 zurück.

- Wenn der Schieber 2 wieder seine Ausgangslage erreicht hat, bewegt sich der nunmehr leere Schiebetisch nach links.

- Wenn der Schiebetisch links angekommen ist, wird das Werkstück vom Schieber 1 wieder auf den Schiebetisch geschoben.

- Ist der Schieber 1 wieder in seiner Ausgangslage angekommen, bewegt sich der Schiebetisch mit dem Werkstück nach rechts.

- In der rechten Endstellung des Tisches bearbeitet die Maschine 1 das Werkstück.

- Sobald die Maschine 1 wieder die Ausgangsstellung erreicht hat, bewegt sich der Schiebetisch mit dem Werkstück nach links.

- Ist der Schiebetisch mit dem Werkstück an der linken Endlage angekommen, stößt der Schieber 2 das Werkstück wieder auf das Band 7.

- Wenn der Schieber 2 wieder seine Ausgangslage erreicht hat, bewegt sich der nunmehr leere Schiebetisch nach rechts, gleichzeitig bewegt das Band 7 das Werkstück auf das Band 8, falls dieses frei ist.

- Das Band 8 befördert das Werkstück zum Band 9. Sollte das Band 9 oder Band 10 belegt sein, wird am zweiten Sensor von Band 8 gewartet.

- Das Band 9 befördert das Werkstück zum Band 10.

- Das Band 10 befördert das Werkstück zum Drehtisch 1, sollte der Drehtisch 1 belegt sein oder vor dem Lager stehen, wird am zweiten Sensor gewartet. (Hier kann auch eine weitere Bearbeitung durch Maschine 4 erfolgen.)

- Sollte der Drehtisch 1 vor dem Lager stehen, wird er durch eine 90°-Drehung mit dem Uhrzeigersinn vor das Band 10 gedreht.

- Der Drehtisch 1 nimmt das Werkstück vom Band 10 auf, stellt die Aktivierung und die nachfolgende Deaktivierung des Bandsensors des Drehtisches fest und transportiert das Werkstück ab. Der Abtransport erfolgt durch Auswerfen gegenüber Band 10.

Während die Bänder 1 bis 6, die Bänder 8 bis 10, der Drehtisch 2 und (mit Einschränkungen) der Drehtisch 1 völlig unabhängig arbeiten, ist die Behandlung der Gruppe Schieber 1 und Schieber 2, Schiebetisch, Machine 1 und Maschine 2 und das Band 10 komplex. Hier liegt eine streng sequentielle Abarbeitung vor.

6.4 Modellierung

Die Entwicklung des Modells für dieses größere, realistische System erfolgte entsprechend dem in Abschnitt 6.1 auf Seite 186 vorgeschlagenen Prozeßmodell, indem nacheinander auf mehreren verschiedenen Abstraktionsebenen modelliert und jeweils eine Verfeinerung des vorhergehenden Modells entwickelt wurde. Im folgenden werden der Aufbau des Modells erläutert und einzelne Teil-Modelle genauer beschrieben. Dabei spielt der modulare Aufbau durch das Verwenden verschiedener Hierarchieebenen eine besondere Rolle.

Das im folgenden beschriebene Modell beinhaltet den Kreislauf eines Werkstücks durch die Fertigungsanlage, in dem das Werkstück aus dem Lager kommt, alle Transportbänder und Drehtische, sowie zwei Maschinen passieren muß.

Um die Architektur des Modells zu erproben, wurde in der ersten Phase der Modellierung ein Modell für eine fiktive, kleinere Fertigungsanlage mit vier Transportbändern und zwei Drehtischen, jedoch ohne Maschinen, entwickelt. Nachdem auf dieser Weise die Komponentenstruktur und die Kommunikationsmechanismen erprobt worden sind, wurde das Modell der Fertigungsanlage durch das Hinzunehmen der restlichen Bänder und Maschinen schrittweise vervollständigt. Dabei wurde nach den gleichen Entwurfsprinzipien gearbeitet, und es wurden methodische Experimente durchgeführt. Z. B. wurden bei der eigenschaftserhaltenden Verfeinerungstransformation einzelne Transformationsschritte mittels Verfeinerungsanalyse verifiziert. In der letzten Phase der Modellierung, nachdem alle Komponenten und die Kommunikation zwischen ihnen fertig modelliert waren, wurde damit begonnen, die Realzeit-Anforderungen einzubringen.

Das Modell besteht aus den folgenden drei Hauptkomponenten, die bereits auf der obersten Abstraktionsschicht unterschieden werden:

Physische Umgebung: Die Hardware-Komponenten der Anlage stellen die physische Umgebung dar, die von der Steuerung beeinflußt werden soll. Das erwartete Verhalten jeder Hardware-Komponente der Fertigungsanlage wird im physischen Modell festgehalten.

Steuerung: Das Verhalten der Steuerung wird in einem separaten Modellteil modelliert. In diesem Modell existiert für bestimmte Einheiten der physischen Umgebung je eine Beschreibung des angestrebten Steuerungsverhaltens für die Komponenten. Das Steuerungsmodell bildet im späten Entwicklungsprozeß die Grundlage für das ausführbare Steuerungsprogramm.

Werkstück: In einem dritten Modellteil wird die Position des zu bearbeitenden und in der Anlage umlaufenden Werkstücks sowie die Bearbeitungszeit repräsentiert, und es wird der Steuerungsablauf programmiert, d. h. in welcher Weise das Werkstück durch die Anlage läuft und von welchen Maschinen es bearbeitet wird.

Diese drei Teilmodelle werden in den folgenden Abschnitten ausführlich erläutert.

6.4.1 Modell der physischen Umgebung

Im Modell der physischen Umgebung werden die vorhandenen und zu steuernden Hardware-Komponenten modelliert. Es definiert die Kommunikationsschnittstelle zur Ansteuerung durch das Steuerungsmodell und legt das Verhalten der Komponenten fest, d. h. alle Annahmen über das Verhalten der Umgebung der Steuerung.

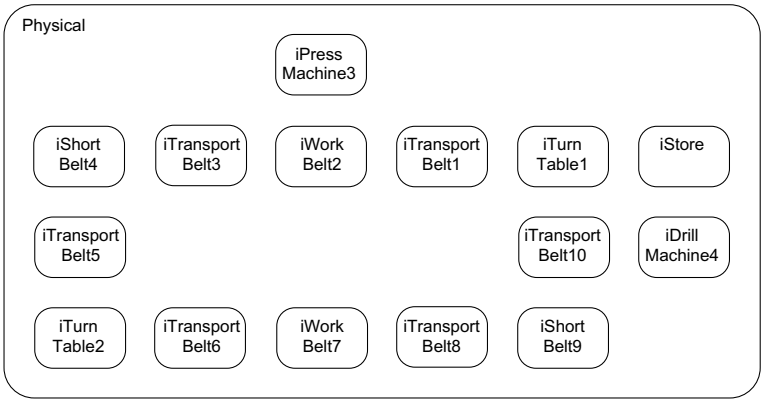


Abbildung 6.5: Physische Struktur der Fertigungsanlage.

In diesem Abschnitt werden einige ausgewählte Teile des Modells der Fertigungsanlage wiedergegeben. Dabei steht die Beschreibung der grundlegenden Strukturen und Modellierungskonzepte im Vordergrund, weniger die Erklärung der Details. Abbildung 6.5 stellt die im weiteren Verlauf der Arbeit berücksichtigten Komponenten der Fertigungsanlage dar (die graphische Notation für Module wird in der Abbildung 4.5 auf Seite 119 erklärt). Das Modul Physical besteht aus den im folgenden aufgeführten Unterkomponenten.

Bestandteile:

- 6 Transportbänder mit zwei Sensoren (Modul TransportBelt).
- 2 Arbeitsbänder mit einem Sensor (Modul WorkBelt).
- 2 kurze Bänder ohne Sensor (Modul ShortBelt).
- 2 Drehtische (Modul TurnTable).
- 2 Bearbeitungsmaschinen (Module DrillMachine und PressMachine).
- 1 Lager für Werkstücke (Module Store).

Bei der Entwicklung des Modells der physischen Umgebung wird damit begonnen, die Systemstruktur der Fertigungsanlage zu erfassen und ins Modell zu übertragen. Der physischen Aufbaustruktur entsprechend wird eine Hierarchie von Komponenten und Unterkomponenten (bzw. Systeme und Subsysteme) entwickelt (siehe Abbildung 6.6). Für jede

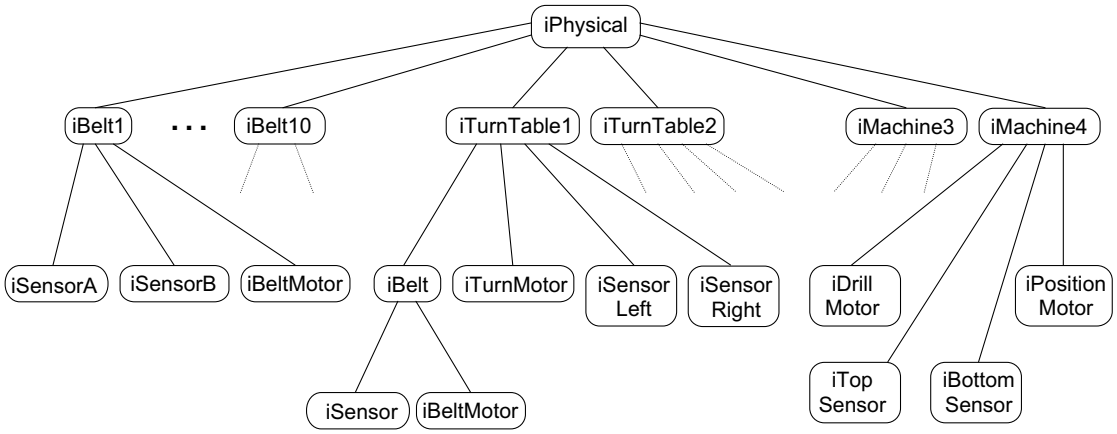


Abbildung 6.6: Baum der Instanzstruktur der Module des physischen Modells.

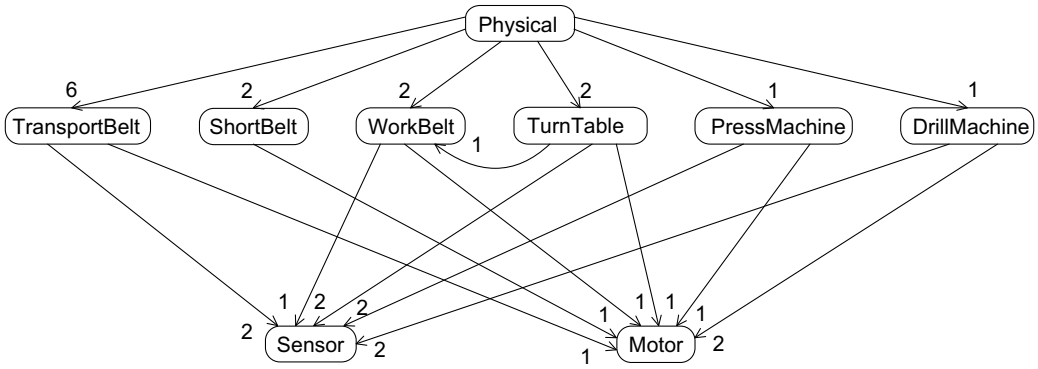


Abbildung 6.7: Enthaltenseinsbeziehungen zwischen den Modulen im physischen Modell.

Komponente wird ein CTA-Modul entwickelt, welches Instanzen der den Unterkomponenten zugeordneten CTA-Module enthält. Das dabei entstehende hierarchische Geflecht aus Enthaltenseinsbeziehungen wird in der Abbildung 6.7 gezeigt. Diese Darstellung stellt den hohen Wiederverwendungsgrad der Modelle auf den niedrigeren Ebenen heraus. So enthält zum Beispiel ein Transportband zwei Sensor-Instanzen und eine Motor-Instanz.

Danach werden die bei der Ansteuerung der Fertigungsanlage real vorkommenden Ereignisse auf Signale (bzw. Synchronisationsmarken) abgebildet. Die Signale auf niedrigster Ebene sind gegeben durch die Ereignisse an der Hardware der Anlage, also den Sensoren und Motoren. Jedem Flankenwechsel der elektrischen Pegel an den Sensoren und Motoren ist ein Signal im physischen Modell zugeordnet. Entsprechend der Rolle der Kommunikationspartner wird hier die ”Richtung” der Signale unterschieden. Motorsignale werden von der Steuerung an die Hardware ”gesendet”, während Sensorsignale von der Hardware an die Steuerung ”gesendet” werden. Diese Signalrichtungen lassen sich auf die CTA-Module des Modells wie folgt übertragen:

- In ein Modul eingehende Signale werden als Eingabe-Signale deklariert; ein Modul muß auf ein solches Signal jederzeit reagieren können. Die Motor-Signale von der Steuerung sind für das physische Modell Eingabe-Signale.

- Von einem Modul ausgehende Signale werden als Ausgabe-Signale deklariert; das Auftreten eines solchen Signals darf nur von diesem Modul bestimmt werden. Die Sensor-Signale sind für das physische Modell Ausgabe-Signale.

Das Verhalten der Komponenten, also die zeitabhängigen Reaktionen auf die verschiedenen Signale und das Erzeugen neuer Signale, wird durch Timed Automata beschrieben. Die Zustände des Automaten entsprechen jeweils der aktuellen Situation der Hardware-Komponente und Zustandsübergänge sind meist gegeben durch den Wechsel der inneren Situation einer Komponente aufgrund eines Ereignisses, d. h. des Auftretens eines Signals oder des Ablaufens einer bestimmten Zeit.

Im folgenden werden exemplarisch für einige wichtige Komponenten des physischen Modells die Bestandteile aufgezählt, eine die enthaltenen Unterkomponenten und Schnittstellensignale verdeutlichende Abbildung gegeben und eine Beschreibung der Schnittstellenelemente sowie des Verhaltens gegeben.

6.4.1.1 Transportband

Die zentrale Komponente für den Transport der Werkstücke bildet das *Transportband*, dessen Aufbau in Abbildung 6.8 dargestellt ist (siehe Abbildung 4.5 auf Seite 119 für die Notation).

Bestandteile:

- 2 Bandsensoren (Modul Sensor).

- 1 Antriebsmotor (Modul Motor).

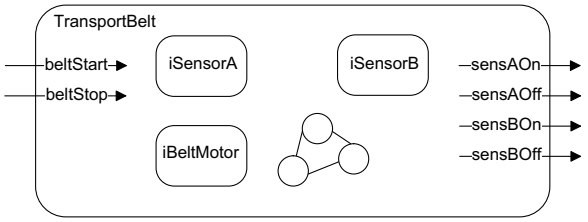


Abbildung 6.8: Modul für das Transportband.

Eingabe-Signale: Das Modul erwartet von der Steuerung das Signal beltStart zum Starten und beltStop zum Stoppen des Bandes.

Ausgabe-Signale: An die Steuerung werden von den zwei Sensor-Instanzen vier Signale weitergegeben. sensAOn signalisiert, daß der erste Sensor ein Werkstück erkannt hat, sensAOff, daß sich kein Werkstück im Bereich des ersten Sensors befindet. Analog existieren für den zweiten Sensor die Signale sensBOn und sensBOff.

Lokale Signale und Uhren: Für die Kommunikation innerhalb des Moduls existieren zwei lokale Signale. sensAPieceArrives teilt dem ersten Sensor mit, daß sich ein Werkstück nähert und der Sensor potentiell seinen Zustand ändern kann. sensBPieceArrives erfüllt die gleiche Funktion für den zweiten Sensor. Die Uhr c dient der Steuerung des Zeitverhaltens.

Konstanten: Das Modul für das Transportband hat zwei Paare von Parametern: minTimeForArriving für die minimale Zeit und maxTimeForArriving für die maximale Zeit, die das Werkstück benötigt, um bis zum ersten Sensor zu gelangen. minTimeBetween und maxTimeBetween sind die entsprechenden Zeitschranken für die Zeit, in der das Werkstück zwischen dem ersten und dem zweiten Sensor befördert wird.

Verhalten: Der Initialzustand des Moduls ist ein nicht belegtes, stehendes Band, d. h. die beiden Sensoren und der Motor befinden sich im inaktiven Zustand. Durch das Signal beltStart werden der Motor des Transportbandes gestartet und eine interne Uhr zurückgesetzt, die die Zeitspanne zwischen Anschalten des Motors und dem Eintreffen des Werkstücks am Sensor mißt. Nach Ablauf einer durch zwei Konstanten eingegrenzten Zeitspanne wird dem ersten Sensor die mögliche Ankunft des Werkstücks signalisiert und der Sensor wechselt innerhalb des Toleranzbereichs in seinen aktiven Zustand.

Nachdem der erste Sensor wieder in seinen inaktiven Zustand zurückgekehrt ist (falls der Motor entsprechend eingeschaltet ist), wird die Uhr zurückgesetzt, um die Zeitspanne für die Werkstückbewegung zwischen den Sensoren zu messen. Wie beim ersten Sensor wird auch dem zweiten Sensor die mögliche Ankunft eines Werkstücks signalisiert und dieser kann sich innerhalb eines Toleranzbereichs aktivieren.

Nachdem sich der zweite Sensor deaktiviert hat, d. h. das Werkstück seinen Bereich verlassen hat, vergeht noch die Zeit bis zum Erreichen des ersten Sensors des nächsten Bandes. Nach Eintreffen des Stop-Signals wird der Motor angehalten und das Transportband kehrt wieder in seinen Initialzustand zurück.

6.4.1.2 Drehtisch

Das Modell des Drehtisches ist interessant, weil es einerseits aus einem Motor und zwei Sensoren für die Drehbewegung besteht, andererseits einen Motor und einen Sensor für den Transport über das integrierte Förderband enthält. Dieser Sachverhalt kann komfortabel modelliert werden, indem das den Drehtisch modellierende Modul eine Instanz eines Bearbeitungsbandes enthält (siehe Abbildung 6.9 auf der nächsten Seite).

Bestandteile:

- 1 Bearbeitungsband mit einem Sensor zum Transport des Werkstücks (Modul WorkBelt). Der Sensor dient dem Anhalten des Bandes vor dem Drehen der Plattform, wenn ein Werkstück in der Mitte des Bandes angekommen ist.
- 1 Motor zum Drehen des Drehtisches in zwei Richtungen (Modul Motor).
- 2 Positionssensoren zur Begrenzung der Drehbewegung des Drehtisches (Modul Sensor). Die Positionssensoren dienen der Steuerung als Signal für das Beenden der Drehbewegung der Plattform (Anschlag-Sensoren).

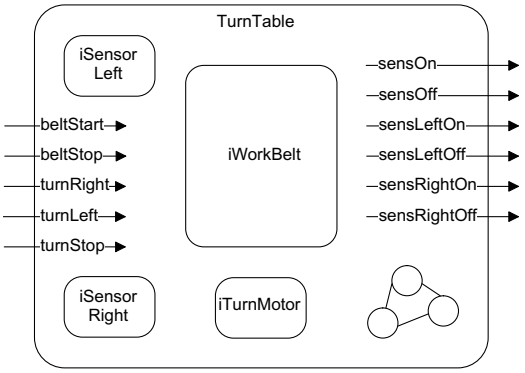


Abbildung 6.9: Drehtisch-Modul.

Eingabe-Signale: Das Signal beltStart dient dem Starten des Bandes und beltStop dem Anhalten. turnStartLeft startet die Drehbewegung der Plattform nach links, turnStartRight nach rechts. turnStop stoppt jede Drehbewegung.

Ausgabe-Signale: sensOn gibt an, daß der Bandsensor ein Werkstück erkannt hat. sensOff signalisiert, daß ein Werkstück den Bereich des Bandsensors verlassen hat. sensLeftOn teilt mit, daß der Drehtisch den linken Anschlagpunkt erreicht hat und sensLeftOff, daß der Bereich des Sensors verlassen worden ist. Analoges gilt für die Sensoren sensRightOn und sensRightOff.

Lokale Signale und Uhren: Das Signal sensRightTTArrives teilt dem Sensor für die rechte End-Position die Bewegung und potentielle Ankunft der Plattform am rechten Anschlag mit. sensLeftTTArrives signalisiert diesen Vorgang für den linken Sensor. Die Uhr c dient wieder der Steuerung des Zeitverhaltens.

Konstanten: Die minimale Zeit zum Drehen der Plattform wird durch die Konstante minTurnTime festgelegt, die maximale Zeit durch maxTurnTime. Die Konstanten für die Bandbewegung sind durch die Instanz iWorkBelt gegeben.

Verhalten: Die initiale Position des Drehtisches ist die linke Endposition mit einem angehaltenen Band, auf dem kein Werkstück liegt. Beim Eingabe-Signal zum Starten der Drehbewegung nach rechts wird die Uhr zurückgesetzt. Nachdem eine durch die Konstanten eingegrenzte Zeit für die Drehbewegung zwischen linker und rechter End-Position vergangen ist, wird dem rechten Sensor signalisiert, das der Drehtisch sich seiner Position nähert. Sobald die rechte End-Position erreicht ist, wartet der Drehtisch auf das Eingabe-Signal zum Zurückdrehen in die Ausgangsposition. Die Drehung nach links in die Ausgangsposition erfolgt analog.

6.4.1.3 Bohrmaschine

Da sich die verschiedenen Maschinen konzeptuell nur wenig unterscheiden, wird hier exemplarisch das Modell für eine Bohrmaschine vorgestellt (siehe Abbildung 6.10 auf der nächsten Seite).

Bestandteile:

- 1 Motor zum Drehen der Bohrspindel (Modul Motor).
- 1 Motor zum Auf- und Abbewegen der Bohrspindel (Modul Motor).
- 2 Sensoren für oberes und unteres Ende der Schiene, auf der die Spindel auf- und abbewegt wird (Modul Sensor). Die Sensoren dienen der Steuerung zur Kontrolle der Auf- und Abwärtsbewegung der Bohrspindel (Erreichen der Anschlagpunkte).

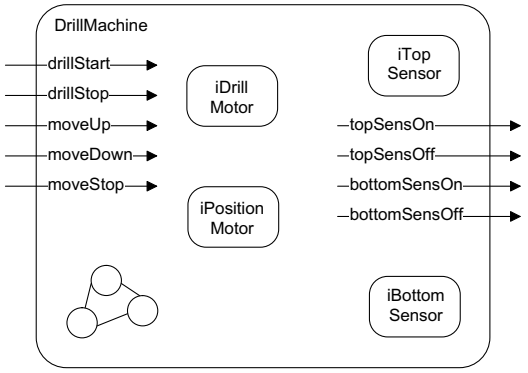


Abbildung 6.10: Bohrmaschinen-Modul (Maschine 4).

Eingabe-Signale: moveUp bewegt die Spindel nach oben. moveDown bewegt die Spindel nach unten. moveStop beendet die Auf- bzw. Ab-Bewegung der Bohrspindel. drillStart versetzt die Bohrspindel mit Bohrer in eine Drehbewegung und drillStop beendet diese Drehbewegung.

Ausgabe-Signale: topSensOn signalisiert das Erreichen der oberen Anschlagposition, und topSensOff signalisiert das Verlassen der oberen Anschlagposition. bottomSensOn und bottomSensOff sind die entsprechenden Signale für den unteren Anschlagsensor.

Lokale Signale und Uhren: bottomSensArrive signalisiert dem unteren Sensor, daß sich die Spindel nähert, topSensArrive dem oberen. Die Uhr c mißt die Zeit bei der Auf- und Abwärtsbewegung.

Konstanten: minTime legt die minimale Zeit und maxTime die maximale Zeit für die Positionierungsbewegung der Maschine fest.

Verhalten: Die Initialposition der Bohrspindel ist die obere Anschlag-Position. Die Eingabe-Signale zum Drehen des Bohrers werden vom Maschinenmodul direkt an das enthaltene Modul Motor weitergegeben. Der Automat im Maschinenmodul beobachtet die Signale zum Bewegen der Spindel. Ein Bewegen der Spindel nach oben über die Position des oberen Anschlagsensors hinaus führt zum Wechsel in einen Fehlerzustand (*PositionError*) des Automaten, d. h. eine fehlerhafte Ansteuerung wird protokolliert.

Sobald beim Senken der Spindel der obere Sensor deaktiviert wird, wird die Uhr der Maschine zurückgesetzt und es vergeht eine Zeitspanne entsprechend der Konstanten. Dem unteren Sensor wird die potentielle Ankunft der Spindel signalisiert. Sobald sich der untere Sensor aktiviert, wird die Uhr wieder zurückgesetzt.

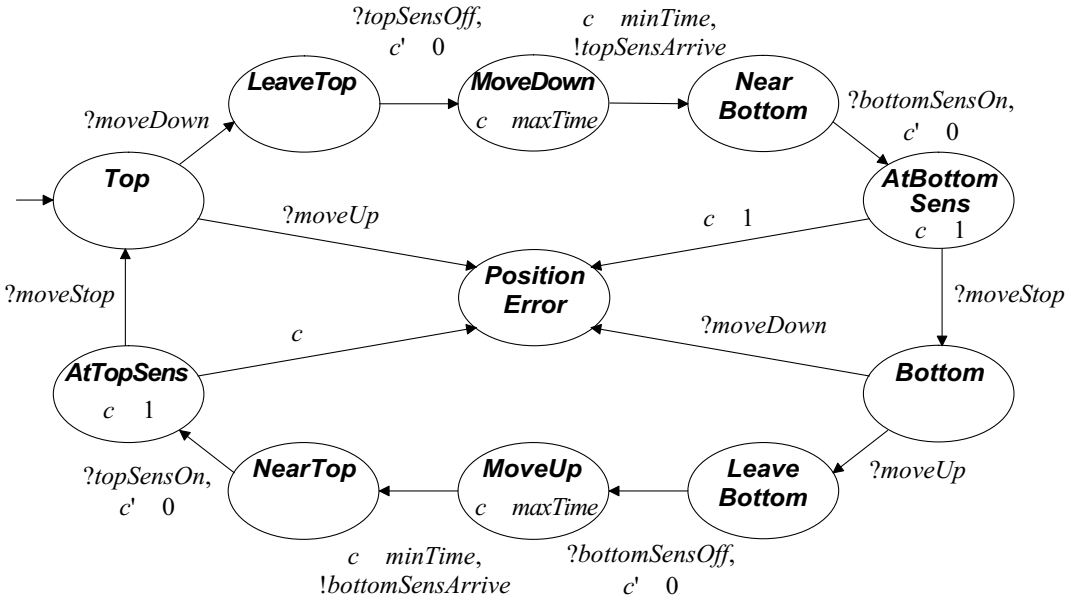


Abbildung 6.11: Automat für das Verhalten der Bohrmaschine.

Tritt das Eingabe-Signal zum Anhalten der Abwärtsbewegung zu spät ein, so fährt die Bohrspindel über den unteren Anschlagsensor hinaus und der Automat wechselt in einen Fehlerzustand (*PositionError*). Die Aufwärtsbewegung der Spindel erfolgt analog zur Abwärtsbewegung. Der Ablauf zur Registrierung der Positionierungsfehler wurde in dem in Abbildung 6.11 dargestellten Automaten modelliert.

6.4.1.4 Sensor

Dieses Modul (siehe Abbildung 6.12 auf der nächsten Seite) modelliert einen Sensor mit einer gewissen Schaltverzögerung beim Wechseln der Schaltzustände (*On*, *Off*) über jeweils einen Zwischenzustand (*PossibleToGoOn*, *PossibleToGoOff*), in dem einige Zeit vergeht. Dadurch werden Ungenauigkeiten beim Erkennen eines Werkstücks in das Umgebungsmodell integriert.

Eingabe-Signale: arrive signalisiert, daß sich ein Werkstück dem Bereich des Sensors nähert. Dieses Signal wird von dem die Sensorinstanz enthaltenden Modul an das Sensor-Modul gesendet (physisches Modell). Die Eingabe-Signale start und stop werden beobachtet, um die Information zu erhalten, ob sich das zu beobachtende Objekt (z. B. Werkstück oder Bohrspindel) bewegt oder nicht. Dies ist notwendig, da sich der Sensorzustand nur bei laufendem Motor verändern darf.

Ausgabe-Signale: Die beiden Ausgabe-Signale sensOn und sensOff geben dem übergeordneten Modul einen Zustandswechsel an.

Lokale Uhren: c ist eine Uhr, die die Zeitverzögerung beim Zustandswechsel des Sensors mißt.

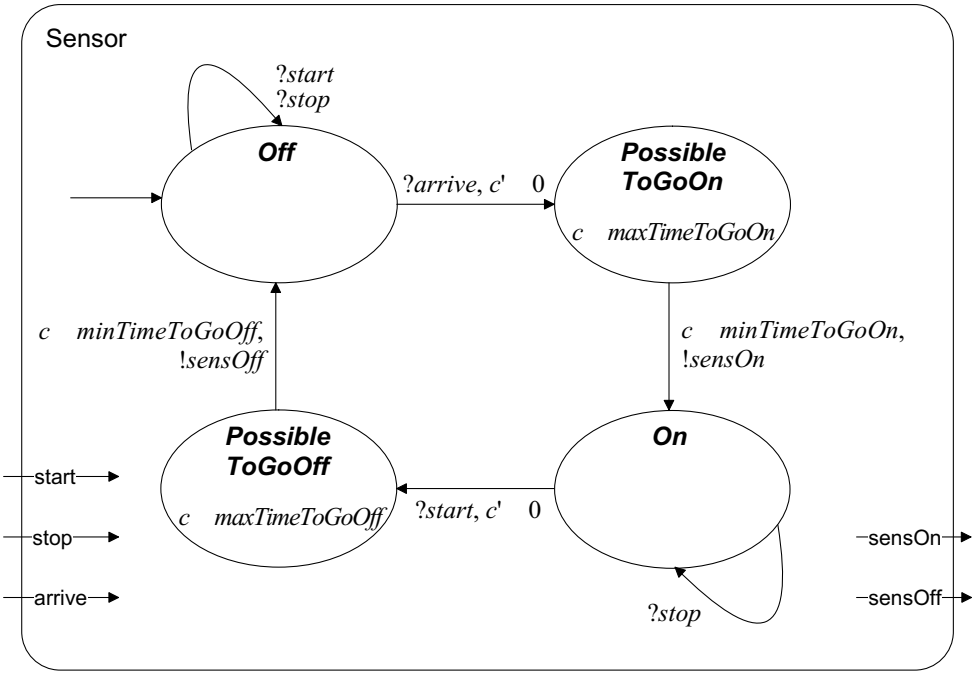


Abbildung 6.12: Sensor-Modul.

Konstanten: minTimeToGoOn legt die minimale Zeit zum Wechsel vom Zustand Off in den Zustand On fest, und maxTimeToGoOn legt die zugehörige maximale Zeit für diesen Wechsel fest. Entsprechend bestimmen die Konstanten minTimeToGoOff und maxTimeToGoOff die Zeitschranken für den Übergang vom Zustand *On* in den Zustand *Off*.

6.4.1.5 Motor

Das in der Abbildung 6.13 auf der nächsten Seite dargestellte Modul modelliert das Verhalten eines Motors. Der Motor wird über drei Signale für Linkslauf, Rechtslauf und Anhalten gesteuert. Das Verhalten des Motors wird in einem Automaten mit drei Kontrollzuständen und einem zusätzlichen Fehlerzustand *Damaged* modelliert. Der Fehlerzustand wird dann betreten, wenn der Motor aus einem Linkslauf direkt auf Rechtslauf umgeschaltet wird (oder umgekehrt), da in dieser Situation die Motormechanik selbst oder die Steuer-Elektronik für den Motor beschädigt werden kann.

Eingabe-Signale: Durch das Signal left wird der Motor angeschaltet für eine Linksdrehung, das Signal right versetzt den Motor in eine Rechtsdrehung und das Signal stop stoppt die momentane Drehbewegung.

6.4.2 Modell der Steuerung

Auch das Modell der Steuerung wird hierarchisch strukturiert: Die Steuerung der Gesamtanlage besteht entsprechend der Struktur der physischen Umgebung aus Steuerungen für

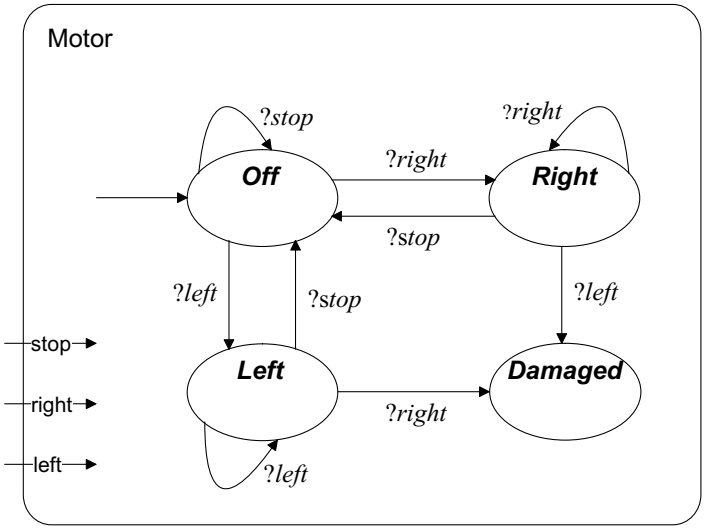


Abbildung 6.13: Motor-Modul.

die Transportbänder, Drehtische, Maschinen, usw. Die Schnittstelle des Steuerungsmodells enthält alle zur Steuerung der physischen Umgebung notwendigen Schnittstellenelemente des physischen Modells sowie die Schnittstellenelemente des Werkstück-Modells, da der Steuerungsablauf durch das Werkstück vorgegeben wird.

Die einzelnen Komponenten der Steuerung kommunizieren auf gleicher Hierarchieebene miteinander. So kommunizieren z. B. zwei Bänder miteinander, um ein Werkstück zu übergeben oder einer Maschine Bereitschaft zu signalisieren. Dies bedeutet, daß für die Kommunikation innerhalb des Steuerungsmodells zusätzliche Signale erforderlich sind, die im Modul Controller als LOCAL deklariert werden, da sie nach außen (für die Modelle von physischer Umgebung und Werkstück) nicht sichtbar sein dürfen. Da es sich bei dieser Synchronisation um ein Warten auf gegenseitige Bereitschaft handelt, werden zu diesem Zweck mehrfach einschränkbare Signale benutzt (MULTREST).

Kommunikation: Der verwendete Kommunikationsmechanismus wird am Beispiel eines Transportbandes erläutert. Das Steuerungsprogramm eines Transportbandes soll auf die physische Umgebung so einwirken, daß das Werkstück am Anfang des Bandes von der Vorgängerkomponente (Produzent) übernommen, zum anderen Ende transportiert und dort an die Nachfolgerkomponente (Konsument) übergeben wird. Damit ist in der Steuerung der folgende Ablauf notwendig:

1. Warten, bis der Produzent ein Werkstück übergeben möchte.
2. Dem Produzenten mitteilen, daß der Transfer stattfinden kann.
3. Bewegung des Transportbandes starten.
4. Warten, bis das Werkstück vollständig auf dem Band liegt (bis es am ersten Sensor angekommen ist).

5. Dem Produzenten mitteilen, das der Transfer abgeschlossen ist.
6. Transport des Werkstücks bis zum zweiten Sensor am Ende des Bandes.
7. Bandbewegung anhalten.
8. Dem Konsumenten mitteilen, daß ein Werkstück zur Übergabe bereit liegt.
9. Warten, bis der Konsument zur Übergabe bereit ist.
10. Bandbewegung starten.
11. Warten, bis der Konsument mitteilt, das das Werkstück erfolgreich übernommen wurde.
12. Bandbewegung anhalten. (Jetzt liegt wieder Bereitschaft für Schritt 1 vor.)

Die Synchronisation bei der Werkstückübergabe von Band zu Band erfolgt nach dem folgenden Protokoll: Die Steuerung der Vorgänger-Komponente kündigt der Nachfolger-Komponente den Beginn eines Werkstücktransfers an. Im Modell kann die entsprechende, mit dem Signal startInTransfer synchronisierte Transition erst geschaltet werden, wenn beide Komponenten bereit sind, eine Transition mit dieser Synchronisationsmarke (beim Nachfolger mit startOutTransfer bezeichnet) zu schalten. Analog wird beim Abschluß des Transfers verfahren. Der Transfer ist abgeschlossen, wenn beide Komponenten eine entsprechend synchronisierte Transition (nun mit dem Signal stopInTransfer beim Nachfolger bzw. stopOutTransfer beim Vorgänger) schalten. Die Verknüpfung der Signale zweier Komponenten für diese Kommunikation wird in Abbildung 6.14 auf der nächsten Seite verdeutlicht. Solange eine der Komponenten nicht bereit ist, muß entsprechend die andere (bzw. die anderen) warten.

Bestandteile:

- 4 Steuerungen für Transportbänder (Modul TransportBeltBehavior).
- 2 Steuerungen für Arbeitsbänder (Modul WorkBeltBehavior).
- 2 Steuerungen für ein langes Band, bestehend aus einem Transportband und einem kurzen Band (Modul LongBeltBehavior).
- 2 Steuerungen für Drehtische (Modul TurnTableBehavior).
- 1 Steuerung für die Presse (Modul PressMachineBehavior).
- 1 Steuerung für die Bohrmaschine (Modul DrillMachineBehavior).

Im folgenden werden einige Steuerungsmodule für die korrespondierenden Hardware-Komponenten hinsichtlich Schnittstellenelemente und Verhalten beschrieben.

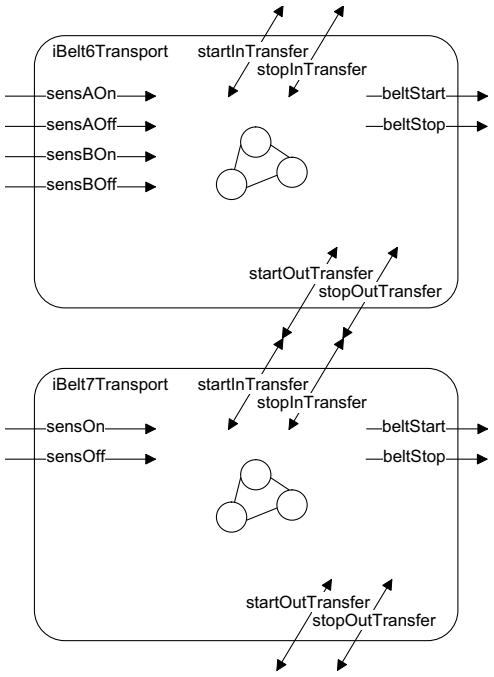


Abbildung 6.14: Synchronisation der Bandsteuerungen anhand zweier konkreter Modulinstanzen.

6.4.2.1 Transportband

Vom Modul TransportBeltBehavior wird die Steuerung eines Transportbandes modelliert (siehe Abbildung 6.16 auf der nächsten Seite).

Eingabe-Signale: Über die Signale sensAOn und sensAOff fragt die Steuerung den Zustandswechsel des ersten Sensors im korrespondierenden physischen Modell des Transportbandes ab. sensBOn und sensBOff erfüllen diese Funktionalität für den zweiten Sensor.

Ausgabe-Signale: Mit den Signalen beltStart (Starten des Bandes) und beltStop (Anhalten des Bandes) wirkt das Steuerungsmodul auf die physische Umgebung ein.

Mehrfach einschränkbare Signale: Das Signal startInTransfer startet den Vorgang des Werkstücktransfers vom vorherigen zu diesem Band, stopInTransfer beendet diesen Vorgang. Beim Transfer des Werkstücks zum Folgeband dienen die Signale startOutTransfer dem Starten und stopOutTransfer dem Beenden.

Lokale Uhren: Die Uhr c dient der Steuerung des Zeitverhaltens.

Verhalten: Als initialer Zustand wird ein angehaltenes Band ohne Werkstück angenommen. Sobald die Synchronisation mit der Steuerung des vorherigen Bandes (bzw. eines Drehtisches) zum Starten des Werkstücktransfers erfolgte (startInTransfer), gibt die Steuerung das Signal zum Starten des eigenen Bandes (beltStart). Ist der erste Bandsensor erreicht, wird das Band angehalten (beltStop) und der Werkstücktransfer mit dem vorherigen Band beendet (stopInTransfer).

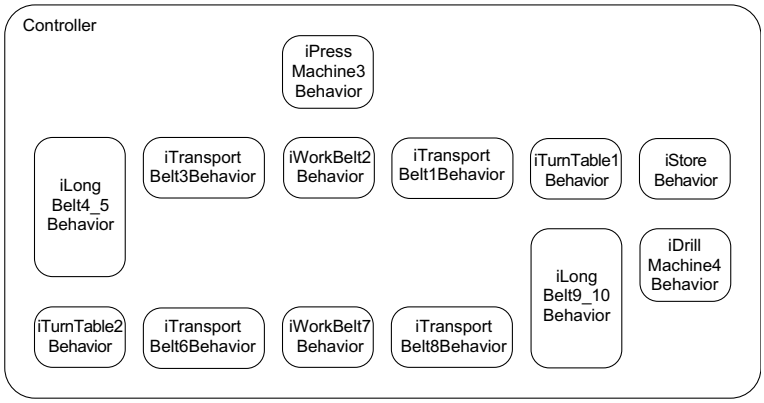


Abbildung 6.15: Gesamtsteuerung der Fertigungsanlage.

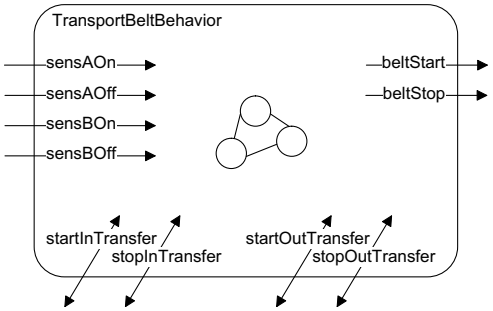


Abbildung 6.16: Steuerungsmodul für ein Transportband.

Zum Weitertransport des Werkstücks wird der Motor wieder gestartet. Beim Erreichen des zweiten Sensors, wird das Band erneut angehalten. Es kann eine Synchronisation zum Start des Werkstücktransfers von diesem zum nächsten Band (startOutTransfer) erfolgen. Nach Synchronisation wird der Motor wieder gestartet. Sobald das Werkstück den Bereich des zweiten Sensors verläßt, wird auf das Signal zum Beenden des Transfers gewartet (stopOutTransfer). Das Band wird nach dem Eintreffen dieses Signals angehalten und hat seinen initialen Zustand wieder erreicht.

6.4.2.2 Langes Band

Die Steuerung für ein ”langes Transportband” steuert zwei Bänder: ein kurzes Band ohne Sensoren und ein (dem kurzen Band im Fertigungsablauf folgendes) Transportband mit zwei Sensoren. Diese Zusammenfassung liegt darin begründet, daß das kurze Band keinen Sensor hat und somit kein Werkstück parken kann. Das kurze Band wird immer gemeinsam mit dem zugeordneten Transportband eingeschaltet. Das Modul wird in Abbildung 6.17 gezeigt. Die Steuerung kann auch als Steuerung einer virtuellen Komponente aufgefaßt werden, d. h. es werden die Steuerungen für zwei einzelne Bänder zu

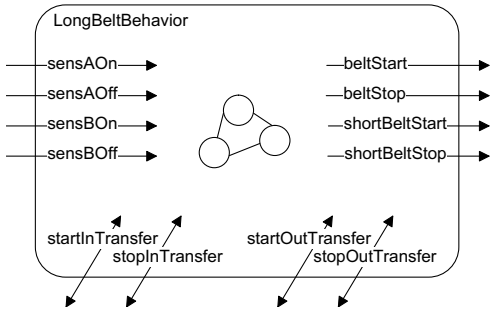


Abbildung 6.17: Steuerungsmodul für ein langes Band.

einer Steuerung so zusammengefaßt, als würde es sich um die Steuerung einer einzelnen Komponente handeln.

Eingabe-Signale: Die Signale sensAOn und sensAOff stellen Veränderungen am ersten Sensor fest. sensBOn und sensBOff gelten entsprechend für den zweiten Sensor.

Ausgabe-Signale: Durch die an die Umgebung gerichteten Signale beltStart und beltStop wird das Transportband von der Steuerung gestartet bzw. gestoppt. shortBeltStart und shortBeltStop sind die analogen Signale für das kurze Band.

Mehrfach einschränkbare Signale: startInTransfer startet den Werkstücktransfer vom vorhergehenden auf dieses Band, stopInTransfer beendet diesen Vorgang. Die Signale startOutTransfer und stopOutTransfer dienen der Steuerung der Werkstückübergabe an die nächste Transportkomponente.

Lokale Uhren: Die Uhr c dient der Steuerung des Zeitverhaltens.

Verhalten: Im initialen Zustand sind beide Bänder gestoppt, und es befindet sich kein Werkstück auf ihnen. Sobald das Signal zum Starten des Werkstücktransfers von der Steuerung des vorherigen Bandes eintrifft, werden das kurze Band und das Transportband in Bewegung gebracht. Sobald der erste Sensor des Transportbandes erreicht ist, werden beide Bänder gestoppt, der Werkstücktransfer mit dem vorherigen Band wird beendet. Jetzt wird nur das Transportband gestartet, das kurze Band bleibt gestoppt. Erreicht das Werkstück den zweiten Sensor, hält das Transportband erneut. Das Signal zum Start des Werkstücktransfers vom Transportband zur nächsten Transport-Komponente (startOutTransfer) wird gesetzt. Nach erfolgreicher Synchronisation mit der nächsten Komponente (bzw. Drehtisch) wird das Transportband wieder gestartet. Sobald das Werkstück den zweiten Sensor verläßt, wird auf das Signal zum Beenden des Transfers gewartet, durch das das Band angehalten wird und seinen initialen Zustand erreicht.

6.4.2.3 Bohrmaschine

Die Steuerung für eine Bohrmaschine wird stellvertretend für alle Maschinentypen beschrieben (siehe Abbildung 6.18 auf der nächsten Seite).

Eingabe-Signale: Das Signal startProcess löst den Beginn des Bohrvorgangs aus, es wird von der Steuerung des Bearbeitungsbandes erwartet, auf dem das Werkstück liegt. Vom physischen Umgebungsmodell werden die Signale topSensOn und topSensOff zum

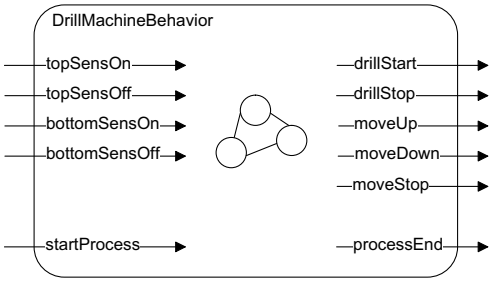


Abbildung 6.18: Steuerung für die Bohrmaschine (Maschine 4).

Feststellen des Zustandswechsels des oberen Sensors benutzt. bottomSensOn und bottomSensOff sind die entsprechenden Eingabe-Signale für den unteren Sensor.

Ausgabe-Signale: Die Signale drillStart und drillStop steuern die Drehbewegung der Bohrspindel und somit des Bohrers. Die Auf- und Abwärtsbewegung der Bohrspindel wird von den Signalen moveUp und moveDown gestartet und vom Signal moveStop beendet. Das Signal processEnd gibt der Steuerung des Bearbeitungsbandes bekannt, daß der Bohrvorgang abgeschlossen ist und die Maschine sich wieder in der Ausgangsposition befindet.

Die Signale zum Starten und Beenden des Bohrvorgangs dienen wieder der internen Kommunikation zwischen der Steuerung der Maschine und der Steuerung des Bearbeitungsbandes, dem die Maschine zugeordnet ist. Diese Signale sind nicht in der physischen Umgebung der Anlage vorhanden. Da die Maschine sich nach der Bearbeitung des Werkstücks wieder in der Ausgangslage befindet, ist für die Synchronisation mit dem zugehörigen Bearbeitungsband ein vereinfachtes Protokoll anwendbar.

Verhalten: Als initiale Position der Bohrspindel wird der obere Anschlagpunkt angenommen, wobei sich die Spindel nicht dreht. Durch das Signal zum Starten des Bohrprozesses werden die Drehbewegung des Bohrers (drillstart) und die Abwärtsbewegung der Bohrspindel (moveDown) gestartet. Hat die Bohrspindel den unteren Sensor erreicht, wird ihre Abwärtsbewegung angehalten. Die Aufwärtsbewegung der Bohrspindel zur Ausgangsposition erfolgt analog. Die Drehbewegung des Bohrers wird gestoppt (drillStop) und das Beenden des Bohrprozesses signalisiert (processEnd). Nach dieser Synchronisation kehrt die Steuerung der Bohrmaschine wieder in den Initialzustand zurück.

6.4.3 Modell des Werkstücks mit Abarbeitungsprogramm

Der Bearbeitungsablauf, d. h. welchen Weg das Werkstück durch die Produktionsanlage nimmt und von welchen Maschinen es bearbeitet wird, soll nicht starr im Steuerungsverhalten der einzelnen Komponenten integriert werden, sondern unabhängig von den Steuerungen programmiert werden können.

Diese Anforderung wird erfüllt, indem die Abfolge der Bearbeitungsschritte von der Steuerung entkoppelt in einem separaten Modell für das Werkstück festgelegt wird, wodurch die eigentliche Steuerung flexibel mit verschiedenen Werkstücken umgehen kann.

Jedes Werkstückmodell enthält das Abarbeitungsprogramm und den Bearbeitungsfortschritt für das jeweilige Werkstück. Somit stellt das Werkstückmodell die den Steuerungsfluß bestimmende Aktivkomponente dar.

Auf einem Bearbeitungsband gibt es zum Beispiel zwei Möglichkeiten im Ablauf: Das Werkstück kann von der Maschine bearbeitet werden oder die Maschine ohne Bearbeitung passieren. Die Entscheidung wird vom Werkstückmodell getroffen.

6.4.4 Virtuelle Komponenten zur Verringerung der Komplexität

Einige Anlagenteile können aufgrund komplexer Kommunikations- und Abhängigkeitsstrukturen unübersichtlich werden. Um dennoch eine klare, verständliche Modellstruktur zu konstruieren, sind bei der Modellierung der vollständigen Anlage besondere Konzepte, wie z. B. die Bildung virtueller Komponenten, notwendig.

Virtuelle Komponenten als Abstraktion. Virtuelle Komponenten bzw. virtuelle Maschinen dienen der abstrakteren Modellierung mehrerer Komponenten, die direkt miteinander interagieren und gemeinsam eine Aufgabe erfüllen. Sie können im physischen Modell und im Steuerungsmodell eingesetzt werden (siehe langes Band).

Virtuelle Komponenten als Bearbeitungsteilprogramm. Für eine Gruppe von Komponenten oder für eine einzelne Komponente im physischen Modell können im Steuerungsmodell mehrere virtuelle Maschinen existieren, die jeweils eine bestimmte Bearbeitungsreihenfolge festhalten und vom Werkstückmodell durch eine einzige Bearbeitungsentscheidung ausgewählt werden können. Dabei wird ein Teil des Abarbeitungsprogramms vom Werkstück in das Steuerungsmodell ausgelagert. Nachdem der Bearbeitungsbefehl an die entsprechende Komponente gegeben wurde (der ”Aufruf” eines im Steuerungsmodell integrierten Programms), wird bis zum Beenden der Bearbeitung gewartet. Der durch die Komponente ausgeführte Arbeitszyklus gliedert sich meist in mehrere Unterschritte (bei der Gruppe um den Schiebetisch sind es 12 Schritte).

Virtuelle Komponente für die Komponentengruppe um den Schiebetisch. Da die Maschinen 1 und 2, der Tisch, die Schieber 1 und 2 sowie Band 7 in einer bestimmten Reihenfolge mehrmals zur Anwendung kommen, sollte dieser Ablauf bei der Modellierung der Steuerung in einer Virtuellen Maschine zusammengefaßt werden. Neben der Ablaufsteuerung sorgt die virtuelle Maschine auch für die korrekte Behandlung gemeinsam genutzter Sensoren (Schieber 1, Band 7).

Das Benutzen virtueller Komponenten bietet den folgenden Vorteil: Hinter einer *genormten Schnittstelle* für die Kommunikation befindet sich nach außen nicht sichtbar eine *einfache oder komplizierte* Maschine bzw. eine Menge interagierender Komponenten.

6.5 Spezifikation und Verifikation

Nachdem ein komplettes Systemmodell vorliegt, können die Anforderungen an das System auf das Modell bezogen konkretisiert und formal spezifiziert werden. Daran anschließend werden mittels Erreichbarkeitsanalyse alle Sicherheitseigenschaften des Modells verifiziert; es wird auch auf die Verfeinerungsanalyse eingegangen.

6.5.1 Einige Sicherheitsanforderungen der Anlage

Bei der exemplarischen Darstellung einiger von der Steuerung zu erfüllenden Sicherheitsanforderungen wird unterschieden zwischen folgenden Kategorien:

1. *Kapazität:*
   (a) Zu jeder Zeit darf sich höchstens ein Werkstück auf einem Förderband (bzw. Schiebetisch) befinden.
   (b) Zu keinem Zeitpunkt darf je ein Werkstück auf Schiebetisch und Förderband 7 liegen.
2. *Koordination:*
   (a) Transporteinheit $\Leftrightarrow$ Maschine:
      i. Wird ein Werkstück von einer Maschine bearbeitet, darf sich das Förderband (bzw. der Schiebetisch) nicht bewegen (damit das Werkstück korrekt bearbeitet werden kann und die Maschinen nicht beschädigt werden).
      ii. Wenn sich das Förderband (bzw. der Schiebetisch) bewegt, muß sich die Maschine in ihrem Ausgangszustand befinden (da sonst ein Werkstück in eine arbeitende Maschine geschoben werden könnte).
   (b) Transporteinheit $\Leftrightarrow$ Schieber:
      i. Wenn sich das Förderband 7 (bzw. der Schiebetisch) bewegt, dürfen sich die Schieber nicht bewegen.
   (c) Schieber $\Leftrightarrow$ Schieber:
      i. Wenn sich einer der beiden gegenüberliegenden Schieber bewegt, muß sich der andere Schieber in seiner Ausgangsposition befinden (sonst würden sie das Werkstück bzw. einander zerstören).
3. *Bearbeitungsvorgang:*
   (a) Eine Maschine bzw. ein Schieber dürfen sich nur bewegen, wenn ein Werkstück da ist (damit die Maschine nicht ins Leere bohrt).
   (b) Eine Maschine darf sich nicht über die obere bzw. untere End-Position hinaus bewegen. Der Arbeitskopf muß innerhalb einer begrenzten Reaktionszeit im Bereich des Anschlagsensors gestoppt werden.
   (c) Wenn sich die Bohrspindel der Bohrmaschine in der unteren Position befindet, darf sie nicht angehalten werden (damit sich der Bohrer nicht verklemmt).
   (d) Ein Drehtisch darf sich nicht über die linke bzw. rechte End-Position hinaus bewegen, sondern er muß immer innerhalb einer begrenzten Reaktionszeit im Bereich der Sensoren zum Stehen kommen.

4. *Blockierungsfreiheit:*

   (a) Spätestens 1150 Zeiteinheiten[1] nach der Initialisierung der Anlage muß jedes Werkstück fertig bearbeitet worden sein.

   (b) Das kurze Band muß sich beim Auswerfen wenigstens 35 Zeiteinheiten bewegen, damit das Werkstück sicher ausgeworfen wird.

5. *Bearbeitungsgeschwindigkeit:*

   (a) Für den einfachen Transportweg vom Förderband 1, Sensor 1 bis zum Förderband 10, Sensor 2 darf das System nicht länger als 800 Zeiteinheiten benötigen.

6.5.2 Annahmen über den Initialzustand der Umgebung

Im Modell der Fertigungsanlage wird von bestimmten Annahmen über den physischen Zustand beim Starten der Anlage ausgegangen, welche die Voraussetzung für das korrekte Verhalten der Steuerung darstellen:

Förderbänder: Auf den Förderbändern befindet sich kein Werkstück.

Drehtische: Auf den Drehtischen befindet sich kein Werkstück, sie sind jeweils dem Förderband zugewandt, von dem sie ein Werkstück empfangen.

Maschinen: Die Arbeitsköpfe aller Maschinen befinden sich in der Ausgangsposition (oben).

Schieber: Vor keinem Schieber befindet sich ein Werkstück, alle Schieber haben ihren Arm eingezogen.

Schiebetisch: Auf dem Tisch befindet sich kein Werkstück, er steht an seiner rechten Position.

Diese Annahmen spiegeln sich in der Menge der Initialkonfigurationen wieder. In jedem Modul und bei jeder Instanziierung kann diese Menge definiert werden. Vor dem Start der Erreichbarkeitsanalyse wird aus den einzelnen Initial-Mengen der Komponenten die Initial-Menge des Gesamtmodells zusammengesetzt, d. h. diese Menge muß nicht manuell für die Analyse definiert werden, sondern wird automatisch dem Modell entnommen. Die Definition der Menge von Initialkonfigurationen für die gesamte Fertigungsanlage besteht aus 189 Zeilen.

6.5.3 Spezifikation der verbotenen Konfigurationsmengen

Die Erreichbarkeitsanalyse berechnet alle von der Menge der Initialkonfigurationen aus erreichbaren Konfigurationen und überprüft, ob in dieser Menge bestimmte (Fehler-) Konfigurationen enthalten sind. Die gesuchten Konfigurationsmengen der Sicherheitseigenschaften werden in der vom Werkzeug angebotenen Analysesprache definiert. Nach diesem Prinzip können auch alle anderen, den jeweiligen Sicherheitseigenschaften entsprechenden Fehlerkonfigurationen formuliert werden. Es folgen einige Beispiele:

[1] Im konkreten Modell der Fertigungsanlage entspricht eine Zeiteinheit einer realen Zeit von 100 ms.

- Eigenschaft 2.a.i): Es darf nie die Situation auftreten, daß sich eine Maschine bei laufendem Motor des Bearbeitungsbandes nicht in der Ausgangsstellung am oberen Anschlag befindet.

```
BMError :=
 STATE(iPhysical.iWorkBelt2.iBeltMotor.MotorBehavior) = On
      AND
 STATE(iPhysical.iPressMachine3.MachinePosition) != Top;
```

- Eigenschaft 3.b): Erhält eine Maschine während einer vertikalen Bewegung beim Erreichen des Endsensors innerhalb der vorgegebenen Reaktionszeit kein Signal zum Stoppen des Motors, so wechselt der Automat, der das Verhalten für die Komponente modelliert, in einen Fehlerzustand. Dieser Zustand darf nie erreicht werden.

```
MPositionError :=
 STATE(iPhysical.iPressMachine3.MachinePosition)
      = PositionError;
```

- Eigenschaft 3.c): Die Bohrspindel darf nie angehalten werden, wenn sie sich in der untersten Position befindet.

```
MDrillError :=
 STATE(iPhysical.iDrillMachine4.MachinePosition) = Bottom
      AND
 STATE(iPhysical.iDrillMachine4.iDrillMotor.MotorBehavior)
      = Off;
```

- Eigenschaft 3.d): Erhält ein Drehtisch während einer Drehbewegung beim Erreichen des Sensors innerhalb der vorgegebenen Reaktionszeit kein Signal zum Stoppen des Motors, so wechselt der modellierende Automat in einen Fehlerzustand. Dieser Zustand darf nie erreicht werden.

```
TTError :=
 STATE(iPhysical.iTurnTable1.CheckTurnControl) = TurnError;
```

- Eigenschaft 4.b): Die Uhr des kurzen Bandes, die die Zeit für das Auswerfen mißt, darf nach dem Stoppen des Bandes keinen Wert kleiner als 35 haben.

```
BeltError :=
 STATE(iController.iBelt4_5Transport.BeltControl)
      = stopShortBelt2 AND
 iController.iBelt4_5Transport.c <= 35
```

Durch die Disjunktion aller in dieser Weise angegebenen Konfigurationsmengen entsteht die Menge der von der Erreichbarkeitsanalyse zu überprüfenden Fehlerkonfigurationen des Gesamtsystems.

```
error := BMError OR
         MPositionError OR
         MDrillError OR
         TTError OR
         BeltError;
```

6.5.4 Erreichbarkeitsanalyse

In diesem Abschnitt werden die Ergebnisse der Verifikation des Modells mittels Erreichbarkeitsanalyse vorgestellt. Die hier wiedergegebenen Meßwerte basieren auf Analysen auf einem Rechner mit einem AMD-Prozessor Athlon mit 1 GHz Taktfrequenz unter dem Betriebssystem Linux (Kernel 2.4). Für die BDD-basierten Berechnungen wurden 400 MB Hauptspeicher verwendet.

Bei der Erreichbarkeitsanalyse werden zunächst alle von der Menge der Initialkonfigurationen erreichbaren Konfigurationen berechnet. Im nächsten Schritt wird der Durchschnitt dieser Erreichbarkeitsmenge mit der Menge der Konfigurationen, die den Anforderungen im Modell entsprechend nicht auftreten dürfen, berechnet. Ist die Schnittmenge leer, wird eine positive Meldung ausgegeben, anderenfalls eine entsprechende negative Meldung und zur Fehlersuche die Menge der erreichten Fehlerkonfigurationen. In der Schreibweise der CTA-Analysesprache werden folgende Analyseanweisungen formuliert:

```
reached := REACH FROM initial FORWARD;

reachedError := reached INTERSECT error;

IF( ISEMPTY(reachedError) )
{
  PRINT "Safety requirements fulfilled.";
}

ELSE
{
  PRINT "Safety requirements not fulfilled.";
  PRINT "Set of reached error configurations: " reachedError;
}
```

Für das Modell mit einem Werkstück werden diese Verifikationsschritte vom Werkzeug Rabbit innerhalb von 136 Sekunden durchgeführt. Das Ergebnis der Analyse zeigt, daß das entwickelte Modell alle geprüften Anforderungen vollständig erfüllt.

Variablenordnen. Für ein Modell mit einer einfachen Kommunikationsstruktur wie Fischers Protokoll scheint es offensichtlich, wie die Komponenten für eine gute Variablenordnung anzuordnen sind. Die für Fischers Protokoll angegebene Variablenordnung für polynomiellen Analyseaufwand entspricht der Intuition des Modellierers.

Nimmt die Komplexität des Modells in starkem Maße zu – wie bei der Fertigungsanlage – ist es für den Modellierer schwierig, eine Intuition für eine gute Variablenordnung zu entwickeln, da die Modellstruktur für den erforderlichen Überblick viel zu komplex ist (es sind 143 Variablen zu ordnen). Das manuelle Entwickeln einer guten Variablenordnung erfordert BDD-Expertenwissen und ist aufwendig. Daher sind automatische Methoden zum Berechnen einer guten Variablenordnung erforderlich.

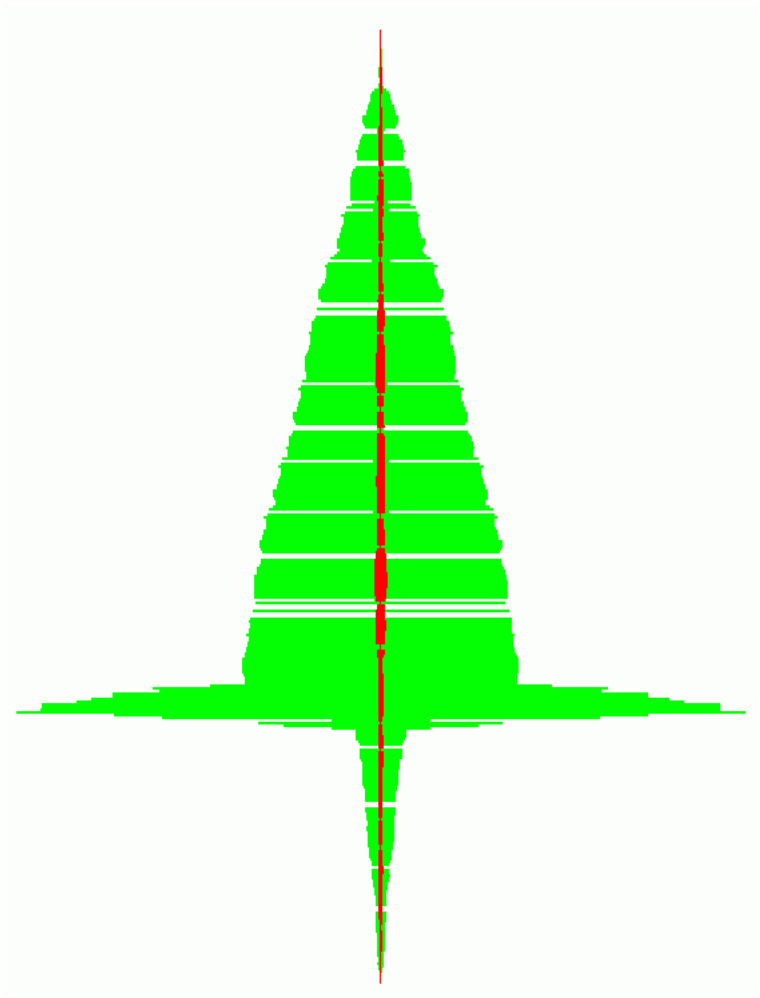


Abbildung 6.19: BDD-Gestalt der Erreichbarkeitsmenge für zwei verschiedene Variablenordnungen (Fertigungsanlage mit einem Werkstück). Die große Gestalt resultiert aus der Präfix-Linearisierung, die schmale Gestalt aus der schätzungsbasierten Ordnung (Maximalbreite des BDDs: 6596 BDD-Knoten).

In Abbildung 6.19 sind für zwei verschiedene Variablenordnungen die BDD-Gestalten der Erreichbarkeitsmenge dargestellt (Erklärung der Visualisierung siehe Abbildung 3.15

auf Seite 78). Im ersten Experiment wurde die Variablenordnung genutzt, die aus der Präfix-Linearisierung der Modulstruktur entsteht. Damit wurde das Wissen des Modellierers über die Kopplung zwischen Komponenten berücksichtigt. Die so bestimmte Variablenordnung ist viel besser als eine zufällig gewählte Ordnung. Mit einer zufällig gewählten Variablenordnung wäre eine Analyse dieses Modells aufgrund des Aufwands nicht denkbar. Der BDD umfaßt insgesamt 374529 Knoten (große Figur); in seiner breitesten Ebene wird eine (Binär-) Variable durch 6596 Knoten repräsentiert.

Wird die in Abschnitt 3.3.2 entwickelte BDD-Größenschätzung für die Berechnung der Variablenordnung verwendet, entsteht eine deutlich bessere Variablenordnung (schmale Figur). Bei dieser Berechnung wird unter Verwendung der Größenschätzung als Kostenfunktion ein heuristischer Algorithmus zur Optimierung angewendet, der BDD enthält insgesamt nur noch 14895 Knoten. Dennoch ist die aus der Modulstruktur gewonnene Präfix-Linearisierung von entscheidender Bedeutung, denn auf der Grundlage einer zufälligen Initial-Variablenordnung berechnet der heuristische Algorithmus schlechtere Variablenordungen als auf der Grundlage der Präfix-Linearisierung.

Variablenordnung	Präfix	Schätzung
Anzahl Konfigurationen	55857	55857
Speicherbedarf (in BDD-Knoten)	378229	14895
Verifikationszeit (in Sekunden)	14553	136

Tabelle 6.1: Einfluß des schätzungsbasierten Variablenordnens auf die Verifikationsperformance beim Modell der Fertigungsanlage (1 Werkstück).

In der Tabelle 6.1 wird der enorme Vorteil der automatischen, schätzungsbasierten Berechnung guter Variablenordnungen (Spalte ”Schätzung”) gegenüber der Präfix-Linearisierung (Spalte ”Präfix”) verdeutlicht. Der Speicherbedarf (Anzahl BDD-Knoten) verringert sich beim Anwenden der schätzungsbasierten Variablenordnung auf weniger als $\frac{1}{20}$ der ursprünglichen Größe, die Verifikationszeit beträgt durch diese Strategie weniger als $\frac{1}{100}$ der Zeit. Diese Strategie zum schätzungsbasierten Variablenordnen ist im Werkzeug Rabbit integriert.

6.5.5 Verfeinerungsanalyse

Im Einordnungskapitel in Abschnitt 1.2.2 wurden verschiedene Anwendungen für die *Spezifikation mittels Modell* vorgestellt. Um die Wirkungsweise der Verfeinerungsanalyse zu verdeutlichen, wird im folgenden eine dieser Techniken, das modulare Beweisen, erläutert. Die Aufgabe besteht darin, das komplette Modell der Fertigungsanlage mit *zwei* Werkstücken auf eine Erreichbarkeitseigenschaft hin zu überprüfen. Sei diese Eigenschaft zum Beispiel, ob eine Situation auftritt, bei der sich sowohl der Arbeitskopf einer Maschine als auch das zugehörige Bearbeitungsband bewegen (Eigenschaft 2.a.i). Da der Aufwand für die Erreichbarkeitsanalyse des Gesamtmodells groß ist, wird die Methode des modularen Beweisens angewendet.

Um den enormen Analyseaufwand bei der Erreichbarkeitsanalyse des Gesamtmodells mit zwei Werkstücken zu demonstrieren, zeigt Abbildung 6.20 den BDD der Erreichbar-

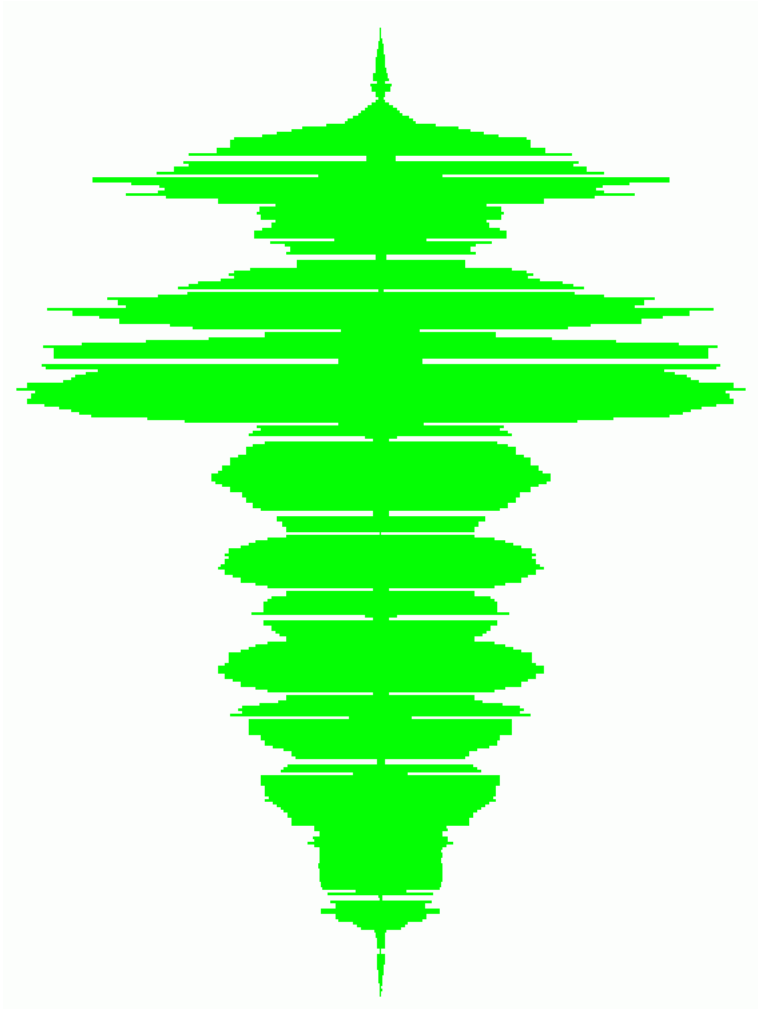


Abbildung 6.20: BDD-Gestalt der Erreichbarkeitsmenge für das Modell mit zwei Werkstücken (Maximalbreite des BDDs: 2369 BDD-Knoten).

keitsmenge mit einer Größe von 271685 Knoten (bei schätzungsbasiertem Variablenordnen). Die Berechnungszeit für die Erreichbarkeitsmenge beträgt 52009 s (ca. 14.5 Stunden) bei $2.6 \cdot 10^8$ erreichbaren Konfigurationen. Werden mehr als zwei Werkstücke verwendet, erhöht sich die Analysezeit weiter und die Verifikation ist nur noch unter Anwendung des modularen Beweisens sinnvoll.

Voraussetzung für das modulare Beweisen ist, daß im Modell Aspekte identifiziert werden können, die auf die nachzuweisende Eigenschaft keinen Einfluß haben. Für jede Art der Eigenschaften müssen diese Aspekte evtl. neu gewählt werden, weil keine für sie relevanten Modellteile ”wegabstrahiert” werden dürfen, die Abstraktion jedoch auch hinreichend stark sein sollte, um den Zeit- und Speicheraufwand bei der Erreichbarkeitsanalyse deutlich zu verkürzen. Vorteilhaft ist es, wenn die in früheren Entwicklungsphasen des Modells entstandenen gröberen Versionen für Komponenten einsetzbar sind, da dadurch der Aufwand für die Entwicklung der abstrakteren Version entfällt.

Aspekte, von denen abstrahiert werden kann, sind in diesem Beispiel die Abweichungen für das Zeitverhalten der Sensoren, d. h. innerhalb eines Toleranzbereichs darf der Sensor das ankommende Werkstück früher oder später als erwartet erkennen. Außerdem sind für Messungen des Durchsatzes der Fertigungsanlage, d. h. wieviele Werkstücke pro Zeiteinheit von der Anlage bearbeitet werden können, in den Steuerungen Uhren zum Messen der Zeiten für die einzelnen Bearbeitungsschritte eingebaut worden. Diese und weitere Aspekte des Zeitverhaltens sind für den Nachweis der oben erwähnten Sicherheitseigenschaft nicht erforderlich. Daher wird das Komponentenmodell für das Band durch eine Version ohne Uhren ersetzt, welches sich abgesehen vom Zeitverhalten genauso verhalten soll wie die Version mit Uhren. Weil das Gesamtmodell mehrere Instanzen des Band-Moduls enthält, wird durch diesen Schritt die Anzahl der Uhren im Modell von 44 auf 12 Uhren reduziert.

Nachdem ein abstrakteres Gesamtmodell durch das Ersetzen der Band-Module mit Uhren (TimedBelt) durch Band-Module ohne Uhren (UntimedBelt) entstanden ist, kann die Analyse in zwei Schritten ausgeführt werden:

1. Es muß mittels Verfeinerungsanalyse bewiesen werden, daß zwischen den Modulen die Simulationsrelation "UntimedBelt simuliert TimedBelt" existiert, d. h. daß das Modul TimedBelt ein spezielleres Verhalten als das Modul UntimedBelt hat.

2. Mittels Erreichbarkeitsanalyse muß bewiesen werden, daß die Sicherheitseigenschaft im abstrakteren Gesamtmodell gilt.

Sind beide Teil-Beweise erfolgreich durchgeführt worden, darf geschlossen werden, daß auch das ursprüngliche Gesamtmodell mit speziellerem Zeitverhalten die Sicherheitseigenschaft erfüllt. Dieser Schluß ist zulässig, weil durch das Ersetzen des speziellen durch das abstrakte Modul keine erreichbaren Konfigurationen weggefallen sind; bei einem abstrakteren Modell im Sinne der Verfeinerungsanalyse kommen immer nur mehr Verhaltensweisen hinzu.

Modellversion	vollständig	reduziert
Anzahl Konfigurationen	$2.6 \cdot 10^8$	$9.8 \cdot 10^7$
Speicherbedarf (in BDD-Knoten)	271685	181712
Rechenzeit (in Sekunden)	52009	10377

Tabelle 6.2: Laufzeiten unterschiedlicher Verifikationsaufgaben (2 Werkstücke).

In der Tabelle 6.2 werden die Verifikationszeiten miteinander verglichen. Eine deutliche Reduzierung der Rechenzeit bei Verwendung eines abstrakteren, reduzierten Modells (mit Instanzen des Moduls UntimedBelt) gegenüber dem ursprünglichen, vollständigen Gesamtsystem (mit Instanzen des Moduls TimedBelt) bei der Erreichbarkeitsanalyse wird nachgewiesen. Dieses Ergebnis zeigt, daß eine Abstraktion in einem kleinen Teil des Systemmodells große Auswirkungen auf die Verifikationszeit haben kann. Da die einzelnen Band-Module klein sind, benötigt die Berechnung der Simulationsrelation wenig Rechenzeit (0.46 Sekunden). Insgesamt kann also die Rechenzeit bereits bei dieser einfachen Abstraktion auf 20 % verkürzt werden.

In einem Modell wie dem der Fertigungsanlage ist es sinnvoll, die durch den Einsatz virtueller Maschinen gewonnene Abstraktion für die Verfeinerungsanalyse zu nutzen. Dazu kann zum Beispiel die gesamte virtuelle Maschine, die aus den Komponenten Band 7, Schiebetisch, Schieber 1, Schieber 2, Maschine 1 und Maschine 2 besteht, durch eine andere, weiter abstrahierte, virtuelle Maschine ersetzt werden, die das Kommunikationsprotokoll der ursprünglichen virtuellen Maschine simulieren kann. Auch das vielfach zum Testen des Modells angewendete Modellierungsschema "Produzent-Komponente-Konsument" kann verwendet werden, um die Schnittstellen einer zu prüfenden Komponente zu belegen. Dabei werden die Komponenten, die die zu untersuchende Komponente umgeben, durch starke Abstraktionen ersetzt, wobei das von der Komponente einzuhaltende Kommunikationsprotokoll gewährleistet wird. Viele weitere Abstraktionsschemen zum Gewinnen analysierbarer Gesamtmodelle sind möglich.

6.6 Zusammenfassung

In dieser Fallstudie wurden die Lösungsvorschäge der vorangehenden Kapitel validiert. Alle in Kapitel 5 verwendeten Beispiel-Modelle haben den Nachteil, daß sie aus einer skalierbaren Menge von ähnlichen Komponenten zusammengesetzt sind und eine übersichtliche Struktur aufweisen. Die Modellierung und Verifikation von industriellen Realzeit-Systemen stellt nach wie vor eine Herausforderung für die Forschung im Bereich der formalen Methoden dar. In der Literatur wurden bereits einige Fallstudien zum Thema Timed Automata vorgestellt, jedoch erreichte keine den Umfang des hier beschriebenen Modells einer praxisrelevanten Fertigungsanlage.

Zur Veranschaulichung der Modellgröße folgen einige quantitative Kenngrößen: Das entwickelte Modell hat in der textuellen Version eine Größe von 5059 Zeilen CTA-Code. Die flache Version, bei dem die hierarchische Struktur und die Instanziierungen aufgelöst wurden, umfaßt 13475 Zeilen CTA-Code. Das Modell enthält 44 Uhren, 17 diskrete Variablen und 82 Automaten, die über 183 verschiedene Synchronisationsmarken kommunizieren. In diesem Modell werden die elektrischen Signale von 44 Sensoren verarbeitet und 28 Motoren mit verschiedenen Laufrichtungen angesteuert. Dies entspricht mehr als der doppelten Größe der Karlsruher Produktionszelle [LL95], die 13 Aktuatoren und 14 Sensoren umfaßt. Bei dieser Betrachtung ist noch zu berücksichtigen, daß keine der in [LL95] veröffentlichten Fallstudien auf Realzeit-Aspekte eingeht.

Es wurde nachgewiesen, daß die Modellierung durch hierarchische Strukturierung des Modells übersichtlich und flexibel durchgeführt werden kann. Dafür wurden die Konzepte und das Vorgehen bei der Modellierung sowie das System-Modell auszugsweise dargestellt. Die verschiedenen Möglichkeiten der Spezifikation von nachzuweisenden Eigenschaften des Modells wurden erläutert und die Ergebnisse der automatischen Verifikation mit Rabbit aufgeführt. Die geringen Analysezeiten weisen nach, daß das Werkzeug Rabbit in der Lage ist, große Systeme algorithmisch zu bearbeiten. Damit wurde die in Kapitel 1 gestellte Forderung 5 erfüllt, die theoretischen Konzepte zur hierarchischen Modellierung und die werkzeugunterstützte Verifikation an einem größeren System zu erproben.

Kapitel 7

Zusammenfassung und Ausblick

In den letzten 10 Jahren erlangte der Formalismus der Timed Automata im Bereich der Modellierung und Verifikation von Realzeit-Systemen immer größere Bedeutung. Sein Einsatz war jedoch mit wesentlichen Problemen verbunden, die im Rahmen dieser Arbeit gelöst worden sind:

- Größere Systeme ließen sich bisher nur kompliziert und unstrukturiert modellieren. Das Modell bestand aus einer Ansammlung von Automaten, die auf einer einzigen Ebene miteinander kommunizierten. Wiederkehrende Systemteile ließen sich nicht einheitlich und wiederverwendbar modellieren; sie mußten mehrfach definiert werden.

- Die algorithmische Verifikation war mit den bisherigen Ansätzen durch enorm hohen Berechnungsaufwand nur bei kleinen Modellen möglich (Zustandsraum-Explosion).

Der in dieser Arbeit eingeführte Modellierungsformalismus **Cottbus Timed Automata** erweitert die bestehenden Konzepte der Timed Automata um Konzepte für die **Modularität**. Das Verhalten eines CTA-Modells wird auf die Basis-Semantik von Timed Automata abgebildet. Durch die Möglichkeit zur hierarchischen Strukturierung mit Modulen und durch einen Instanziierungsmechanismus zum mehrfachen Verwenden von Modulen sowie deren Austauschbarkeit konnten auch größere Systeme übersichtlich modelliert werden.

Zur Verifikation von CTA-Modellen wurde eine **effiziente BDD-basierte Erreichbarkeitsanalyse** vorgestellt, die die in der Literatur bekannten Verifikationswerkzeuge für Timed Automata an Performance übertrifft. Eines der Schlüsselkonzepte war dabei, die modulare Struktur der Modelle zur Berechnung einer guten Variablenordnung zu nutzen. Zusätzlich wurde eine **Verfeinerungsanalyse** eingeführt, deren Algorithmen ebenfalls auf der effizienten BDD-Repräsentation basieren. Mit ihr sind auch größere Modelle verifizierbar, die mit reiner Erreichbarkeitsanalyse nicht analysierbar wären.

Bei der Modellierung hybrider Systeme reicht die Ausdruckskraft von Timed Automata nicht aus, um alle Aspekte eines Systems zu erfassen. Deshalb wurde der CTA-Formalismus zusätzlich um den Basisformalismus der hybriden Automaten erweitert; für

die Integration der Verifikation hybrider Modelle in das Werkzeug wurde eine rechnerinterne Repräsentation auf der Basis der Double Description Method umgesetzt, einer matrix-basierten Datenstruktur für konvexe Polyeder.

Die in dieser Arbeit vorgestellten Konzepte wurden in einer **Werkzeugimplementierung** realisiert. Das grundlegende Architekturprinzip ist die Flexibilität. So können nicht nur einzelne Bestandteile des Werkzeugs flexibel verändert und erweitert werden, sondern es wird auch die gesamte Repräsentationsdatenstruktur zur Laufzeit gewählt. Dadurch kann das Werkzeug für verschiedene (zukünftige) Repräsentationsarten als "Einsatzumgebung" genutzt werden, wenn neue Ideen zu Datenstrukturen und Algorithmen der Erreichbarkeitsanalyse schnell empirisch validiert werden sollen. Mit Hilfe mehrerer Modell-Beispiele wurde nachgewiesen, daß das in dieser Arbeit entstandene Werkzeug für die betrachteten Modelle effizienter ist als die bisher existierenden Verifikationswerkzeuge.

Abschließend wurde eine **Fallstudie** zur Modellierung und Verifikation eines größeren Systems vorgestellt, eine Fertigungsanlage mit 44 Sensoren und 28 Motoren. Dabei wurde auf die Vorgehensweise bei der Modellierung eingegangen und die hierarchische Struktur des Modells erklärt. Eigenschaften wurden exemplarisch spezifiziert und mittels Erreichbarkeitsanalyse überprüft. Unter Verwendung modularer Beweistechniken wurden Eigenschaften des Systems nicht nur für kleine Ausschnitte des Systems verifiziert, sondern auf das Gesamtsystem übertragen.

Offene Probleme und Erweiterungen. Aus den folgenden, in dieser Arbeit nicht bearbeiteten Aspekten ergeben sich für die Weiterentwicklung des vorgestellten Ansatzes interessante Fragestellungen:

- **Codeerzeugung.** Die formale Modellierung einer System-Steuerung und die anschließende Verifikation des Modells sagt wenig aus über die Korrektheit der letztendlich entworfenen Steuerungssoftware. Das Generieren eines ausführbaren Programms aus dem korrekten Modell für die Steuerung würde eine wesentliche Bereicherung und Vervollständigung der formalen Methode darstellen.
  Zu diesem Thema wurden am Lehrstuhl Software-Systemtechnik bereits einige Untersuchungen vorgenommen. Derzeit stehen ein Interpreter für CTA-Modelle, der eine als CTA-Modell gegebene Steuerung ausführt, und eine spezifische Schnittstelle für die Ansteuerung der Fertigungsanlage zur Verfügung. Die Eingabe-Signale des Modells werden dabei über den Parallelport eines PCs den Flanken der Sensoren in der Fertigungsanlage zugeordnet, die Ausgabe-Signale bewirken Prozeduraufrufe, die über entsprechende Bits des Parallelports die Motoren steuern. Dadurch kann aus einem verifizierten Steuerungsmodell direkt ein reales Steuerungsprogramm entstehen, vorerst als Prototyp für Test und Validierung.

- **Verfeinerungsanalyse für hybride Automaten.** Da insbesondere für hybride Systeme bisher in der Forschung keine effizienten Datenstrukturen und Algorithmen verfügbar sind, wäre eine automatische Verfeinerungsanalyse eine sinnvolle Ergänzung zur Repräsentation hybrider Modelle. Größere hybride Modelle könnten durch modulares Beweisen besser bewältigt werden.

Die theoretischen Grundlagen für die algorithmische Berechnung einer Simulationsrelation liegen vor, und der Einsatz der Verfeinerungsanalyse im Bereich der Realzeit-Systeme ergab vielversprechende Ergebnisse (siehe Abschnitt 6.5.5).

- **Synthese von DBM und BDD.** Die BDD-Repräsentation hat einen Nachteil gegenüber der auf Matrizen basierenden Datenstruktur: Große Konstanten in Uhrenbedingungen der Automaten wirken sich negativ auf BDD-Größe und Berechnungszeit aus. Eine Synthese von DBMs und BDDs könnte die Vorteile beider Datenstrukturen vereinen. Erste Ansätze solcher Datenstrukturen liegen vor, brachten jedoch bisher nicht den gewünschten Erfolg (siehe Abschnitt 1.2.2).

- **Benutzeroberfläche der Werkzeuge.** Ein Werkzeug zur Verifikation formaler Modelle wird in der industriellen Praxis nur eingesetzt, wenn eine graphische Benutzeroberfläche für Modellierung und Bedienung der Analysemechanismen zur Verfügung steht. Das Werkzeug darf zwar kommandozeilenorientiert ausgelegt sein, muß jedoch um eine graphische Oberfläche ergänzt werden.
  Für den praktischen Einsatz ist eine graphische Notation der Module und Automaten erforderlich, so daß Modelle durch Zeichnungen statt durch textuelle Beschreibung definiert und Analyseparameter durch Klick-Boxen komfortabel konfiguriert werden können.

- **Effiziente DDM-Bibliothek.** Die Repräsentation für die Analyse der hybriden Automaten wurde in dieser Arbeit nur prototypisch implementiert, da die Hauptaufgabe in der Entwicklung effizienter Verifikationsverfahren für Realzeit-Systeme bestand. Das Problem der Repräsentation hybrider Systeme mit geeigneten Datenstrukturen und effizienten Algorithmen wurde bisher weder in der Literatur noch in dieser Arbeit gelöst. Ansätze zur Verbesserung der vorliegenden DDM-Repräsentation wurden in Abschnitt 5.4.1.5 aufgezeigt.

- **Entwicklung von Modellen.** Im letzten Kapitel wurde angedeutet, daß Bedarf an der Untersuchung alternativer Entwicklungsprozesse für die formale Modellierung großer Systeme besteht. Bei der Konstruktion des Modells für die Fertigungsanlage war bereits deutlich zu spüren, daß Verbesserungen zur Vorgehensweise und Methodik möglich sind.
  Auch technische Prinzipien der Softwaretechnik wurden bei der Entwicklung des Modells der Fertigungsanlage eingesetzt: Zum Erzeugen ähnlicher Module wurden bestehende Komponenten durch das Muster eines Adapters bzw. Wrappers im Zusammenspiel mit dem Schablonenmechanismus des CTA-Formalismus wiederverwendet. Auch für die Protokolle zur Kommunikation der Komponenten treten Muster auf.

Fazit. Entgegen der zuweilen vertretenen Meinung, daß die Technologie des Model-Checking ihre Grenzen erreicht habe (siehe Fußnote auf Seite 183), ist diese Arbeit ein Beleg dafür, daß durch die Verknüpfung der Ergebnisse aus verschiedenen Bereichen Modelle einer neuen Größenordnung verifiziert werden können. Der Leitgedanke war die Nutzung von **Struktur**: Ein Modellierungsformalismus wurde entwickelt, durch den die in Systemen explizit vorhandene Struktur von Komponenten und deren Kommunikation in natürlicher Weise im Modell widergespiegelt werden kann. Die strukturierte Modellierung ermöglicht die Konstruktion von Modellen, die leicht verständlich, verwendbar und veränderbar sind. Darüberhinaus wird diese Struktur genutzt, um die Verifikationsaufgabe zu erleichtern. Die hier vorgestellte BDD-Repräsentation ist durch das strukturorientierte Variablenordnen effizient und ermöglicht die Erreichbarkeitsanalyse größerer Modelle. Durch die Verfeinerungsanalyse ist der Einsatz kompositioneller Beweistechniken möglich. Die modulare Modellstruktur führt beim Entwicklungsprozeß zu Verfeinerungsstrukturen, die nicht künstlich erzeugt, sondern zur Grundstruktur des Systems kompatibel sind und als Abstraktion in modularen Beweisen verwendet werden können.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ABB+01] Tobias Amnell, Gerd Behrmann, Johan Bengtsson, Pedro R. D'Argenio, Alexandre David, Ansgar Fehnker, Thomas Hune, Bertrand Jeannet, Kim G. Larsen, M. Oliver Möller, Paul Pettersson, Carsten Weise und Wang Yi. Uppaal - Now, Next, and Future. In F. Cassez, C. Jard, B. Rozoy und M. Ryan, Hrsg., *Proceedings of the 4th Summer School on Modelling and Verification of Parallel Processes (MOVEP 2000)*, LNCS 2067, Seiten 99–124. Springer-Verlag, 2001.

[ABK+97] Eugene Asarin, Marius Bozga, Alain Kerbrat, Oded Maler, Amir Pnueli und Anne Rasse. Data-Structures for the Verification of Timed Automata. In O. Maler, Hrsg., *Proceedings of the 1st International Workshop on Hybrid and Real-Time Systems (HART 1997)*, LNCS 1201, Seiten 346–360. Springer-Verlag, 1997.

[ACD90] Rajeev Alur, Costas Courcoubetis und David L. Dill. Model-Checking for Real-Time Systems. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS 1990)*, Seiten 414–425. IEEE Computer Society Press, Philadelphia (PA), 1990.

[ACD93] Rajeev Alur, Costas Courcoubetis und David L. Dill. Model-Checking in Dense Real-Time. *Information and Computation*, 104(1):2–34, Mai 1993.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis und S. Yovine. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science*, 138:3–34, 1995.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger und Psi-Hsin Ho. Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn und H. Rischel, Hrsg., *Proceedings of the Workshop on Hybrid Systems (HS 1991, HS 1992)*, LNCS 736, Seiten 209–229, 1993.

[AD90] Rajeev Alur und David L. Dill. Automata for Modelling Real-Time Systems. In M. S. Paterson, Hrsg., *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, LNCS 443, Seiten 322–335. Springer-Verlag, 1990.

[AD94] Rajeev Alur und David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AdAG+01] Rajeev Alur, Luca de Alfaro, Radu Grosu, Thomas A. Henzinger, M. Kang, Rupak Majumdar, Freddy Y. C. Mang, Christoph M. Kirsch und Bow-Yaw Wang. MOCHA: A Model Checking Tool that Exploits Design Structure. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, Seiten 835–836, 2001.

[AGH+00] Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar und Insup Lee. Modular Specification of Hybrid Systems in CHARON. In N. A. Lync und B. H. Krogh, Hrsg., *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC 2000)*, LNCS 1790, Seiten 6–9. Springer-Verlag, Berlin, 2000.

[AH96] Rajeev Alur und Thomas A. Henzinger. Reactive Modules. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, Seiten 207–218. IEEE Computer Society Press, 1996.

[AH97] Rajeev Alur und Thomas A. Henzinger. Modularity for Timed and Hybrid Systems. In A. W. Mazurkiewicz und J. Winkowski, Hrsg., *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR 1997)*, LNCS 1243, Seiten 74–88. Springer-Verlag, Berlin, 1997.

[AHM+98] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani und Serdar Tasiran. MOCHA: Modularity in Model Checking. In A. J. Hu und M. Y. Vardi, Hrsg., *Proceedings of the 10th International Conference on Computer-Aided Verification (CAV 1998)*, LNCS 1427, Seiten 521–525. Springer-Verlag, Berlin, 1998.

[AIKY95] Rajeev Alur, Alon Itai, Robert P. Kurshan und Mihalis Yannakakis. Timing Verification by Successive Approximation. *Information and Computation*, 118:142–157, 1995.

[AK96] Rajeev Alur und Robert P. Kurshan. Timing Analysis in COSPAN. In R. Alur, T. A. Henzinger und E. D. Sontag, Hrsg., *Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control (HS 1995)*, LNCS 1066, Seiten 220–231. Springer-Verlag, Berlin, 1996.

[Ake78] Sheldon B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, Juni 1978.

[Alu99] Rajeev Alur. Timed Automata. In N. Halbwachs und D. Peled, Hrsg., *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV 1999)*, LNCS 1633, Seiten 8–22. Springer-Verlag, 1999.

[AMP98] Eugene Asarin, Oded Maler und Amir Pnueli. On Discretization of Delays in Timed Automata and Digital Circuits. In R. de Simone und D. Sangiorgi, Hrsg., *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR 1998)*, LNCS 1466, Seiten 470–484. Springer-Verlag, 1998.

[ATB94] Adnan Aziz, Serdar Tasiran und Robert K. Brayton. BDD Variable Ordering for Interacting Finite State Machines. In *Proceedings of the 31st ACM/IEEE Design Automation Conference (DAC 1994)*, Seiten 283–288. ACM Press, 1994.

[Bac92] Peter Bachmann. *Mathematische Grundlagen der Informatik*. Akademie-Verlag, Berlin, 1992.

[Bal96a] Felice Balarin. Approximate Reachability Analysis of Timed Automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS 1996)*, Seiten 52–61. IEEE Computer Society Press, Los Alamitos (CA), 1996.

[Bal96b] Helmut Balzert. *Lehrbuch der Software-Technik (Band 1): Software-Entwicklung*. Spektrum Akademischer Verlag, Heidelberg, 1996.

[Bal98] Helmut Balzert. *Lehrbuch der Software-Technik (Band 2): Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum Akademischer Verlag, Heidelberg, 1998.

[BCL+94] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan und David L. Dill. Symbolic Model Checking for Sequential Circuit Verification. *IEEE Transactions on CAD*, 13(4):401–424, April 1994.

[BCM+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill und L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, Februar 1992.

[BD91] Bernard Berthomieu und Michel Diaz. Modelling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis und Sergio Yovine. Kronos: a model-checking tool for real-time systems. In A. J. Hu und M. Y. Vardi, Hrsg., *Proceedings of the 10th International Conference on Computer-Aided Verification (CAV 1998)*, LNCS 1427, Seiten 546–550. Springer-Verlag, 1998.

[Bey98] Dirk Beyer. Ein Analysewerkzeug für zeitbehaftete Automaten. Diplomarbeit, Technical University of Brandenburg, Cottbus, 1998.

[Bey01a] Dirk Beyer. Efficient Reachability Analysis and Refinement Checking of Timed Automata using BDDs. In T. Margaria und T. F. Melham, Hrsg., *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2001, Livingston)*, LNCS 2144, Seiten 86–91. Springer-Verlag, Berlin, 2001.

[Bey01b] Dirk Beyer. Improvements in BDD-based Reachability Analysis of Timed Automata. In J. N. Oliveira und P. Zave, Hrsg., *Proceedings of the 10th International Symposium of Formal Methods Europe (FME 2001, Berlin): Formal Methods for Increasing Software Productivity*, LNCS 2021, Seiten 318–343. Springer-Verlag, Berlin, 2001.

[Bey01c] Dirk Beyer. Rabbit: Verification of Real-Time Systems. In P. Pettersson und S. Yovine, Hrsg., *Proceedings of the Workshop on Real-Time Tools (RT-TOOLS 2001, Aalborg)*, Seiten 13–21, Uppsala, 2001.

[BGK+96] Johan Bengtsson, W. O. David Griffioen, Kåre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson und Wang Yi. Verification of an Audio Protocol with Bus Collision Using Uppaal. In R. Alur und T. A. Henzinger, Hrsg., *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, LNCS 1102, Seiten 244–256. Springer-Verlag, Berlin, 1996.

[BH89] Ferenc Belina und Dieter Hogrefe. The CCITT-Specification and Description Language SDL. *Computer Networks and ISDN Systems*, 16(1-2):311–341, 1989.

[BH95] Jonathan P. Bowen und Michael G. Hinchey. Seven More Myths of Formal Methods. *IEEE Software*, 12(4):34–41, Juli 1995.

[BH01] Dirk Beyer und Andy Heinig. Different Strategies for BDD-based Reachability Analysis of Timed Automata. In C. Rattray, M. Sveda und J. Rozenblit, Hrsg., *Proceedings of the 2nd IEEE/IFIP Joint Workshop on Formal Specifications of Computer-Based Systems (FSCBS 2001, Washington, D.C.)*, Seiten 89–98, Stirling, 2001.

[BLL+96] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson und Wang Yi. Uppaal – a Tool Suite for Automatic Verification of Real-Time Systems. In R. Alur, T. A. Henzinger und E. D. Sontag, Hrsg., *Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control (HS 1995)*, LNCS 1066, Seiten 232–243. Springer-Verlag, Berlin, 1996.

[BLP+99] Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise und Wang Yi. Efficient Timed Reachability Analysis Using Clock Difference Dia-

grams. In N. Halbwachs und D. Peled, Hrsg., *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV 1999)*, LNCS 1633, Seiten 341–353. Springer-Verlag, 1999.

[BLR00] Dirk Beyer, Claus Lewerentz und Heinrich Rust. Modelling and Analysing a Railroad Crossing in a Modular Way. In S. Gnesi, I. Schieferdecker und A. Rennoch, Hrsg., *Proceedings of the 5th International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS 2000, Berlin)*, Seiten 287–303, Berlin, 2000.

[BLS00] Dirk Beyer, Claus Lewerentz und Frank Simon. Flattening Inheritance Structures - OR - Getting the Right Picture of Large OO-Systems. Technical Report I-12/2000, BTU Cottbus, 2000.

[BLS01] Dirk Beyer, Claus Lewerentz und Frank Simon. Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object Oriented Systems. In R. Dumke und A. Abran, Hrsg., *Proceedings of the 10th International Workshop on Software Measurement (IWSM 2000, Berlin): New Approaches in Software Measurement*, LNCS 2006, Seiten 1–17. Springer-Verlag, Berlin, 2001.

[BMPY97] Marius Bozga, Oded Maler, Amir Pnueli und Sergio Yovine. Some Progress on the Symbolic Verification of Timed Automata. In O. Grumberg, Hrsg., *Proceedings of the 9th International Conference on Computer Aided Verification (CAV 1997)*, LNCS 1254, Seiten 179–190. Springer-Verlag, Berlin, 1997.

[BN00] Dirk Beyer und Andreas Noack. BDD-basierte Verifikation von Realzeit-Systemen. In J. Grabowski und S. Heymer, Hrsg., *Tagungsband Formale Beschreibungstechniken für verteilte Systeme (FBT 2000, Lübeck)*, Seiten 79–89. Shaker-Verlag, Aachen, 2000.

[BN01] Dirk Beyer und Andreas Noack. Efficient Verification of Timed Automata using BDDs. In S. Gnesi und U. Ultes-Nitsche, Hrsg., *Proceedings of the 6th International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS 2001, Paris)*, Seiten 95–113. INRIA, Paris, 2001.

[BR98] Dirk Beyer und Heinrich Rust. Modeling a Production Cell as a Distributed Real-Time System with Cottbus Timed Automata. In H. König und P. Langendörfer, Hrsg., *Tagungsband Formale Beschreibungstechniken für verteilte Systeme (FBT 1998, Cottbus)*, Seiten 148–159. Shaker-Verlag, Aachen, 1998.

[BR99a] Dirk Beyer und Heinrich Rust. Concepts of Cottbus Timed Automata. In K. Spies und B. Schätz, Hrsg., *Tagungsband Formale Beschreibungstechniken für verteilte Systeme (FBT 1999, München)*, Seiten 27–34. Herbert-Utz-Verlag, München, 1999.

[BR99b] Dirk Beyer und Heinrich Rust. A Formalism for Modular Modelling of Hybrid Systems. Technical Report I-10/1999, BTU Cottbus, 1999.

[BR00a] Dirk Beyer und Heinrich Rust. Modular Modelling and Verification with Cottbus Timed Automata. In C. Rattray und M. Sveda, Hrsg., *Proceedings of the IEEE/IFIP Joint Workshop on Formal Specifications of Computer-Based Systems (FSCBS 2000, Edinburgh)*, Seiten 17–24, Stirling, 2000.

[BR00b] Dirk Beyer und Heinrich Rust. A Tool for Modular Modelling and Verification of Hybrid Systems. In A. Crespo und J. Vila, Hrsg., *Proceedings of the 25th IFAC/IFIP Workshop on Real-Time Programming 2000 (WRTP 2000, Palma)*, Seiten 169–174. Elsevier Science, Oxford, 2000.

[BR01] Dirk Beyer und Heinrich Rust. Cottbus Timed Automata: Formal Definition and Semantics. In C. Rattray, M. Sveda und J. Rozenblit, Hrsg., *Proceedings of the 2nd IEEE/IFIP Joint Workshop on Formal Specifications of Computer-Based Systems (FSCBS 2001, Washington, D.C.)*, Seiten 75–87, Stirling, 2001.

[BRB90] Karl S. Brace, Richard L. Rudell und Randal E. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC 1990)*, Seiten 40–45. IEEE Computer Society Press, 1990.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transaction on Computers*, C-35(8):677–691, 1986.

[Bry92] Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BS00] Ramesh Bharadwaj und Steve Sims. Salsa: Combining Constraint Solvers with BDDs for Automatic Invariant Checking. In S. Graf und M. I. Schwartzbach, Hrsg., *Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2000)*, LNCS 1785, Seiten 378–394. Springer-Verlag, Berlin, 2000.

[Büc60] J. Richard Büchi. On a Decision Method in Restricted Second Order Arithmetic. In E. Nagel et al., Hrsg., *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, Seiten 1–11. Stanford University Press, Stanford (CA), 1960.

[BW96] Beate Bollig und Ingo Wegener. Improving the Variable Ordering of OBDDs Is NP-Complete. *IEEE Transactions on Computers*, 45(9):993–1002, September 1996.

[CCJ+01] Pankay Chauhan, Edmund M. Clarke, Somesh Jha, James H. Kukula, Helmut Veith und Dong Wang. Using Combinatorial Optimization Methods

for Quantification Scheduling. In T. Margaria und T. F. Melham, Hrsg., *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2001)*, LNCS 2144, Seiten 293–309. Springer-Verlag, Berlin, 2001.

[CE81] Edmund M. Clarke und E. Allen Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In *Proceedings of the Workshop on Logic of Programs*, LNCS 131, Seiten 52–71. Springer-Verlag, 1981.

[Čer93] Karlis Čerāns. Decidability of Bisimulation Equivalences for Parallel Timer Processes. In G. v. Bochmann und D. K. Probst, Hrsg., *Proceedings of the 4th International Conference on Computer-Aided Verification (CAV 1992)*, LNCS 663, Seiten 269–300. Springer-Verlag, 1993.

[CGP99] Edmund M. Clarke, Orna Grumberg und Doron A. Peled. *Model Checking*. MIT Press, Cambridge (MA), London, 1999.

[Che65] N. V. Chernikova. An Algorithm for Finding a General Formula for nonnegative Solutions of System of Linear Inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5:228–233, 1965.

[Che68] N. V. Chernikova. Algorithm for Discovering the Set of all Solutions of a Linear Programming Problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[CL00] Franck Cassez und Kim Larsen. The Impressive Power of Stopwatches. In C. Palamidessi, Hrsg., *Proceedings of the International Conference on Concurrency Theory (CONCUR 2000)*, LNCS 1877, Seiten 138–152. Springer-Verlag, 2000.

[DHWT92] David L. Dill, Alan J. Hu und Howard Wong-Toi. Checking for Language Inclusion Using Simulation Preorders. In K. Larsen und A. Skou, Hrsg., *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV 1991)*, LNCS 575, Seiten 255–265. Springer-Verlag, 1992.

[Dil90] David L. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, Hrsg., *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems (1989)*, LNCS 407, Seiten 197–212. Springer-Verlag, Berlin, 1990.

[DOTY96] Conrado Daws, Alfredo Olivero, Stavros Tripakis und Sergio Yovine. The Tool Kronos. In R. Alur, T. A. Henzinger und E. D. Sontag, Hrsg., *Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control (HS 1995)*, LNCS 1066, Seiten 208–219. Springer-Verlag, Berlin, 1996.

[dR98] Willem-Paul de Roever. The Need for Compositional Proof Systems: A Survey. In W.-P. de Roever, H. Langmaack und A. Pnueli, Hrsg., *Proceedings of the International Symposium on Compositionality (COMPOS 1997): The Significant Difference*, LNCS 1536, Seiten 1–22. Springer-Verlag, Berlin, 1998.

[dRLP98] Willem-Paul de Roever, Hans Langmaack und Amir Pnueli. *Proceedings of the International Symposium on Compositionality (COMPOS 1997): The Significant Difference*. LNCS 1536. Springer-Verlag, Berlin, 1998.

[DS95] Charles Donnelly und Richard Stallman. Bison: A YACC-Compatible Parser Generator, Version 1.24. Technical report, Free Software Foundation, Cambridge (MA), 1995.

[DT98] Conrado Daws und Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In B. Steffen, Hrsg., *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1998)*, LNCS 1384, Seiten 313–329. Springer-Verlag, 1998.

[DY96] Conrado Daws und Sergio Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS 1996)*. IEEE Computer Society Press, 1996.

[EMD+98] Michael Eckrich, Wilfried Mala, Werner Damm, Klaus Winkelmann, Manfred Broy und Oscar Slotosch. Korrekte Software für sicherheitskritische Systeme (KORSYS). Technical report, KORSYS-Projektgruppe, 1998.

[Far02] Gyula Farkas. Theorie der einfachen Ungleichungen. *Reine und Angewandte Mathematik*, 124:1–27, 1902.

[Föl84] Otto Föllinger. *Regelungstechnik: Eine Einführung in die Methoden und ihre Anwendung*. Hüthig-Buch-Verlag, Heidelberg, 4. Auflage, 1984.

[FP96] Komei Fukuda und Alain Prodon. Double Description Method Revisited. In M. Deza, R. Euler und I. Manoussakis, Hrsg., *Combinatorics and Computer Science*, LNCS 1120, Seiten 91–111. Springer-Verlag, 1996.

[FvDFH94] James D. Foley, Andries von Dam, Steven K. Feiner und John F. Hughes. *Einführung in die Computergraphik: Einführung, Methoden, Prinzipien*. Addison-Wesley, Bonn, 1. Auflage, 1994.

[GHJV93] Erich Gamma, Richard Helm, Ralph E. Johnson und John M. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In O. Nierstrasz, Hrsg., *Proceedings of the 7th European Conference on Object-Oriented Programming (ECOOP 1993)*, LNCS 707, Seiten 406–431. Springer-Verlag, Berlin, 1993.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading (MA), 1994.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[GPV94] Aleks Göllü, Anuj Puri und Pravin Varaiya. Discretization of timed automata. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, Seiten 957–958, 1994.

[GSB98] Radu Grosu, Thomas Stauner und Manfred Broy. A Modular Visual Model for Hybrid Systems. In A. P. Ravn und H. Rischel, Hrsg., *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 1998)*, LNCS 1486, Seiten 75–91. Springer-Verlag, 1998.

[HA72] C. Antony R. Hoare und D. C. S. Allison. Incomputability. *ACM Computing Surveys*, 4(3):169–178, September 1972.

[Hal93] Nicholas Halbwachs. Delay Analysis in Synchronous Programs. In C. Courcoubetis, Hrsg., *Proceedings of the 5th International Conference on Computer Aided Verification (CAV 1993)*, LNCS 697, Seiten 333–346. Springer-Verlag, 1993.

[Har87] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[Hen96] Thomas A. Henzinger. The Theory of Hybrid Automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, Seiten 278–292. IEEE Computer Society Press, 1996.

[Hen00] Thomas A. Henzinger. Masaccio: A Formal Model for Embedded Components. In J. v. Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses und T. Ito, Hrsg., *Proceedings of the First International IFIP Conference on Theoretical Computer Science (TCS 2000): Exploring New Frontiers of Theoretical Informatics*, LNCS 1872, Seiten 549–563. Springer-Verlag, 2000.

[HHMWT00] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar und Howard Wong-Toi. Beyond HyTech: Hybrid Systems Analysis Using Interval Numerical Methods. In N. A. Lync und B. H. Krogh, Hrsg., *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC 2000)*, LNCS 1790, Seiten 130–144. Springer-Verlag, Berlin, 2000.

[HHWT95a] Thomas A. Henzinger, Pei-Hsin Ho und Howard Wong-Toi. HyTech: The Next Generation. In *Proceedings of the 16th Annual IEEE Real-Time Systems Symposium (RTSS 1995)*, Seiten 56–65. IEEE Computer Society Press, 1995.

[HHWT95b] Thomas A. Henzinger, Pei-Hsin Ho und Howard Wong-Toi. A User Guide to HyTech. In E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria und B. Steffen, Hrsg., *Proceedings of the 1st Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1995)*, LNCS 1019, Seiten 41–71. Springer-Verlag, 1995.

[HK90] Zvi Har'El und Robert P. Kurshan. Software for Analytical Development of Communication Protocols. *AT&T Technical Journal*, 1990.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri und Pravin Varaiya. What's Decidable About Hybrid Automata. *Computer and Systems Sciences*, 57:94–124, 1998.

[HLN+90] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring und Mark B. Trakhtenbrot. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering (TSE)*, 16(4):403–414, April 1990.

[HMP92] Thomas A. Henzinger, Zohar Manna und Amir Pnueli. What Good are Digital Clocks? In W. Kuich, Hrsg., *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP 1992)*, LNCS 623, Seiten 545–558. Springer-Verlag, Berlin, 1992.

[HMP94] Thomas A. Henzinger, Zohar Manna und Amir Pnueli. Temporal Proof Methodologies for Timed Transition Systems. *Information and Computatin*, 112(2):273–337, 1994.

[HN96] David Harel und Amnon Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering Methodology (TOSEM)*, 5(4):293–333, Oktober 1996.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis und Sergio Yovine. Symbolic Model-Checking for Real-Time Systems. *Information and Computation*, 111:193–244, 1994.

[Hoa85] C. Antony R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Hemel Hempstead, 1985.

[HPR97] Nicolas Halbwachs, Yann-Eric Proy und Patrick Roumanoff. Verification of Real-Time Systems using Linear Relation Analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

[Jai94] Raj Jain. *FDDI-Handbook: High-Speed Networking with Fiber and Other Media*. Addison-Wesley, Reading (MA), 1994.

[JM87] Farnam Jahanian und Aloysius Ka-Lau Mok. A Graph-theoretic Approach for Timing Analysis and its Implementation. *IEEE Transactions on Computation*, C-36:961–975, 1987.

[JPHS91] Seh-Woong Jeong, Bernard Plessier, Gary D. Hachtel und Fabio Somenzi. Variable Ordering and Selection for FSM Traversal. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1991)*, Seiten 476–479, 1991.

[Kle56] Stephen C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies*, Seiten 3–42. Princeton University Press, Princeton (NJ), 1956.

[KLL+97] Kåre J. Kristoffersen, François Larussinie, Kim G. Larsen, Paul Pettersson und Wang Yi. A Compositional Proof of a Real-Time Mutual Exclusion Protocol. In M. Bidoit und M. Dauchet, Hrsg., *Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development (TAPSOFT 1997)*, LNCS 1214, Seiten 565–579. Springer-Verlag, Berlin, 1997.

[Knu84] Donald E. Knuth. The Complexity of Songs. *Communications of the ACM (CACM)*, 27(4):344–348, April 1984.

[Kop96] Helmut Kopka. *LaTeX: Einführung*. Addison-Wesley, Bonn, 1996.

[Koy92] Ron Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. LNCS 651. Springer-Verlag, Berlin, Heidelberg, 1992.

[LA90] Nancy A. Lynch und Hagit Attiya. Using Mappings to Prove Timing Properties. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC 1990)*, Seiten 265–280, 1990.

[Lam83a] Leslie Lamport. Specifying Concurrent Program Modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.

[Lam83b] Leslie Lamport. What Good is Temporal Logic? In R. E. A. Mason, Hrsg., *Proceedings of the 9th IFIP World Computer Congress on Information Processing (1983)*, Seiten 657–668. North Holland, 1983.

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

[Lea92] Doug Lea. *User's Guide to the GNU C++ Library, Version 2.0*. Free Software Foundation, Cambridge (MA), 1992.

[Lee59] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

[Lev95] Nancy G. Leveson. *Safeware*. Addison Wesley, Reading (MA), 1995.

[LL95] Claus Lewerentz und Thomas Lindner, Hrsg. *Formal Development of Reactive Systems*. LNCS 891. Springer-Verlag, Berlin, Heidelberg, 1995.

[LLKS85] Eugene L. Lawler, Jan K. Lenstra, A. H. G. Rinnooy Kan und David B. Shmoys, Hrsg. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.

[LM96] Annette Lötzbeyer und Reinhard Mühlfeld. Aufgabenstellung einer flexiblen Fertigungszelle mit Realzeiteigenschaften. Technical report, Forschungszentrum Informatik, Karlsruhe, 1996.

[Löt96] Annette Lötzbeyer. Spezifikation, Modellierung und Verifikation von Realzeitsystemen. Technical Report 6/96, Forschungszentrum Informatik, Karlsruhe, 1996.

[LPWY99] Kim G. Larsen, Justin Pearson, Carsten Weise und Wang Yi. Clock Difference Diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[LPY95a] Kim G. Larsen, Paul Pettersson und Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS 1995)*, Seiten 76–89. IEEE Computer Society Press, 1995.

[LPY95b] Kim G. Larsen, Paul Pettersson und Wang Yi. Model-Checking for Real-Time Systems. In H. Reichel, Hrsg., *Proceedings of the 10th International Symposium on Fundamentals of Computation Theory (FCT 1995)*, LNCS 965, Seiten 62–88. Springer-Verlag, Berlin, 1995.

[LPY97] Kim G. Larsen, Paul Pettersson und Wang Yi. Uppaal in a Nutshell. *Software Tools for Technology Transfer*, 1(1-2):134–152, Oktober 1997.

[LSS87] Jacques Loeckx, Kurt Sieber und Ryan D. Stansifer. *The Foundations of Program Verification*. Wiley, Teubner, Chichester, Stuttgart, 2. Auflage, 1987.

[LSVW96] Nancy A. Lynch, Roberto Segala, Frits W. Vaandrager und Henri B. Weinberg. Hybrid I/O Automata. In R. Alur, T. A. Henzinger und E. D. Sontag, Hrsg., *Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III (HS 1995)*, LNCS 1066, Seiten 496–510. Springer-Verlag, Berlin, 1996.

[LT87] Nancy A. Lynch und Mark R. Tuttle. Hierarchical Correctness Proofs for Distributed Algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC 1987)*, Seiten 137–151. ACM Press, 1987.

[LT93] Nancy G. Leveson und Clark S. Turner. Investigation of the Therac-25 Accidents. *IEEE Computer*, 26(7):18–41, September 1993.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer, 1993. PhD Thesis, School of Computer Science, Carnegie Mellon University.

[Mea55] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal 34*, 5(34):1045–1079, 1955.

[Mey88] Bertrand Meyer. *Object-oriented Software Construction*. Prentice Hall, London, 1988.

[MF76] Philip M. Merlin und David J. Farber. Recoverability of Communication Protocols – Implications of a Theoretical Study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.

[MHS00] In-Ho Moon, Gary D. Hachtel und Fabio Somenzi. Border-Block Triangular Form and Conjunction Schedule in Image Computation. In Warren A. Hunt Jr. und Steven D. Johnson, Hrsg., *Proceedings of the 3rd International Conference on Formal Methods in Computer-Aided Design (FMCAD 2000)*, LNCS 1954, Seiten 73–90. Springer-Verlag, 2000.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, Hemel Hempstead, 1989.

[Min65] Marvin L. Minski. Matter, Mind, and Models. In W. A. Kalenich, Hrsg., *Proceedings of the IFIP Congress: Information Processing, Vol. 1*, Seiten 45–49. Spartan Books, 1965.

[MLAH99] Jesper Møller, Jakob Lichtenberg, Henrik R. Andersen und Henrik Hulgaard. Difference Decision Diagrams. In J. Flum und M. Rodriguez-Artalejo, Hrsg., *Proceedings of the 13th International Workshop on Computer Science Logic (CSL 1999)*, LNCS 1683, Seiten 111–125. Springer-Verlag, 1999.

[MN99] Eduard Moser und Wolfgang Nebel. Case Study: System Model of Crane and Embedded Control. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 1999)*, Seiten 721–723. IEEE Computer Society Press, 1999.

[Moo56] Edward F. Moore. Gedanken experiments on sequential machines. In C. E. Shannon und J. McCarthy, Hrsg., *Annals of Mathematics Studies*

(34), Automata Studies, Seiten 129–153. Princeton University Press, Princeton (NJ), 1956.

[MP43] Warren S. McCulloch und Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bull. Math. Biophysics*, 5:115–133, 1943.

[MP93] Zohar Manna und Amir Pnueli. Models for Reactivity. *Acta Informatica*, 30(7):609–678, 1993.

[MRTT53] Theodore S. Motzkin, H. Raiffa, G. L. Thompson und R. M. Thrall. The Double Description Method. In H. W. Kuhn und A. W. Tucker, Hrsg., *Contributions to Theory of Games, Vol. 2*. Princeton University Press, Princeton (NJ), 1953.

[MT90] Faron Moller und Chris M. N. Tofts. A Temporal Calculus of Communicating Processes. In J. C. M. Baeten und J. W. Klop, Hrsg., *Proceedings of the International Conference on Concurrency Theory (CONCUR 1990): Unification and Extension*, LNCS 458, Seiten 401–415. Springer-Verlag, Berlin, 1990.

[Noa99] Andreas Noack. Ein ZBDD-Paket für effizientes Model Checking von Petrinetzen. Technical report, Technical University of Brandenburg, Cottbus, 1999.

[Noa00] Andreas Noack. BDD-basierte Verification von Echtzeitsystemen. Diplomarbeit, Technical University of Brandenburg, Cottbus, 2000.

[Ost90] Jonathan S. Ostroff. *Temporal Logic of Real-Time Systems*. Research Studies Press, 1990.

[Pag91] Bernd Page. *Diskrete Simulation: Eine Einführung mit Modula-2*. Springer-Verlag, Berlin, Heidelberg, 1991.

[Par72] David L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM (CACM)*, 15(12):1053–1058, Dezember 1972.

[Pax95] Vern Paxson. Flex: A Fast Scanner Generator, Version 2.5. Technical report, University of California, Berkeley, 1995.

[PCW84] David L. Parnas, Paul C. Clements und David M. Weiss. The Modular Structure of Complex Systems. In *Proceedings of the 7th International Conference on Software Engineering (ICSE 1984)*, Seiten 408–419, 1984.

[Pet62] Carl A. Petri. Kommunikation mit Automaten. Technical report, Schriften des Instituts für instumentelle Mathematik, Bonn, 1962.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, Seiten 46–57. IEEE Computer Society Press, 1977.

[Pop91] Louchka Popova. On Time Petri Nets. *Information Processing and Cybernetics*, 27(4):227–244, 1991.

[PV94] Anuj Puri und Pravin Varaiya. Decidability of Hybrid Systems with Rectangular Differential Inclusions. In D. L. Dill, Hrsg., *Proceedings of the 6th International Conference on Computer Aided Verification (CAV 1994)*, LNCS 818, Seiten 95–104. Springer-Verlag, 1994.

[PZH97] Louchka Popova-Zeugmann und Monika Heiner. Worst-case Analysis of Concurrent Systems with Duration Interval Petri Nets. In E. Schnieder und D. Abel, Hrsg., *Tagungsband Entwurf komplexer Automatisierungssysteme (EKA 1997)*, Seiten 162–179, 1997.

[RAB+95] Rajeev K. Ranjan, Adnan Aziz, Robert K. Brayton, Carl Pixley und Bernhard Plessier. Efficient BDD Algorithms for Synthesizing and Verifying Finite State Machines. In *Workshop Notes of the IEEE/ACM International Workshop on Logic Synthesis (IWLS 1995)*, 1995.

[Ram74] Chander Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. Dissertation, Massachusetts Institute of Technology, Report MAC TR-120, Cambridge, Massachusetts, 1974.

[RR88] George M. Reed und A. W. Roscoe. A Timed Model for Communicating Sequential Processes. *Theoretical Computer Science*, 58:249–261, 1988.

[Rud93] Richard Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1993)*, Seiten 42–47. IEEE Computer Society Press, Los Alamitos (CA), 1993.

[Rus94] Heinrich Rust. *Zuverlässigkeit und Verantwortung*. Vieweg, Braunschweig, Wiesbaden, 1994.

[Rus99] Heinrich Rust. Modeling a Production Cell Component as a Hybrid Automaton: A Case Study. Technical Report I-2/1999, BTU Cottbus, 1999.

[Sha38] Claude E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. *Transactions of the AIEE*, 57:713–723, 1938.

[SL95] Alexander Stepanov und Meng Lee. *The Standard Template Library*. Hewlett-Packard Laboratories, Palo Alto, 1995.

[SM95] Richard M. Stallman und Roland McGrath. GNU Make: A Program for Directing Recompilation, Version 3.74. Technical report, Free Software Foundation, Cambridge (MA), 1995.

[SM97] Oscar Slotosch und Wilfried Mala. KORSYS-Prozeßmodell mit Kritikalen Aspekten. Technical report, KORSYS-Projektgruppe, 1997.

[SR94] Amit Shah und G. Ramakrishnan. *FDDI – A High-Speed Network*. Prentice-Hall, Englewood Cliffs (NJ), 1994.

[Sta95a] Richard Stallman. GNU Emacs Manual, Version 19.30. Technical report, Free Software Foundation, Cambridge (MA), 1995.

[Sta95b] Richard M. Stallman. Using and Porting GNU CC, Version 2.7.2. Technical report, Free Software Foundation, Cambridge (MA), 1995.

[Str98] Bjarne Stroustrup. *Die C++-Programmiersprache*. Addison-Wesley-Longman, Bonn, 1998.

[Tan89] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs (NJ), 2. Auflage, 1989.

[Tho90] Wolfgang Thomas. Automata on Infinite Objects. In J. van Leeuwen, Hrsg., *Handbook of Theoretical Computer Science, Vol. B*, Seiten 165–191. Elsevier, Amsterdam, 1990.

[THY93] Seiichiro Tani, Kiyoharo Hamaguchi und Shuzo Yajima. The Complexity of the Optimal Variable Ordering Problem of A Shared Binary Decision Diagram. Technical Report 93-6, Department of Information Science, Faculty of Science, University of Tokyo, Tokyo (Japan), 1993.

[TSL+90] Hervé J. Touati, Hamid Savoj, Bill Lin, Robert K. Brayton und Alberto Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1990)*, Seiten 130–133. IEEE Computer Society Press, Los Alamitos (CA), 1990.

[Tur37] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1937.

[vB69] Ludwig von Bertalanffy. *General System Theory: Foundations, Development, Applications*. George Braziller, New York, 1969.

[Wan00] Farn Wang. Efficient Data Structure for Fully Symbolic Verification of Real-Time Software Systems. In S. Graf und M. I. Schwartzbach, Hrsg., *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, LNCS 1785, Seiten 157–171. Springer-Verlag, 2000.

[Wan01] Farn Wang. RED: Model-Checker for Timed Automata with Clock-Restriction Diagram. In P. Pettersson und S. Yovine, Hrsg., *Proceedings of the Workshop on Real-Time Tools (RT-TOOLS 2001)*, Seiten 35–46, Uppsala, 2001.

[Wil93] Doran K. Wilde. A Library For Doing Polyhedral Operations. Technical Report 785, Institut de Recherche en Informatique et Systèmes Al'eatoires, Rennes, 1993.

[WL97] Carsten Weise und Dirk Lenzkes. Efficient Scaling-Invariant Checking of Timed Bisimulation. In R. Reischuk und M. Morvan, Hrsg., *Proceedings of the 14th Annual Symposion on Theoretical Aspects of Computer Science (STACS 1997)*, LNCS 1200, Seiten 177–188. Springer-Verlag, 1997.

[WT94] Howard Wong-Toi. *Symbolic approximations for verifying real-time systems*. PhD thesis, Department of Computer Science, Stanford University, CA, 1994. Technical report STAN-CS-TR-95-1546.

[YBO+98] Bwolen Yang, Randal E. Bryant, David R. O'Hallaron, Armin Biere, Olivier Coudert, Geert Janssen, Rajeev K. Ranjan und Fabio Somenzi. A Performance Study of BDD-Based Model Checking. In G. Gopalakrishnan und P. J. Windley, Hrsg., *Proceedings of the 2nd International Conference on Formal Methods in Computer-Aided Design (FMCAD 1998)*, LNCS 1522, Seiten 255–289. Springer-Verlag, 1998.

[Yov97] Sergio Yovine. Kronos: A Verification Tool for Real-Time Systems. *Software Tools for Technology Transfer*, 1(1-2):123–133, Oktober 1997.

[Yov98] Sergio Yovine. Model Checking Timed Automata. In G. Rozenberg und F. W. Vaandrager, Hrsg., *Lectures on Embedded Systems*, LNCS 1494, Seiten 114–152. Springer-Verlag, 1998.

[Zie95] Günter M. Ziegler. *Lectures on Polytopes*. Springer-Verlag, New York, 1995.

Index

Lebenslauf

Persönliche Daten

Name:	Dirk Thilo Beyer
Geburtsdatum:	15. Januar 1972
Geburtsort:	Finsterwalde
Anschrift:	03238 Gorden-Staupitz
	Staupitz, Feldweg 1
Staatsbürgerschaft:	Deutschland
Familienstand:	verheiratet, zwei Kinder
Religion:	evangelische Kirche

Ausbildung

Sep. 1978 - Aug. 1981	Polytechnische Oberschule
	Goethe-Oberschule Hohenleipisch
Sep. 1981 - Juni 1988	Polytechnische Oberschule
	Grube-Oberschule Staupitz
	Abschluß mittlere Reife ”sehr gut”
Sep. 1988 - Juli 1990	Berufsausbildung Elektromonteur
	VEB Braunkohlenveredlung Lauchhammer
	Abschluß Facharbeiter ”sehr gut”
	Bestenförderung ab 1989
Sep. 1991 - März 1993	Ausbildung Techniker
	Informatik- und Bildungszentrum Cottbus
	Abschluß IT-Support-Fachmann ”sehr gut”
Sep. 1991 - Juni 1993	Abendgymnasium
	Kreisvolkshochschule Finsterwalde
	Abschluß Abitur ”ausgezeichnet”
Okt. 1994 - Juni 1998	Universitätsstudium Informatik
	Brandenburgische Technische Universität Cottbus
	Abschluß Diplom ”ausgezeichnet”
	Universitätspreis 1998 für Diplomarbeit
Mai 2002	Abgabe Dissertation
	”Formale Verifikation von Realzeit-Systemen
	mittels Cottbus Timed Automata”
Nov. 2002	Mündliche Doktorprüfung
	Abschluß Dr. rer. nat. ”magna cum laude”

Berufliche Tätigkeiten

Juli 1990 - Aug. 1990	Rangierleiter Anschlußbahn Braunkohlenveredlung Lauchhammer GmbH
Mai 1993 - Sep. 1994	Techniker und Programmierer hard & soft systems GmbH Finsterwalde
Nov. 1994 - Juni 1998	Studentischer Mitarbeiter Brandenburgische Technische Universität Cottbus
Juli 1998 - Okt. 1998	Softwareingenieur Siemens AG, Business Services Dresden, Abt. Großprojekte
Nov. 1998 - heute	Wissenschaftlicher Mitarbeiter Lehrstuhl für Software-Systemtechnik Brandenburgische Technische Universität Cottbus

Sonstiges

Sep. 1990 - Aug. 1991	Zivildienstleistender Gemeinde Staupitz
Apr. 1996 - Apr. 1997	Mitglied Fachschaftsrat Studiengang Informatik Brandenburgische Technische Universität Cottbus
Apr. 1997 - Juni 1998	Vorsitzender Fachschaftsrat Studiengang Informatik Brandenburgische Technische Universität Cottbus
Dez. 1998 - Dez. 2000	Mitglied Fakultätsrat Fakultät 1 Brandenburgische Technische Universität Cottbus
März 1999 - heute	Mitglied Gemeindevertretung Gemeinde Staupitz