TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern - Fallstudie -

Johann Schumann und Max Breitling

TUM-I9904
SFB-Bericht Nr. 342/02/99 A
Januar 99

TUM–INFO–01-I9904-80/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Formalisierung und Beweis einer Verfeinerung aus Focus mit automatischen Theorembeweisern*
- Fallstudie -

Johann Schumann und Max Breitling

Institut für Informatik
Technische Universität München
D-80290 München

E-mail: {schumann|breitlin}@informatik.tu-muenchen.de

18. Januar 1999

Zusammenfassung

Das Ziel dieser Arbeit ist eine Evaluierung inwieweit der Theorembeweiser Setheo an das Spezifikationswerkzeug AutoFocus zur weitgehend automatischen Durchführung von Verifikationsaufgaben angeschlossen werden kann.

Dazu wurde ein zweielementiger Puffer sowohl abstrakt durch eine einzelne Komponente spezifiziert, als auch als verteiltes System, bestehend aus zwei einelementigen Puffern und einer Kontrolleinheit. Aus den graphisch erstellten Spezifikationen wurden formale Darstellungen erzeugt, mit denen der Beweis der Verfeinerungsrelation zwischen den beiden Spezifikationen mit Hilfe der automatischen Beweiser Setheo und SMV durchgeführt werden konnte.

*Diese Arbeit ist eine Kooperation der Teilprojekte A5 und A6 des Sonderforschungsbereiches 342 ”Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen“, finanziell unterstützt durch die DFG.

Inhaltsverzeichnis

1 Einleitung

Formale Methoden liefern die Voraussetzung zur Entwicklung von komplexen Systemen, an die hohe Sicherheits- und Qualitätsanforderungen gestellt werden. Durch eine formale Basis ist es möglich, Spezifikationen mit einer klaren Semantik zu formulieren, und einen Nachweis der Gültigkeit von Systemeigenschaften zu erbringen. Allerdings können in den meisten Fällen nur erfahrene Anwender derartige Nachweise von Hand und mit großem Aufwand durchführen.

Ein Ziel ist es daher, anwendernähere Beschreibungs- und Spezifikationstechniken anzubieten und den formalen Anteil möglichst im Hintergrund zu halten. Der explizite Umgang mit mathematischen Formalismen sollte nach Möglichkeit vermieden werden können und stattdessen mit graphischen Beschreibungstechniken gearbeitet werden. Das auf der Methodik FOCUS basierende Werkzeug AUTOFOCUS verfolgt diesen Ansatz.

Ein langfristiges Ziel ist es, eine Beweisunterstützung für AUTOFOCUS anzubieten, die idealerweise das Erstellen von Systemspezifikationen unter Verwendung graphischer und auch textueller Editoren ermöglicht, und entstehende Verifikationaufgaben 'per Knopfdruck' an einen angebundenen automatischen Beweiser übergibt.

In diesem Papier wird exemplarisch ein Beispiel einer Verifikation durchgeführt, um die Machbarkeit einer Koppelung von AUTOFOCUS mit einem automatischen Inferenzsystem zu erproben. Dazu wird die Korrektheit eines Verfeinerungsschrittes eines zweielementigen Puffers unter Verwendung des automatischen Theorembeweisers SETHEO und des Model Checkers SMV nachgewiesen.

In Abschnitt 2 findet sich eine kurze Einführung in FOCUS und AUTOFOCUS. Abschnitt 3 beinhaltet eine Beschreibung und Formalisierung der Aufgabenstellung mit den Spezifikationen aus AUTOFOCUS. Die Erstellung des automatischen Beweises wird in Abschnitt 4 beschrieben. Erfahrungen und offene Fragestellungen werden schließlich in Abschnitt 5 zusammengefaßt. Im Anhang werden die ausführlichen formalen Spezifikationen angegeben.

Diese Arbeit entstand als Kooperation der Teilprojekte A5 und A6 des SFB 342. Teilprojekt A6 verfolgt mit dieser Arbeit das Ziel einer verbessserten Unterstützung der Verifikation verteilter Systeme mit AUTOFOCUS, während Teilprojekt A5 die Leistungsfähigkeit des automatischen Beweisers SETHEO in diesem speziellen Umfeld erproben möchte.

2 FOCUS und AUTOFOCUS

FOCUS ([BS98, BDD+93]) ist eine am Lehrstuhl von Prof. M. Broy entwickelte Methodik zur formalen Entwicklung verteilter, reaktiver Systeme. Im Sinne von FOCUS wird ein verteiltes System als Netzwerk von interagierenden Komponenten modelliert, deren Zusammenspiel durch den Austausch von Nachrichten über Verbindungskanäle erfolgt. Mit den angebotenen Beschreibungstechniken können Spezifikationen von Systemen auf verschiedenen Abstraktionsebenen erstellt werden. In einer Top-Down-Entwicklung eines Systems wird zunächst eine abstrakte Spezifikation erstellt, die die Basis für die weitere Systementwicklung darstellt. Jeder Entwicklungsschritt führt durch Entwurfsentscheidungen und Strukturierung des Systems in Unterkomponenten näher zu einer Beschreibung der intendierten Implementierung. Dabei muß für eine konsequente formale Entwicklung die Korrektheit jedes Verfeinerungsschrittes nachgewiesen werden.

Mit AutoFocus ([HSS96] steht ein Werkzeugprototyp zur Verfügung, der die Entwicklung von Systemen im Sinne von Focus unterstützt. Über Editoren können u.a. sogenannte Systemstrukturdiagramme (SSD) und Zustandsübergangsdiagramme (STD) erstellt, Konsistenzen zwischen den Dokumenten überprüft und der Ablauf von Systemen simuliert werden. Eine geplante Erweiterung, zu der hiermit Vorarbeit geleistet werden soll, ist die Integration von Beweiswerkzeugen zum Nachweis von Verfeinerungsbeziehungen zwischen Systemen auf verschiedenen Abstraktionsebenen und dem Nachweis von eventuellen weiteren Eigenschaften verteilter Systeme.

Systemstrukturdiagramme (wie beispielsweise in Abbildung 3) beschreiben die statische Struktur eines Systems durch die Angabe aller Komponenten und ihrer Verbindungsstruktur. Das Verhalten von Komponenten wird durch *Zustandsübergangsdiagramme* dargestellt (z.B. Abbildung 2). Darin wird ein Übergang vom Zustand s zu s' in der Form

$$\text{s} \xrightarrow{\text{PRE : I?x : O!y : POST}} \text{s'}$$

dargestellt. In PRE kann eine Vorbedingung über lokale Variable und eingehende Nachrichten formuliert werden, die erfüllt sein muß, damit der Übergang stattfinden darf. Mit $POST$ wird die Nachbedingung formuliert, die die ausgegebenen Nachrichten und die neuen Werte der lokalen Variablen nach der Transition zu erfüllen haben. Die Ausdrücke $I?x$ und $O!y$ beschreiben das Lesen eines Wertes vom Kanal I zusammen mit seiner Zuweisung an die Variable x bzw. das Schreiben des Wertes von y auf den Kanal O.

Focus bietet eine Reihe von Spezifikationstechniken, Verfeinerungsbegriffen und Konzepten zur Zeitmodellierung, auf die hier aber nicht eingegangen werden soll. Auch die formale Basis der *stromverarbeitenden Funktionen* wird nicht ausführlich erläutert, sondern es wird nur auf der Ebene der Zustandsautomaten im synchronen Zeitmodell (d.h. mit genau einer Nachricht pro Zeiteinheit) argumentiert. Für eine weiterführende Beschreibung von Focus und AutoFocus sei auf die Literatur verwiesen, die in [Foc, Aut] zusammenfassend aufgeführt wird.

3 Aufgabenstellung und Spezifikationen

Für diese Fallstudie wurde als Verifikationsaufgabe ein Verfeinerungsschritt bei der exemplarischen Entwicklung eines *zweielementigen Puffers* gewählt. Dieser ist in der Lage, zwei Werte vom Typ Nat in FIFO-Weise zu speichern. Der Puffer erhält von seiner Umgebung über einen einzigen Eingabekanal sowohl zu speichernde Werte vom Typ Nat als auch Anforderungen (repräsentiert durch die Nachricht Req), gespeicherte Werte über seinen Ausgabekanal wieder auszugeben. Das Speichern von Werten in dem Puffer wird nicht quittiert. Wird der Puffer aufgefordert, ein neues Datum zu speichern, obwohl bereits zwei Werte gepuffert sind, so soll diese Aufforderung vernachlässigt werden. Ebenso sind Req-Nachrichten zu ignorieren, falls der Puffer leer ist. Fehlermeldungen werden in diesen beiden Fällen nicht ausgegeben.

Zunächst wird der Puffer als eine Komponente spezifiziert, die der verbalen Beschreibung möglichst genau entspricht. Diese Spezifkation (genannt A) übernimmt in dieser Fallstudie die Rolle einer *abstrakten Spezifikation*. In einer zweiten Spezifikation wird der zweielementige

Puffer als ein verteiltes System (B) modelliert, das aus zwei einelementigen Puffern und einer Kontrolleinheit zusammengesetzt ist. Als eine für FOCUS typische Beweisaufgabe ist dann zu zeigen, daß B tatsächlich eine Implementierung von A darstellt, notiert als $A \rightsquigarrow B$.

Im folgenden werden nun die beiden Spezifikation und die formalen Grundlagen näher beschrieben.

3.1 Spezifikation System A

Die abstrakte Spezifikation des zweielementigen Puffers umfaßt die Definition der syntaktischen und semantischen Schnittstelle. Die syntaktische Schnittstelle ist durch das einfache SSD in Abbildung 1 gegeben. Es werden Nachrichten vom Typ $\mathsf{Message} = \mathsf{Nat} \cup \{\textsc{Req}\}$ vom Eingabekanal I gelesen und natürliche Zahlen auf dem Ausgabekanal O ausgegeben.

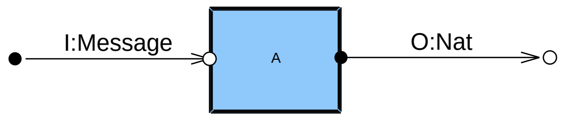

Abbildung 1: SSD für A

Das Verhalten des Puffers kann mit den in AUTOFOCUS angebotenen Techniken graphisch durch ein STD beschrieben werden, das in Abbildung 2 angegeben ist. Der Zustand *Leer* ist der Startzustand. Wird auf dem Eingabekanal eine positive Zahl d empfangen, so wird diese in $q1$ gespeichert und in den Zustand *Eins* übergegangen. Wird nun die Nachricht REQ empfangen, so wird die Ausgabe des in $q1$ gespeicherten Wertes initiiert, indem die Zustände *Wait1* und *Wait2* durchlaufen werden und schließlich auf dem Kanal O der Wert ausgegeben wird. Jeder mit *empty label* beschriftete Übergang verbraucht dabei eine Zeiteinheit, ohne an der Ein- und Ausgabe und den Daten Veränderungen durchzuführen. Diese Zwischenzustände wurden eingeführt, um die Reaktion auf eine Anfrage zu verzögern. Damit wird A bewußt so verlangsamt, daß es bezüglich der Zeit im Gleichtakt mit System B läuft. Diese Modifikation ist selbstverständlich nicht natürlich, und wurde nur durchgeführt, um die Komplexität der Beweisaufgabe zu reduzieren. Ein systematischer Umgang mit derartigen Modifikationen des zeitlichen Verhaltens ist weiter zu untersuchen (siehe Abschnitt 5). Im Zustand *Eins* kann ein weiteres Datum empfangen werden, das dann in $q2$ gespeichert wird. Sind zwei Werte gespeichert, befindet sich das System im Zustand *Zwei*. Bei einer Anfrage durch REQ wird dann der zuerst gespeicherte Wert $q1$ ausgegeben, und der Wert in $q2$ „rutscht" durch die Nachbedingung $q1' = q2$ auf $q1$ vor.

Der Zustand $s \in \mathsf{State}_A$ besteht aus einem Tripel, das den Kontrollzustand und die aktuellen Werte für $q1$ und $q2$ enthält. Der bezüglich des Datenanteils nichtdeterministische Initialzustand hat die Form $(Leer, q1, q2)$ mit beliebigen $q1, q2 \in \mathsf{Nat}$.

Die Transitionsfunktion Δ_A vom Typ $\mathsf{Message} \times \mathsf{State}_A \rightarrow \mathcal{P}(\mathsf{Nat} \times \mathsf{State}_A)$ gibt für einen Zustand und eine empfangene Nachricht die Menge der möglichen Ausgaben und Nachfolgezustände an. Sie ist durch das STD eindeutig definiert. Eine formale Darstellung ist automatisch aus der graphischen Repräsentation generierbar und ist in Anhang A explizit angegeben.

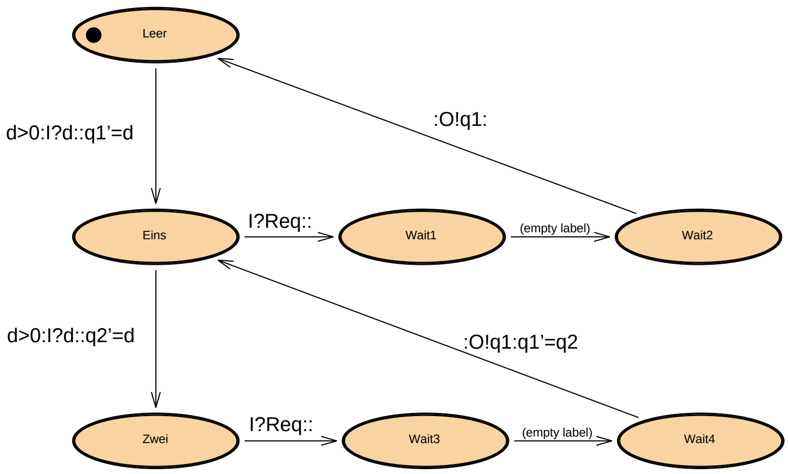

Abbildung 2: STD für A

3.2 Spezifikation System B

Die Systemstruktur des Systems B ist durch Abbildung 3 beschrieben. Die Schnittstelle zur Umgebung besteht wie bei A aus einem Eingabekanal I und einem Ausgabekanal O. An die zentrale Komponente $Controller$ sind zwei einelementige Puffer $B1$ und $B2$ über die internen Kanäle $X1, Y1, X2$ und $Y2$ verbunden.

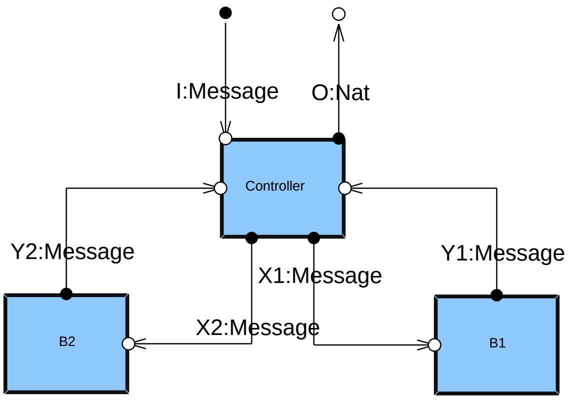

Abbildung 3: SSD für B

Die Funktionsweise der einelementigen Puffer $B1$ und $B2$ ist sehr einfach, und durch Abbildung 4 definiert: Wird ein zu speichernder Wert empfangen, so wird dieser in einer lokalen Variablen gespeichert; wird die Nachricht Req empfangen, wird der aktuell gespeicherte Wert ausgegeben. Das Verhalten des Controllers ist durch Abbildung 5 beschrieben. Wird ein Wert auf dem Eingabekanal empfangen, so wird dieser an Puffer $B1$ bzw. $B2$ weitergesendet. Wird ein gespeicherter Wert von der Umgebung angefordert, so wird diese Anforderung an $B1$ weitergegeben. Sind zwei gültige Werte in den Puffern enthalten, so wird auch an $B2$ ein Req geschickt, um den darin enthaltenen Wert nach $B1$ umspeichern zu können.

Da der Controller keinen Datenzustand und die Puffer keinen relevanten Kontrollzustand besitzen, ergibt sich der Gesamtzustand von B aus dem Kontrollzustand des Controllers, den Datenzuständen von $B1$ und $B2$ und den Nachrichten auf den internen Kanälen. Die Transitionsfunktion Δ_B ist eine Kombination der Transitionsfunktionen der drei Komponenten und läßt sich wieder systematisch aus den STDs der Komponenten generieren. Ein Zustandsübergang von B entspricht dabei einem gekoppelten Übergang der drei Komponenten. Die detaillierte Formalisierung ist in Anhang B zu finden.

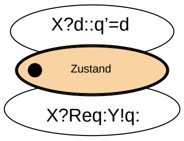

Abbildung 4: STD für $B1$ und $B2$

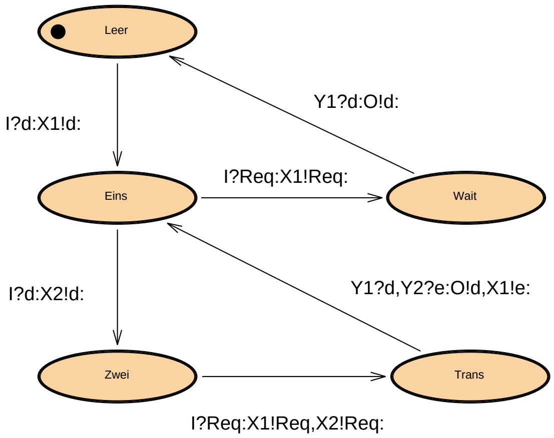

Abbildung 5: STD für den Controller aus B

3.3 Formale Grundlagen

In diesem Abschnitt werden einige grundlegende Definitionen angegeben, die in den nächsten Abschnitten benötigt werden.

Der Nachrichtenfluß zwischen Komponenten wird durch *Ströme* modelliert. Ein Strom $xs \in X^{\underline{\omega}}$ ist eine Sequenz von Elementen aus $X \cup \{\surd\}$. Dabei symbolisiert $\surd$ die Abwesenheit einer Nachricht. $\langle\rangle$ bezeichnet den leeren Strom. Der Ausdruck $xs \frown x$ liefert den Strom, der durch das Anfügen der Nachricht x an den Strom xs entsteht. Damit gelten die beiden später benötigten Eigenschaften

$$xs \frown x \neq \langle\rangle \qquad\qquad xs \frown x = ys \frown y \;\rightarrow\; xs = ys \wedge x = y \tag{1}$$

Ein nichtdeterministischer *Automat* $A = (I, O, Z, Z^0, D)$ besteht aus einem Eingabealphabet I, einem Ausgabealphabet O, einer Zustandsmenge Z, einer nichtleeren Menge von Startzuständen Z^0 mit $\emptyset \neq Z^0 \subseteq Z$, und einer Transitionsrelation $D \subseteq Z \times I \times O \times Z$. Kann der Automat im Zustand z durch Lesen der Nachricht i und Ausgabe von o in Zustand z' übergehen, so gilt $D[z, i, o, z']$. Ist für einen Automaten die Transitionsfunktion $\Delta : I \times Z \rightarrow \mathcal{P}(O \times Z)$

gegeben, so ergibt sich daraus D durch[1]

$$D[z,i,o,z'] \;\leftrightarrow\; \begin{array}{ll} & (o,z') \in \Delta(i,z) \\ \vee & \Delta(i,z) = \emptyset \wedge z = z' \wedge o = \surd \end{array} \tag{2}$$

D heißt deterministisch, wenn gilt

$$D[z,x,y_1,z_1'] \;\wedge\; D_A[z,x,y_2,z_2'] \;\rightarrow\; y_1 = y_2 \wedge z_1' = z_2' \tag{3}$$

Die Hülle $S[z, is, os, z'] \subseteq Z \times I^{\underline{\omega}} \times O^{\underline{\omega}} \times Z$ der Transitionsrelation D gibt an, daß vom Zustand s durch Lesen des Eingabestromes is und Ausgabe von os der Zustand z' erreicht werden kann:

$$S[z,is,os,z'] \;\leftrightarrow\; \begin{array}{ll} & is = \langle\rangle \;\wedge\; os = \langle\rangle \;\wedge\; z = z' \\ \vee & \exists z'',\, x,\, i',\, y,\, o' \,\cdot\;\; is = i' {}^\frown x \;\wedge\; os = o' {}^\frown y \\ & \quad\wedge\; S[z,i',o',z''] \;\wedge\; D[z'',x,y,z'] \end{array} \tag{4}$$

Die Ein-/Ausgaberelation $T \subseteq I^{\underline{\omega}} \times O^{\underline{\omega}}$ eines Automaten wird definiert durch

$$T[is,os] \;\leftrightarrow\; \exists z,z'.\; z \in Z^0 \wedge S[z,is,os,z'] \tag{5}$$

3.4 Die Beweisverpflichtung

Die *Verfeinerungsrelation* zwischen zwei Automaten A und B gilt, wenn jedes Paar von Ein-/Ausgabeströmen (is, os), das ein Verhalten von B repräsentiert, auch ein Verhalten von A ist. Dies wird mit Hilfe der vorangegangenen Definitionen formalisiert durch[2]

$$A \rightsquigarrow B \;\leftrightarrow\; \forall is, os \cdot\; T_B[is,os] \;\rightarrow\; T_A[is,os] \tag{6}$$

Diese Aussage wird in den nächsten Abschnitten bewiesen. Dabei wird das folgende Lemma verwendet, das besagt, daß zu jedem Schritt von B ein entsprechender Schritt von A existiert:

$$\begin{array}{l} w \in Z_B^0 \wedge S_B[w,is,os,w''] \wedge D_B[w'',x,y,w'] \wedge v \in Z_A^0 \wedge S_A[v,is,os,v''] \\ \quad\rightarrow\; \exists v' \cdot\; D_A[v'',x,y,v'] \end{array} \tag{7}$$

Der Beweis dieses Lemmas wird unter Verwendung einer Äquivalenzrelation R und unter Ausnutzung der Tatsache, daß D_A deterministisch ist, über die Zustände geführt. Zustände von A und B stehen in Relation R, wenn beide Automaten sich identisch verhalten, falls man sie in diesen Zuständen startet. Es muß also gelten, daß die Startzustände in der Relation R stehen, und für jeden Schritt von B ein entsprechender Schritt von A existiert, so daß R erhalten bleibt:

$$\forall v,w \cdot\; Z_A^O(v) \wedge Z_B^O(w) \;\rightarrow\; R(v,w) \tag{8}$$

$$\forall v,w,x,y,w' \cdot\; R[v,w] \wedge D_B[w,x,y,w'] \;\rightarrow\; \exists v'.D_A[v,x,y,v'] \wedge R[v',w'] \tag{9}$$

Es würde der Nachweis der Existenz von R genügen. In dieser Arbeit wird R explizit angegeben (Anhang C).

---

[1] Freie Variable seien implizit allquantifiziert, und die Typen werden zur besseren Lesbarkeit nicht explizit angegeben, da sie sich aus dem Kontext erschließen lassen.

[2] Zur Unterscheidung der Bezeichner der beiden Automaten werden diese im folgenden jeweils mit A bzw. B indiziert.

4 Automatisches Bearbeiten der Beweisaufgaben

Die im vorhergehenden Abschnitt dargestellte Beweisaufgabe ist nun mittels automatischer Beweisertechniken zu bearbeiten. Diese erste Fallstudie dient dazu, die Realisierbarkeit eines möglichen Anschlusses des Theorembeweisers SETHEO an das AUTOFOCUS-System zu untersuchen.

Als automatischer Beweiser wurde der im Teilprojekt A5 (weiter)entwickelte Beweiser SETHEO und dessen parallele Variante P-SETHEO verwendet. SETHEO kann Formeln der Prädikatenlogik 1. Stufe (mit Gleichheit und baumartig strukturierten Sorten) bearbeiten. Zur Bearbeitung bestimmter Teilaufgaben wurde der Model Checker SMV verwendet. Zunächst sehen wir uns jedoch die Charakteristik der gestellten Beweisaufgabe an. Daraus werden dann die Anforderungen an die Kopplung an AUTOFOCUS entwickelt. Von besonderem Interesse ist natürlich, zu untersuchen, welche Eigenschaften nur für diese Beweisaufgabe gelten, und welche sich auf ähnliche Aufgaben übertragen lassen. Die Übertragung der in dieser Fallstudie gewonnenen Erkenntnisse auf andere Arten von durch AUTOFOCUS erzeugte Beweisaufgaben wird in Abschnitt 5 ausführlich diskutiert.

4.1 Charakteristika und Probleme

Die wesentlichen Charakteristika unserer Fallstudie sind in der (vom ersten Autor entwickelten) Tabelle 1 eingetragen. Diese Tabelle umfaßt folgende verschiedene Kategorien:

Syntaktische Kriterien Hierzu gehören die Anzahl der zu lösenden Beweisbedingungen (pro Sitzung), die Zeitbeschränkung (hier ca. 1min), sowie äußere Kriterien der Aufgabe (Anzahl und Länge der Klauseln, Anzahl der verschiedenen Symbole und ihre Stelligkeit), sowie das Auftreten, bzw. Nichtauftreten von verschachtelten und rekursiven Termen.

An unserem Beispiel läßt sich erkennen, daß die Formeln vergleichsweise klein und einfach strukturiert sind, und innerhalb realistischer Zeiträume zu bearbeiten sind.

Logik Die hier aufgeführten Kategorien umfassen die Schwierigkeit der Transformation zwischen den Repräsentationsebenen (hier: AUTOFOCUS$\leftrightarrow$ PL1), sowie auftretende Theorien (Gleichheit, Arithmetik, Ströme), die die Abarbeitung durch automatische Beweiser stark beeinflussen können.

Von besonderer Bedeutung ist hier das Auftreten von endlichen Domänen (Zustände der Automaten), und von Strömen, die im allgemeinen mittels Induktion zu beweisen sind.

Gültigkeit der Beweisaufgaben Unsere Beweisaufgabe ist ein Theorem, also beweisbar. In vielen Fällen können jedoch auch — z.B. durch falsche Benutzereingaben — nicht beweisbare Aufgaben entstehen, die durch klassische automatische Beweiser meist nur schlecht behandelt werden können.

Die (relativ geringe) Komplexität der Teilbeweise (siehe Tabelle 3) und die Größe der Formeln lassen das ganze Problem in die Kategorie „proving in the small/medium" fallen. Allerdings ist, bedingt durch die SSD-Darstellung, auf Skalierbarkeit zu achten.

Category	Range	Value(s)
Syntax		
number	**S** M L	$\bar{N} = 1$ (with subtasks)
frequency	L **M** H	$f = 1min^{-1}$
size	L **M** H	$\bar{n_{cl}} = 75$ [50 ... 120]
prop. complexity	S **M** L	$l_{cl} = 4$ [1 ... 5]
richness	**S** M L	$N_{symb} = XXX$ $[XXX \ldots XXX]$
arity	**S** M L	$\bar{ar} = 2$ [0 ... 4]
nesting	nesting	**Y** N
	recursive	**Y** N
Logic		
source logic/language		AutoFocus
target logic		FOL
$d_{\text{source–target}}$	**L** **M** H	
simplification	**N** some Y	
sorts & types	static	Y **N**
equality	0% ... 100%	$\bar{n} = 60\%$
arithmetic	0% ... 100%	$\bar{n} = 0\%$
	domain	**finite** nat/int real
other theories	Y N	Name: traces
	decidable	Y **N**
Validity		
theorems/nonthms	0% ... 100%	$\bar{n} = 100\%$
proving in the	**S** **M** L	
Answer	**YN** answer subst	proof
soundness	reasonable **Y**	
completeness	reasonable **Y**	
System		
relation proof tasks	NA	
automatic generation	**Y** N	what
guidance	**Y** N	induction schemas
semantic info	N **some** Y	
limits known	**Y** some N	
human understandable	Y **some** **N**	

Tabelle 1: Charakteristika der Beweisaufgabe. Aus [Sch98].

Als Ergebnis ist „beweisbar/nicht beweisbar“, sowie in einigen Fällen auch der Beweis abzuliefern. Das Beweissystem muß natürlich korrekt und vollständig sein.

Systemspezifische Kriterien Die Beweisaufgaben werden automatisch generiert. Dabei ist die semantische Bedeutung der Symbole bekannt. Zur Kontrolle/Steuerung des Beweisers sind Induktionsschemate und Lemmata vorhanden. Die maximalen verfügbaren Resourcen (Laufzeit, Speicherbedarf) sind vorab bekannt.

Dieser erste Überblick ergibt die Einschätzung, daß die gestellte Beweisaufgabe mit automatischen Beweisern bearbeitet werden kann.

Ein schwerwiegendes Problem stellen die endlichen Domänen dar (z.B. die Zustände der Automaten), über die Beweise zu führen sind. Logisches Arbeiten mit endlichen Domänen läßt sich nicht ohne weiteres mittels Theorembeweisern für Prädikatenlogik behandeln.

Als Beispiel diene unsere Teilbedingung (8): $\forall v \cdot \forall w \cdot Z_A^0(v) \wedge Z_B^0(w) \quad \rightarrow \quad R(v, w)$. Dabei steht das $\forall v$ abkürzend für $\forall v : \mathsf{state}_A$. Die Grundmenge von state_A enthält eine endliche Aufzählung.

SETHEO versucht, einen Beweis von z.B. $\forall v : \mathsf{state}_A \cdot p(v)$ mittels Widerlegung zu finden. Dies bedeutet, daß nach einer Widerlegung von $p(v')$ gesucht wird, wobei v' eine neue (Skolem)-Konstante ist. Da die Grundmenge jedoch endlich ist, gibt es ein solches v' nicht.

In der Literatur gibt es verschiedene Möglichkeiten, Schließen über endliche Domänen mittels Prädikatenlogik zu modellieren. So kann zum einen ein spezielles Prädikat *dom* eingeführt werden, das die Domäne modelliert. So wird etwa $\forall X : \{\mathsf{a}, \mathsf{b}, \mathsf{c}\} \cdot p(X)$ in $(dom(a) \vee dom(b) \vee dom(c)) \wedge \forall X \cdot dom(X) \wedge p(X)$ transformiert. Theorembeweiser wie etwa Satchmo [MB88] oder der parallele MGTP [FHKF92] verwenden derartige Verfahren. Wird diese Transformation jedoch für SETHEO oder ähnliche Beweiser verwendet, so entstehen im allgemeinen sehr große Suchräume und lange Beweise.

Ein anderer Ansatz ersetzt die Aufzählung mittels „$\vee$“ durch ein kleines Logikprogramm *member*, das feststellt, ob ein Term in der Domäne (als Menge dargestellt) liegt oder nicht. Hier ist es jedoch — etwa im Gegensatz zur Implementierung des Prädikats „member“ aus PROLOG — auch erforderlich, daß $\neg member(X)$ korrekt behandelt wird. Ist die Kardinalität der endlichen Domänen und/oder die Anzahl der Quantoren über diese Mengen groß, so ist die Effizienz auch dieses Verfahrens äußerst gering und kann daher praktisch nicht eingesetzt werden.

Für die Bearbeitung unserer Beweisbedingung wurde daher eine Kombination von automatischem Beweisen für Prädikatenlogik 1. Stufe und einer effizienten Entscheidungsprozedur für endliche Domänen, dem Model Checking, untersucht und angewandt.

4.2 Kombination von ATP und Model Checking

Bei der Aufteilung der Beweisaufgabe in prädikatenlogische Teile und Teile für den Model Checker wurde pragmatisch vorgegangen: Alle Formelteile, die Quantoren über die Zustände der Automaten beinhalten, sind zunächst Kandidaten für Model Checking. In unserem Falle sind das die Eigenschaften der Relation R, d.h. (8) und (9). Sind diese Eigenschaften gezeigt, kann über die endlichen Quantoren abstrahiert werden, und die Formeln als Lemmata zu den SETHEO-Aufgaben gegeben werden.

Insgesamt erhält man also eine Aufteilung der Beweisaufgaben in zwei Bereiche (siehe Abbildung 6): Der Bereich *Top* enthält alle Teilaufgaben, die durch die Aufspaltung der ursprünglichen Aufgabe (6) entstehen. Diese Aufspaltung in Basis- und Schrittfälle ergibt sich natürlicherweise aus den zu bearbeitenden Induktionen. Ihre Behandlung mit SETHEO wird im Abschnitt 4.4.2 beschrieben. Die für die Beweise benötigten Eigenschaften der Automaten werden als Lemmata (mittlerer Teil) zu den Formeln gegeben. Diese Lemmata sind die Beweisbedingungen des Bereiches *Bottom*, die mit Hilfe von Model Checking bearbeitet werden.

Wünschenswert ist natürlich eine dynamische Koppelung beider Verfahren. Das hieße, daß beim Auftreten eines Teilzieles, das die Automateneigenschaften berührt, automatisch das Entscheidungsverfahren gestartet wird, um den logischen Wahrheitswert zu bestimmen. Eine derartige Koppelung ist im Prinzip vollständig und korrekt (siehe z.B. PVS [COR+95] und Meta-Amphion [LvB95]). In der Praxis treten jedoch verschiedene Probleme damit auf: Die

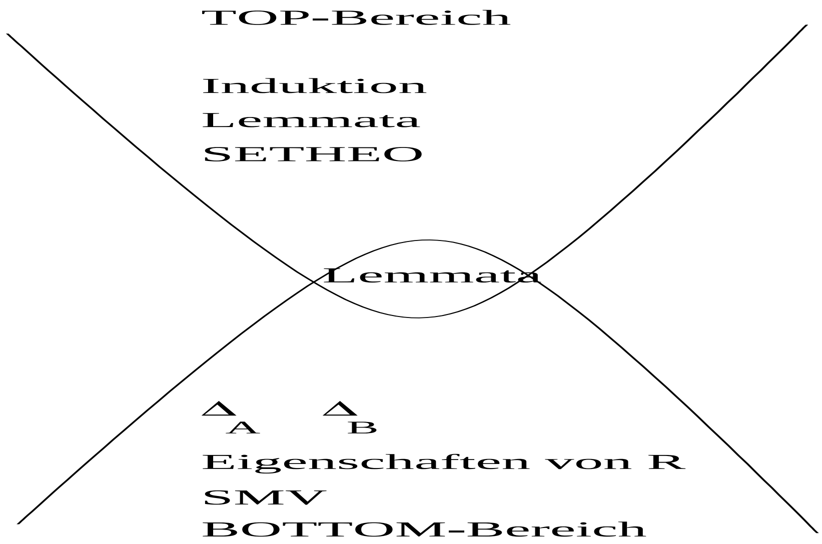

Abbildung 6: Aufteilung der Beweisaufgaben für SETHEO und Model Checking

Terme des Teilziels müssen vollständig instantiiert sein (was bei zielorientierten Beweisern wie SETHEO nur selten der Fall ist), und es treten dieselben Teilziele sehr häufig auf, was eine große Redundanz bei der Bearbeitung bewirkt.

Es wurde daher eine statische Aufteilung des Problems vorgenommen. Nach der Aufteilung werden die mit dem Entscheidungsverfahren zu bearbeitenden Teilaufgaben gelöst und dann für den Theorembeweiser als Lemmata zur Verfügung gestellt. Erschwerend kommt in unserem Beispiel hinzu, daß der Datentyp der Zustände des Automaten (z.B. state_A) ein zusammengesetzter Typ ist, der als Komponenten Werte der Sorte $\{\textsc{Req}\} \cup \mathsf{Nat}$ enthält. Dies bedeutet natürlich, daß die Domäne eigentlich unendlich ist, und somit von einem Modelchecker gar nicht bearbeitet werden kann.

Um trotzdem das Entscheidungsverfahren einsetzen zu können, wurden die Beweisaufgaben *abstrahiert*: in unserem Fall wurde der Datentyp $\mathsf{Nat} \cup \{\surd\} \cup \{\textsc{Req}\}$ durch den endlichen Aufzählungstypen $\{\surd, 0, \mathsf{any}\}$ ersetzt[3]. Ein derartiges Vorgehen ist jedoch nicht in allen Fällen korrekt und vollständig (siehe auch Abschnitt 5). Da in unserem Beispiel jedoch nur die Operatoren $X = 0$ und $X \neq 0$, bzw. Gleichheit $X = Y$ vorkommen, kann man sich leicht klarmachen, daß diese Abstraktion für unser Beispiel ausreichend ist. Würden jedoch andere Operatoren auftreten (z.B. Arithmetik), so könnte im allgemeinen keine derartige Abstraktion mehr angewendet werden.

4.3 Der Automatische Beweiser SETHEO

SETHEO (für Details siehe [LSBB92, LMG94]) ist ein automatischer Theorembeweiser für Prädikatenlogik erster Stufe in Klauselnormalform. Der SETHEO zugrundeliegende Kalkül ist Modellelimination [Lov78], ein korrekter und vollständiger Kalkül.

[3]Die Einführung der neuen Konstanten any ist im Prinzip dasselbe Vorgehen wie bei der Skolemisierung.

Die Eingabe von SETHEO besteht aus einer Menge von Klauseln $C_1 \wedge C_2 \wedge \ldots \wedge C_n$ mit Klauseln C_i der Form $C_i \equiv L_{i_1} \vee \ldots L_{i_{m_i}}$, wobei die L_{i_j} Literale bezeichnen. Literale sind atomare Formeln oder Negationen davon. Die Eingabesyntax ist PROLOG-ähnlich und sieht wie folgt aus: Eine Klausel

$$A_1 \vee \ldots A_k \vee \neg B_1 \vee \ldots \vee \neg B_l$$

wird geschrieben als:

```
A1 ; ... ; Ak <- B1 , ... , Bl.
```

Die A_i werden positive (Kopf-) Literale genannt, die B_i Rumpfliterale der Klausel. Eine Klausel mit $k = 0$, d.h. eine Klausel, die nur aus negativen Literalen besteht ist eine Startklausel oder „query". Das ";" kann also wie ein $\vee$, das "," wie ein $\wedge$, und das "<-" wie ein nach links gerichteter Implikationspfeil gelesen werden.

Wird SETHEO eine Menge derartiger Klauseln gegeben, so versucht SETHEO die Formel durch Konstruktion eines geschlossenen Tableaus zu widerlegen. Ein *Tableau* ist ein Baum, dessen Knoten mit Instantiierungen von Literalen der Formel markiert sind. Ein Ast in einem Tableau heißt *geschlossen*, falls der Pfad von der Wurzel zum aktuellen Blatt ein Paar komplementärer Literale enthält. Schließlich ist eine Formel genau dann *unerfüllbar*, wenn es ein zu dieser Formel gehörendes Tableau gibt, bei dem alle Äste geschlossen sind.

Ein Modell-Eliminations-Tableau läßt sich durch wiederholte Anwendung der folgenden Inferenzregeln konstruieren. Abbildung 7 zeigt eine Beispielsformel und ein zugehöriges geschlossenes Tableau.

Start Beginnend mit einem leeren Tableau, das nur aus dem Wurzelknoten (mit ϵ markiert) besteht, kann man eine beliebige Klausel als Startklausel auswählen und deren Literale als Nachfolgerknoten der Wurzel in das Tableau hängen.

Extension Gegeben sei ein Tableau mit einem offenen Blatt L. Wenn es ein Literal K in einer Klausel der Formel gibt, das mit L unifizierbar ist und ein komplementäres Paar erzeugt, (d.h. es gibt eine Substitution σ, sodaß $\sigma L = \neg\sigma K$), so können alle Literale dieser Klausel als Nachfolger an L angehängt werden. Der Blattknoten K wird als geschlossen markiert und die Substitution σ auf das ganze Tableau angewendet. Man beachte, daß hierbei neue offene Blätter („Subgoals") entstehen können.

Reduktion Wenn ein offenes Blatt L mit einem Vorgänger K existiert, so daß $\sigma L = \neg\sigma K$ für eine Substitution σ gilt, so kann der Blattknoten als geschlossen markiert werden. Auch hier wird die Substitution auf das ganze Tableau angewendet.

Für Details über SETHEO und dessen Implementierung sei auf die Literatur, z.B. [LSBB92, LMG94, GLMS94] verwiesen. Ein Zusatzmodul zu SETHEO, das für unsere Experimente wichtig ist, ist der DELTA-Präprozessor [Sch94]. Es erzeugt Unit-Klauseln in einer Phase des Preprocessing. Diese Klauseln werden dann als Lemmata für die Beweissuche verwendet. Bei der Verwendung von DELTA wird also zielorientierte Top-down-Suche mit Bottom-up Generierung verbunden. Somit lassen sich in vielen Fällen auch schwierige Probleme lösen.

Neben SETHEO wurden für unsere Experimente auch der im Teilprojekt A5 entwickelte parallele Beweiser P-SETHEO [SSW98, Wol98] eingesetzt. Hier werden parallel im Wettbewerb verschiedene SETHEO-Strategien ausprobiert.

$\leftarrow$ p(a,b), q(b,a).
p(a,b) $\leftarrow$ q(b,a).
q(b,a) $\leftarrow$ p(a,b).
p(a,b); q(b,a) $\leftarrow$.

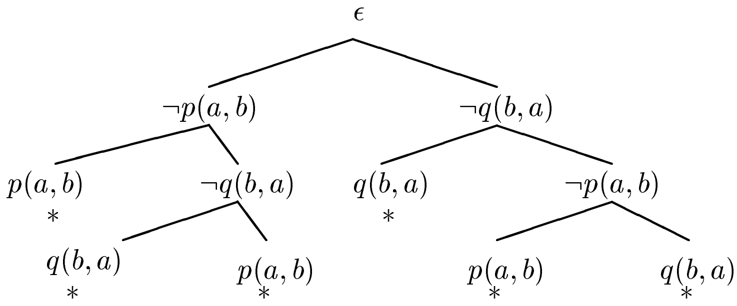

Abbildung 7: SETHEO Eingabeformel und Modelleliminations-Tableau

4.4 Formalisierung für SETHEO

Für die automatische Bearbeitung der Beweisaufgabe (6) muß diese mit den benötigten Axiomen, Lemmata, etc., in LOP, die Eingabesprache von SETHEO transformiert werden. Wir gehen hierbei bereits von der aufgeteilten Beweisaufgabe (siehe Abschnitt 4.2) aus. Die Transformation beinhaltet mehrere Schritte:

1. Syntaktische Transformation
2. Bearbeitung der Induktion
3. Hinzufügen von Axiomen und Lemmata
4. Simplifikation I
5. Transformation in Klauselnormalform (CNF)
6. Simplifikation II

Der Beweiser kann dann für die einzelnen Teilaufgaben (d.h. Basis- und Schrittfälle der Induktion) gestartet werden. Wurde ein Beweis gefunden, so muß dieser in

7. eine lesbare Form transformiert werden.

4.4.1 Syntaktische Transformation

Dieser Schritt beinhaltet im wesentlichen das Umbennen von Symbolen und Operatoren mit dem Ziel, eine volle ASCII-Darstellung aller Symbole und eine Präfixnotation aller Operatoren (z.B. `equal(X,Y)` für $X = Y$) zu erreichen. Dabei beginnen die Namen von Prädikats- und Funktionszeichen mit Kleinbuchstaben (z.B. p, *conc*), Variablennamen wie in PROLOG mit Großbuchstaben (z.B. X, Y).

In einer späteren Ausbaustufe des prototypischen Werkzeugs AUTOFOCUS soll eine entsprechende Repräsentation bereits zur Verfügung gestellt werden, so daß dieser Schritt entfallen kann.

4.4.2 Bearbeiten der Induktion

Da der Beweiser SETHEO — als rein prädikatenlogischer Beweiser — keine induktiven Aufgaben lösen kann, müssen externe Module diese Aufgaben wahrnehmen. In unserem Fall ist das relativ einfach, da bekannt ist,

(a) über welche Variablen die Induktion zu führen ist (im Beispiel die Ströme O und I), und

(b) wie die rekursive (higher-order) Definition des hier verwendeten Operators „$\frown$" ist.

Hieraus lassen sich — sogar mit Computerunterstützung (siehe [Reg94]) — die benötigten Induktionsschemata und zusätzliche Lemmata generieren. In unserem Fall bekommen wir für eine Beweisaufgabe der Form $\forall S : \mathsf{stream} \cdot \mathcal{F}(S)$ folgendes Schema:

$$\mathcal{F}(\langle\rangle) \wedge \forall S : \mathsf{stream}\ \forall S_1 : \mathsf{stream}\ \forall I : \mathsf{Item} \cdot (\mathcal{F}(S) \wedge S_1 = S \frown I \rightarrow \mathcal{F}(S_1))$$

Die beiden, durch $\wedge$ verknüpften Teilformeln sind nun (nach Instantiierung von $\mathcal{F}$) reine prädikatenlogische Formeln erster Stufe und werden als zwei eigenständige Beweisaufgaben behandelt[4]. Wird über $n > 1$ Variablen induziert, so gibt es natürlich eine verschachtelte Induktion mit 2^n Fällen. So sind bei unserer Beweisbedingung $2^2 = 4$ Fälle zu lösen.

4.4.3 Hinzufügen von Axiomen und Lemmata

Zu der zu beweisenden Bedingung („Conjecture") müssen im allgemeinen noch zusätzliche Teilformeln hinzugefügt werden, damit ein Beweis gefunden werden kann. In unserem Fall sind dies

- zusätzliche Annahmen, z.B. daß die Anzahl der Startzustände nicht leer ist, d.h. $\exists z \cdot Z_A^0(z)$,
- Definitionen der verwendeten Konstanten, meist in der Form: *Konstante* $\leftrightarrow$ *Rumpf* (ein typisches Beispiel ist die Definition der Kleene-Hülle in (4)), sowie
- Axiome über die auftretenden Operatoren und Symbole, z.B. $\forall S : \mathsf{stream}\ \forall I : \mathsf{Item} \cdot S \frown I \neq \langle\rangle$.

  Wird ein Theorembeweiser ohne eingebaute Behandlung von Gleichheit verwendet, so müssen auch die Axiome für die Gleichheit (Reflexivität, Symmetrie, Transitivität, Substitutionsaxiome) hinzugenommen werden.

Hierbei ist jedoch zu beachten, daß die meisten automatischen Theorembeweiser sehr empfindlich auf die Art und Menge der zugegebenen Formeln reagieren. Zu viele (unnötige) Formeln können den Suchraum sehr leicht so aufblähen, so daß nicht einmal mehr einfachste Beweisaufgaben gelöst werden können.

[4] Im Prinzip ist es natürlich möglich, die gesamte Formel auf einmal mit dem Beweiser zu bearbeiten. In diesem Fall (bedingt durch das „Ungleichgewicht" der Schwierigkeiten der beiden Teile) wächst der Suchraum gigantisch an. Da beide Teilformeln keine gemeinsamen Variablen enthalten, kann man die Formel problemlos zerlegen.

4.4.4 Vereinfachung

Eine Vereinfachung der Formel zur Reduktion ihrer Größe und Komplexität ist von größter Bedeutung für fast jede Anwendung automatischer Theorembeweiser. Nur so können die von der Anwendung generierten (und häufig Redundanzen enthaltenden) Beweisaufgaben bewältigt werden. Leider wurde dieser Aspekt bei der Entwicklung automatischer Theorembeweiser vollständig vernachlässigt[5]. Da für diese Fallstudie keine Werkzeuge zur Verfügung standen, wurden nur folgende Vereinfachungen durchgeführt (vor und nach der Herstellung der Klauselform):

1. Alle Definitionen, die nicht rekursiv sind (z.B. die Kleene-Hülle) werden expandiert. Dieses Vorgehen erspart jeweils Extensionsschritte in die Definitionen.

2. Die Definition der Kleene-Hülle (4) wird in zwei Teile ($\rightarrow$) und ($\leftarrow$) aufgespaltet. Dies ermöglicht zum einen weitere Simplifikationen (s.u.), zum anderen können bei normalen Verfahren der CNF-Erzeugung (wie sie z.Zt. bei uns verwendet werden) kürzere Klauseln erzeugt werden.

3. Unbedingte Gleichungen (in CNF-Form) werden in der ganzen Formel ersetzt. In vielen Fällen können damit große Vereinfachungen erzielt werden, und die Anzahl der auftretenden Variablen und Konstanten verringert werden. Hierbei werden alle Gleichungen der Form $c = t$ (mit c als Konstante) entfernt. Auf die restliche Formel wird die Substitution $[c \backslash t]$ angewendet.

4. Bei der Erzeugung der Klauselnormalform müssen die Quantoren durch Skolemisierung entfernt werden. Durch Optimierung dieses Verfahrens (siehe z.B. [Lov78]) können damit sehr häufig lange Skolemfunktionen vermieden werden. Statt dessen wird nur eine Skolemkonstante benötigt, die einen wesentlich kleineren Suchraum induziert.

5. Die Optimierung der Formel durch Entfernung nicht benötigter Formelteile muß sowohl eine gute Strategie für die Auswahl von Axiomen (siehe vorhergehenden Abschnitt), als auch die Entfernung „purer“ Formelteile beinhalten, also von Teilen, die an keinem Inferenzschritt teilnehmen können. Letztgenannte Optimierung wird automatisch im SETHEO durchgeführt.

   Da die Anzahl der Axiome in unserem Beispiel sehr klein war, konnte auf eine Axiomenauswahl verzichtet werden. Ansonsten sind Verfahren, wie z.B. in [SR98] beschrieben, anzuwenden.

Diese Simpifikationsregeln , besonders 3, können einfach an folgendem Beispiel (Schrittfall $I = \langle\rangle, O = Os \frown Y$ der Beweisbedingung 6) erläutert werden. Von Bedeutung ist hier

[5] Der Schwerpunkt bei der Entwicklung lag meist auf der Suche nach komplexen Beweisen in kleinen Formel, wie sie typischerweise bei mathematischen Problemen auftreten. Häufig wurden dann von Hand die wirklich für den Beweis benötigten Axiome ausgewählt und nur diese zur Formel gegeben. Die Sammlung derartiger Probleme in Benchmarkbibliotheken (z.B. TPTP) verschärfte noch die Vernachlässigung dieser Simplifikationsaspekte.

insbesondere die Definition der Kleene-Hülle (s_A und s_B, Formel 4). Gegeben ist also eine Formel der folgenden Struktur:

$$\begin{array}{l} \ldots \wedge \\ S_A[v, is, os, v'] \leftrightarrow \ldots \wedge \\ S_B[w, is, os, w'] \leftrightarrow \ldots \rightarrow \\ (is = \langle\rangle \ \wedge \ os = o^{'} \frown y \rightarrow \neg S_B[\ldots]) \end{array}$$

Läßt man diese Formel mit SETHEO bearbeiten, so benötigt SETHEO etwa 3 Sekunden um einen Beweis zu finden, wie aus Tabelle 2, 1. Zeile zu ersehen ist. Wird aus der Formel die nicht benötigte Definition von S_A herausgenommen (Vereinfachungsschritt 5), so ist die Laufzeit schon deutlich geringer (Tabelle 2, 2. Zeile). Werden jetzt noch die Gleichungen $is = \langle\rangle$ und $os = o^{'} \frown y$ auf die ganze Formel angewendet (Regel 3), so kann bei der Definition von S_B der ganze erste Teil der rechten Seite, d.h., $is = \langle\rangle \ \wedge \ os = \langle\rangle$ nach kleiner Vereinfachung entfernt werden. Die hieraus resultierende Formel ist nicht nur deutlich kleiner, es kann auch der Beweis wesentlich schneller gefunden werden (Tabelle 2, 3. Zeile).

Formel	Anzahl Klauseln	T[s]
original	122	3.10
Entfernen von S_A	68	1.38
zus. Vereinfachung	51	0.01

Tabelle 2: Experimente mit Vereinfachungen mit SETHEO

4.4.5 Erzeugung der Klauselnormalform

Zur Erzeugung der Klauselnormalform haben wir für diese Fallstudie einen Standardalgorithmus (z.B. [Lov78]) verwendet. Dieser übernimmt auch die Skolemisierung in optimierter Weise.

4.4.6 Transformation des Beweises

Ein von SETHEO erzeugter Beweis ist ein geschlossenes Modelleliminations-Tableau. Diese baumartige Struktur ist jedoch für den Benutzer nicht lesbar. Daher werden die entstehenden Beweise mit ILF-SETHEO [WS97] in eine lesbare (LaTeX-)Form überführt. Hier ist dann auch die unter 4.4.1 durchgeführte syntaktische Transformation der Symbole und Operatoren rückgängig gemacht.

4.5 Experimente und Resultate

Durch die Behandlung der Induktion bei den Beweisbedingungen (6) und (7) entstanden insgesamt 8 Beweisaufgaben. Diese konnten von SETHEO (bzw. P-SETHEO) vollautomatisch bearbeitet werden.

Die Zusammenstellung der Resultate ist in Tabelle 3 zu finden. Dabei bedeutet #cl die Größe der Formel in Klauseln und #inf die Länge des gefundenen Beweises (mit der erfolgreichsten

Strategie). T[s] schließlich ist die Laufzeit in Sekunden. Alle Experimente wurden auf einer Sun ULTRA II durchgeführt. Die Laufzeiten beziehen sich auf den eigentlichen Lauf von SETHEO bzw. P-SETHEO, nicht jedoch auf die Zeit zur Formelvorbereitung. Diese Zeit wird — nach vollständiger Implementierung des Prototyps — bei wenigen Sekunden liegen. Ebenso ist die Zeit für die Beweistransformation (ebenfalls wenige Sekunden) nicht eingerechnet.

Bei den Experimenten wurde festgestellt, daß die Beweise der Basisfälle ($i = \langle\rangle$ oder $o = \langle\rangle$) sehr einfach sind und daher von der Standardstrategie „default" schnell gelöst werden konnten. Der Schrittfall dagegen ist von relativ großer Komplexität (27 Inferenzen bzw. 31 Inferenzen[6]) und kann nur in größerer Tiefe gefunden werden. Daher wurde hierfür eine Kombination von Top-down (SETHEO) und Bottom-up (DELTA) Beweisverfahren angewandt. Eine derartige Koppelung, die insbesondere im Rahmen von P-SETHEO von großem Interesse ist, kann in vielen Fällen die Beweiszeiten drastisch reduzieren. Frühere Studien (z.B. [Sch95] und der im Teilprojekt A5 entwickelte SiCoTHEO-DELTA [Sch96, Sch97]) bestätigen dies.

Aufgabe	Fall		#cl	Strategie	#inf	T[s]
(6)	$i = \langle\rangle$	$o = \langle\rangle$	125	default	5	0.3
	$i = \langle\rangle$	$o = os\&y$	122	default	9	3.1
	$i = is\&x$	$o = \langle\rangle$	122	default	9	3.6
	$i = is\&x$	$o = os\&y$	60	top-down/bottom-up	$13^{\dagger}$	1.3+0.1
(7)	$i = \langle\rangle$	$o = \langle\rangle$	53	default	12	0.95
	$i = \langle\rangle$	$o = os\&y$	100	default	5	0.2
	$i = is\&x$	$o = \langle\rangle$	100	default	5	0.2
	$i = is\&x$	$o = os\&y$	63	top-down/bottom-up	$9^{\ddagger}$	5.3+1.1

Tabelle 3: Resulate mit SETHEO

4.6 Der Model Checker SMV

Der Model Checker SMV basiert auf der sogenannten Computation Tree Logic (CTL), einer propositionalen, verzweigenden Temporallogik [CES86]. Eine CTL Formel besteht aus propositionalen Ausdrücken und Formeln, sowie temporalen Operatoren. Ein solcher Operator besteht aus einem Quantor sowie einer temporalen Modalität. Der Quantor **A** erlaubt die Quantifizierung über *alle* möglichen Abläufe (computation paths), während der Quantor **E** die Existenz von mindestens einem Pfad fordert. So ist zum Beispiel die CTL Formel $\mathbf{AG}\mathcal{F}$ wahr in einem Zustand s, wenn $\mathcal{F}$ in allen zukünftigen Zuständen entlang aller möglichen Berechnungspfade gilt. Die in CTL gebräuchlichen Quantoren und Modalitäten und ihre Bedeutung sind in Tabelle 4 aufgelistet.

Die Semantik von CTL ist über einen markierten Zustandsübergangsgraphen definiert (siehe [BCMD90]). Der Model Checker SMV benutzt BDDs (binary decision diagrams) um diesen Graphen effizient darzustellen und um die Erfüllbarkeit einer CTL-Formel mittels einer bottom-up Fixpunktberechnung zu bestimmen. Für eine Beschreibung von SMV und des zugrundeliegenden Kalküls siehe [CGH+93].

[6]In der Tabelle 3 ist nur die Anzahl der Inferenzen für den abschließenden Top-Down Beweis mit SETHEO angegeben († und ‡). Mit dem Modul DELTA wurden 6 Ebenen vorberechnet und dabei 181 (610 für ‡) Unitklauseln erzeugt.

Modalität		Quantor	
X	„Next-time“ Operator	**A**	für alle Pfade
G	gilt universell	**E**	es existiert ein Pfad
F	„eventually“ Operator		
U	„until“ Operator		

Tabelle 4: Quantoren und Modlaitäten in CTL

4.7 Formalisierung für SMV

Zur Bearbeitung der Beweisaufgabe für das Model Checking müssen, wie in Abschnitt 4.2 beschrieben, die Spezifikationen der endlichen Automaten abstrahiert werden. In unserer Fallstudie bedeutet dies, daß die Markierungen der Zustände, die bisher int enthielten, durch eine endliche Aufzählung $\{\surd, 0, \mathsf{any}\}$ zu ersetzen sind.

Die auf diese Weise modifizierte Automatendarstellung muß dann in eine Menge von CTL-Formeln gebracht werden. Hierzu bietet das SMV-System jedoch eine gute Unterstützung an, da es erlaubt, Automaten in einer relativ natürlichen Form darzustellen. Für jeden der beiden Automaten A und B müssen die Definitionen, die Übergangsfunktion Δ, sowie die zwischen ihnen bestehende Relation R dargestellt werden.

4.7.1 Allgemeines Gerüst und Beweisbedingungen

Zur Darstellung der Automaten wird das Modulkonzept von SMV ausgenützt. Jeder Automat ist als abstrakter Datentyp deklariert, der als Parameter den Namen des Eingabestromes (in unserem Fall `p_msg_in`) bekommt. Im “Hauptprogramm”, dem Modul `main`, werden dann Variablen des entsprechenden Typs, sowie die gemeinsame Eingangsvariable `msg_in` deklariert. Der Rumpf unseres Eingabeformates ist in Abbildung 8 gezeigt.

```
MODULE state_a(p_msg_in) ...
MODULE state_b(p_msg_in) ...
  -- Definitionen von Automat A und B (siehe unten)

MODULE main
VAR
      msg_in   : {tick,0,any};
      fsma : state_a(msg_in);
      fsmb : state_b(msg_in);

DEFINE
      zoa := fsma.state = empty ;
      zob := fsmb.state = empty ;

      r   := (
      (fsma.state = leer & fsmb.state = empty) |
      (fsma.state = eins & fsmb.state = one &
        ...
      );

SPEC AG ((zoa & zob) -> r)

SPEC AG r
```

Abbildung 8: Rahmen und Hauptteil der SMV-Spezifikation

Die Variablendeklarationen erzeugen die Datenstrukturen für die beiden Automaten, die über die gemeinsame Eingabevariable `msg_in` miteinander verbunden sind. Letztere ist vom Typ {`tick,0,any`}, was unserer Abstraktion von $\{\surd\} \cup \{\text{Req}\} \cup \mathsf{nat}$ entspricht.

In dem mit dem Schlüsselwort `DEFINE` eingeleiteten Abschnitt werden Prädikate für die Anfangszustände (Z_A^0, bzw. Z_B^0), sowie die Relation R definiert. Hier ist deutlich zu sehen, wie auf die Variablen der einzelnen Automaten zugegriffen werden kann (z.B. `fsma.state = leer`).

Zwei Beweisbedingungen sind nun zu zeigen: die zu (8) gehörende Formel ist

```
AG ((zoa & zob) -> r).
```

Sie besagt, daß in allen möglichen Abläufen immer (`AG`) gelten soll, daß R gilt, wenn wir uns im Startzustand befinden. Die zweite Beweisbedingung `AG r` stellt sogar noch eine Verstärkung von (9) dar, und besagt, daß für alle Abläufe ständig R gelten muß.

4.7.2 Repräsentation der Automaten

Wie bereits gesagt, wird jeder Automat als ein parametrisierter Datentyp im Rahmen eines Moduls definiert. Das Modul enthält neben der Deklaration lokaler Variablen die Darstellung der Automatenübergänge, wie in Ausschnitten in den Abbildungen 9 und 10 gezeigt wird.

```
MODULE state_a(p_msg_in)

VAR
    state : {leer,eins,zwei,wait1,wait2,wait3,wait4};
    q1  : {tick,0,any};
    q2  : {tick,0,any};
    msg_out  : {tick,0,any};

INIT
    state = leer
```

Abbildung 9: Deklaration und Initialisierung der Zustandsübergange

Diese Darstellung entspricht der abstrahierten Deklaration des Systems A von Abschnitt 3.1. Da es in dieser Sprache keine zusammengesetzten Datentypen gibt, wird state_A (definiert als: $\text{Control\_State}_A \times$ Integer $\times$ Integer) mittels verschiedener Variablen (`state, q1, q2`) dargestellt.

Die Zustandsübergänge werden mittels `ASSIGN` für jede der Statusvariablen angegeben. Für die Ausgabe `msg_out` gilt ähnliches (Abbildung 10).

Dabei gibt es für jeden Übergang einen Standardfall: Trifft keine der angegebenen Bedingungen zu, so kann immer der mit „1“ markierte Wert genommen werden. Von besonderem Interesse ist die Default-Anweisung für den Ausgabestrom `msg_out`: Selbst wenn kein Zustandsübergang stattfindet, kann ein $\surd$ ausgegeben werden. Das entspricht genau unserer Definition von Δ_A (Anhang A).

4.8 Experimente und Resultate

Obige Beschreibung für die Systeme A und B wurden nun als Eingabe für das SMV-system genommen. Beide Beweisbedingungen über R konnten auf einer Sun ULTRA-sparc in $0.17s$ bewiesen werden.

```
ASSIGN
next(state) :=
  case
  state = leer & !msg_in = 0 & !msg_in = tick:   eins;
  state = eins & msg_in = 0 :                    wait1;
  state = eins & !msg_in = 0 & !msg_in = tick :  zwei;
      ...
  1: state;
  esac;
next(msg_out) :=
  case
  state = leer & !msg_in = 0 & !msg_in = tick:   tick;
  state = eins & msg_in = 0 :                    tick;.
     ...
  1: {tick};
  esac;
 next(q1) := ...
 next(q2) := ...
```

Abbildung 10: Darstellung der Zustandsübergänge und der Ausgabefunktion in SMV

5 Erfahrungen und Ausblick

In dieser Fallstudie wurde als Verifikationsaufgabe der Nachweis einer Verfeinerungsbeziehung zwischen zwei mit AutoFocus erstellten Spezifikationen eines zweielementigen Puffers behandelt. Die dabei auftretenden Beweisbedingungen wurden mit dem automatischen Theorembeweiser Setheo und dem Model Checker SMV bearbeitet. Diese Fallstudie dient zur Evaluierung, ob ein Anschluß eines automatischen Beweisers an den Werkzeugprototypen AutoFocus möglich und sinnvoll ist.

Die Struktur der gestellten Beweisaufgaben ist derart, daß diese natürlicherweise mittels Induktion zu beweisen sind. Setheo als Beweiser für reine Prädikatenlogik 1. Stufe kann keine induktiven Aufgaben bearbeiten. Deshalb mußte die Beweisaufgabe vorverarbeitet werden. Da bei der Verfeinerung grundsätzlich über Ströme induziert wird und (durch die Struktur der Spezifikation) bekannt ist, über welche Ströme die Induktion zu führen ist, gestaltete sich die Generierung der einzelnen Induktionsfälle sehr einfach. Die (im Prinzip leicht automatisch) herstellbaren Formeln für die einzelnen Fälle sind dann die eigentliche Eingabe für Setheo.

Als ein größeres Problem erwies sich jedoch, daß die Anzahl der (Kontroll-) Zustände endlich ist, und die Formeln Quantoren über diese enthalten. Da die Bearbeitung endlicher Domänen mit Setheo sehr ineffizient ist, wurde ein Weg beschritten, die Beweisaufgaben statisch in zwei Teile zu zerlegen. Der Teil, der Schlüsse über die Zustände enthält, wurde abstrahiert und mit dem Model Checker SMV bearbeitet. Die Ergebnisse wurden dann als Lemmata zu den Setheo-Aufgaben hinzugefügt. Damit konnte dann die gesamte Beweisverpflichtung vollautomatisch bearbeitet werden.

Es kann somit festgestellt werden, daß automatische Beweiser prinzipiell für die Aufgabenstellung der Verfeinerung von AutoFocus-Spezifikationen geeignet sind. Diese Fallstudie hat jedoch auch gezeigt, welche Probleme und offene Fragestellungen für einen praktisch einsetzbaren Anschluß automatischer Beweiser an AutoFocus zu lösen sind. Diese Fragestellungen betreffen zum einen Erweiterungen der formalen Methodik von AutoFocus (z.B. für ungezeitete oder nichtdeterministische Spezifikationen), als auch die beweisertechnische Seite (etwa Koppelung von Setheo und Model Checker, formale Rechtfertigung der gewählten

Abstraktion, oder Einfluß der Problemgröße auf das Verhalten des Beweisers).

Unsere Fallstudie hat gezeigt, daß ein gewinnbringender Anschluß des automatischen Beweisers SETHEO und dessen paralleler Variante (P-SETHEO) möglich ist. Ziel eines solchen Anschlusses wird es — nach Bearbeitung zentraler, oben angesprochener Fragestellungen — sein, wesentliche, wenn auch sicher nicht alle bei AUTOFOCUS auftretenden Beweisverpflichtungen automatisch erfolgreich bearbeiten zu können. Dies wird die Benutzer von AUTOFOCUS stark von lästiger Detailarbeit entlasten und somit die Praxisrelevanz des Werkzeugs und seiner formalen Methodik erheblich steigern.

Literatur

[Aut] *Homepage des* AUTOFOCUS-*Projektes.* Verfügbar im World Wide Web unter http://autofocus.informatik.tu-muenchen.de/.

[BCMD90] J. R. Burch, E. M. Clarke, K. L. McMillan und D. L. Dill. *Sequential Circuit Verification Using Symbolic Model Checking.* in *Proc. 27th ACM/IEEE Design Autom. Conf.* IEEE Comp. Soc. Predd 1990.

[BDD+93] Manfred Broy, Frank Dederich, Claus Dendorfer, Max Fuchs, Thomas Gritzner und Rainer Weber. *The Design of Distributed Systems - An Introduction to FOCUS.* TUM-I 9202-2, Technische Universität München, 1993. SFB-Bericht Nr.342/2-2/92 A.

[BS98] Manfred Broy und Ketil Stølen. FOCUS - *on System Development*, to appear 1998. Manuskript.

[CES86] E. M. Clarke, E. A. Emerson und A. P. Sistla. *Automatic Verification of finite-state concurrent systems using temporal logic specifications.* ACM Trans. Prog. Lang. Syst. **8** (1986) 2, 244–263.

[CGH+93] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long und K. L. McMillad L. A. Nessn an. *Verification of the Futurebus+ Cache Coherence Protocol.* in *Proc. 11th Intl. Symp. on Comp. Hardware Description Lang. and their Applications*, 1993.

[COR+95] J. Crow, S. Owre, J. Rushby, N. Shankar und M. Srivas. *A Tutorial Introduction to PVS.* in *WIFT'95 Workshop on Industrial strength formal specifcation techniques, Boca Raton, Fl, USA*, 1995.

[FHKF92] Masayuki Fujita, Ryuzo Hasegawa, Miyuki Koshimura und Hiroshu Fujita. *Model Generation Theorem Provers on a Parallel Inference Machine.* in ICOT (Hrsg.), *Proceedings of the International Conference on Fifth Generation Computer Systems.* ICOT 1992.

[Foc] *Homepage des* FOCUS-*Projektes mit allen relevanten Veröffentlichungen.* Verfügbar im World Wide Web unter http://www4.informatik.tu-muenchen.de/proj/focus/.

[GLMS94] Chr. Goller, R. Letz, K. Mayr und J. Schumann. *SETHEO V3.2: Recent Developments (System Abstract)* . in *Proc. CADE 12*, 1994, S. 778–782.

[HSS96] Franz Huber, Bernhard Schätz und Katharina Spies. *AutoFocus - Ein Werkzeugkonzept zur Beschreibung verteilter Systeme* . in Ulrich Herzog Holger Hermanns (Hrsg.), *Formale Beschreibungstechniken für verteilte Systeme*. Universität Erlangen-Nürnberg 1996, S. 165–174. Erschienen in: Arbeitsberichte des Instituts für mathematische Maschinen und Datenverarbeitung, Bd.29, Nr. 9.

[LMG94] R. Letz, K. Mayr und C. Goller. *Controlled Integration of the Cut Rule into Connection Tableau Calculi*. Journal Automated Reasoning (JAR) (1994) 13, 297–337.

[Lov78] D. W. Loveland. *Automated Theorem Proving: a Logical Basis*. North–Holland, 1978.

[LSBB92] R. Letz, J. Schumann, S. Bayerl und W. Bibel. *SETHEO: A High-Performance Theorem Prover*. Journal of Automated Reasoning **8** (1992) 2, 183–212.

[LvB95] M. Lowry und J. van Baalen. *META-AMPHION: Synthesis of Efficient Domain-Specific Program Synthesis Systems*. in *Proceedings of the 10th Knowledge-Based Software Engineering Conference*, 1995, S. 2–10.

[MB88] R. Manthey und F. Bry. *SATCHMO: a Theorem Prover Implemented in Prolog*. in *Conference on Automated Deduction (CADE)*, 1988.

[Reg94] F. Regensburger. *HOLCF: Eine konservative Erweiterung von HOL um LCF*. Dissertation, Technische Universität München, 1994.

[Sch94] J. Schumann. *DELTA — A Bottom-up Preprocessor for Top-Down Theorem Provers, System Abstract*. in *Proc. of Conference on Automated Deduction CADE 12*, LNAIBd. 814. Springer 1994, S. 774–777.

[Sch95] J. Schumann. *Using SETHEO for Verifying the Development of a Communication Protocol in* FOCUS *– A Case Study –*. in P. Baumgartner, R. Hähnle und J. Posegga (Hrsg.), *Proc. of Workshop Analytic Tableaux and Related Methods, Koblenz*, LNAIBd. 918. Springer 1995, S. 338–352.

[Sch96] J. Schumann. *SiCoTHEO: Simple Competitive parallel Theorem Provers*. in *Conference on Automated Deduction (CADE) 13*. Springer 1996.

[Sch97] J. Schumann. *SiCoTHEO — Simple Competitive parallel Theorem Provers based on SETHEO*. in J. Geller, V. Kumar und C.B. Suttner (Hrsg.), *Parallel Processing for Artificial Intelligence 3*, Machine Intelligence and Pattern Recognition 20. Elsevier 1997, S. 231–246.

[Sch98] J. Schumann. *Automated Theorem Proving in Software Engineering*. Habilitation, Technische Universität München, Institut für Informatik, 1998. in preparation.

[SR98] Gerhard Schellhorn und Wolfgang Reif. *Theorem Proving in Large Theories*. in W. Bibel und P. Schmitt (Hrsg.), *Automated Deduction*. Kluwer, 1998, Kap. III.11. (to appear).

[SSW98] Johann Schumann, Christian Suttner und Andreas Wolf. *Parallel Theorem Provers based on SETHEO.* in W. Bibel und P. Schmitt (Hrsg.), *Automated Deduction.* Kluwer, 1998, Kap. II.7. (to appear).

[Wol98] Andreas Wolf. *p-SETHEO: Strategy Parallelism in Automated Theorem Proving.* in *Proceedings of 7th International Conference on Analytic Tableaux and Related Methods*, Lecture Notes in Artifical Intelligence (LNAI). Springer-Verlag 1998. to appear.

[WS97] A. Wolf und J. Schumann. *ILF-SETHEO: Processing Model Elimination Proofs for Natural Language Output.* in *Conference on Automated Deduction (CADE) 14*, 1997, S. 61–65.

Danksagung

Wir danken Bernhard Schätz für seine Anregungen und Unterstützung bei der Suche nach einem geeigneten Beweisprinzip, und Ingolf Krüger für seine aufmerksame und hilfreiche Durchsicht dieser Arbeit.

A Spezifikation des Systems A

Die Transitionsfunktion $\Delta_A : \mathsf{Message} \times \mathsf{State}_A \to \mathcal{P}(\mathsf{Nat} \times \mathsf{State}_A)$ mit

$$\begin{array}{rcl} \mathsf{Message} & = & \mathsf{Nat} \cup \{\textsc{Req}\} \\ \mathsf{ControlState}_A & = & \{Leer, Eins, Zwei, Wait1, Wait2, Wait3, Wait4\} \\ \mathsf{State}_A & = & \mathsf{ControlState}_A \times \mathsf{Nat} \times \mathsf{Nat} \end{array}$$

ist im folgenden definiert. Dabei bezeichnet die Menge $A^{\surd}$ die Nachrichtenmenge A, die um das Symbol $\surd$ erweitert wurde, also $A^{\surd} = A \cup \{\surd\}$.

$\forall i \in \mathsf{Message}^{\surd}, o' \in \mathsf{Nat}^{\surd}, d, q1, q1', q2, q2' \in \mathsf{Nat}, s, s' \in \mathsf{ControlState}_A :$

$$(o', (s', q1', q2')) \in \Delta_A(i, (s, q1, q2)) \Leftrightarrow$$

$$\begin{array}{llllllllllllll} & s = Leer & \wedge & d > 0 & \wedge & i = d & \wedge & o' = \surd & \wedge & q1' = d & \wedge & q2' = q2 & \wedge & s' = Eins \\ \vee & s = Eins & & & \wedge & i = \textsc{Req} & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Wait1 \\ \vee & s = Wait1 & & & & & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Wait2 \\ \vee & s = Wait2 & & & & & \wedge & o' = q1 & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Leer \\ \vee & s = Eins & \wedge & d > 0 & \wedge & i = d & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = d & \wedge & s' = Zwei \\ \vee & s = Zwei & & & \wedge & i = \textsc{Req} & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Wait3 \\ \vee & s = Wait3 & & & & & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Wait4 \\ \vee & s = Wait4 & & & & & \wedge & o' = q1 & \wedge & q1' = q2 & \wedge & q2' = q2 & \wedge & s' = Eins \\ \vee & s = Leer & & & \wedge & i = \surd & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Leer \\ \vee & s = Eins & & & \wedge & i = \surd & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Eins \\ \vee & s = Zwei & & & \wedge & i = \surd & \wedge & o' = \surd & \wedge & q1' = q1 & \wedge & q2' = q2 & \wedge & s' = Zwei \end{array}$$

B Spezifikation des Systems B

Die Funktion $\Delta_B : \mathsf{Message} \times State_B \to \mathcal{P}(\mathsf{Message} \times \mathsf{State}_B)$ mit

$$\begin{array}{rcl} \mathsf{ControlState}_B & = & \{Leer, Eins, Zwei, Wait, Trans\} \\ \mathsf{State}_B & = & \mathsf{ControlState}_B \times \mathsf{Nat}^2 \times \mathsf{Message}^4 \end{array}$$

ist definiert durch

$\forall i, x1, x1', x2, x2' \in \mathsf{Message}^{\surd}, o' \in \mathsf{Nat}^{\surd}, b1, b1', b2, b2' \in \mathsf{Nat}, s, s' \in \mathsf{ControlState}_B :$

$$(o', (s', b1', b2', x1', x2', y1', y2')) \in \Delta_B(i, (s, b1, b2, x1, x2, y1, y2)) \Leftrightarrow$$

$$\begin{array}{lll} & & x1 = \surd \wedge y1' = \surd \wedge b1' = b1 \\ & \vee & (x1 \neq 0 \wedge x1 \neq \surd) \wedge y1' = \surd \wedge b1' = x1 \\ & \vee & x1 = 0 \wedge y1' = b1 \wedge b1' = b1 \\ \\ \wedge & & x2 = \surd \wedge y2' = \surd \wedge b2' = b2 \\ & \vee & (x2 \neq 0 \wedge x2 \neq \surd) \wedge y2' = \surd \wedge b2' = x2 \\ & \vee & x2 = 0 \wedge y2' = b2 \wedge b2' = b2 \end{array}$$

$$
\begin{array}{lll}
\wedge & & s = Leer \wedge d > 0 \wedge i = d \wedge x1' = i \wedge x2' = \surd \wedge o' = \surd \wedge s' = Eins \\
& \vee & s = Eins \wedge i = \text{REQ} \wedge x1' = \text{REQ} \wedge x2' = \surd \wedge o' = \surd \wedge s' = Wait \\
& \vee & s = Wait \wedge y1 \neq \surd \wedge x1' = \surd \wedge x2' = \surd \wedge o' = y1 \wedge s' = Leer \\
& \vee & s = Eins \wedge d > 0 \wedge i = d \wedge x1' = \surd \wedge x2' = i \wedge o' = \surd \wedge s' = Zwei \\
& \vee & s = Zwei \wedge i = \text{REQ} \wedge x1' = \text{REQ} \wedge x2' = \text{REQ} \wedge o' = \surd \wedge s' = Trans \\
& \vee & s = Trans \wedge y1 \neq \surd \neq y2 \wedge x1' = y2 \wedge x2' = \surd \wedge o' = y1 \wedge s' = Eins \\
& \vee & i = y1 = y2 = \surd \wedge o' = x1' = x2' = \surd \wedge s = s'
\end{array}
$$

C Die Äquivalenzrelation R

Die Äquivalenzrelation $R \subset \mathsf{State}_A \times \mathsf{State}_B$ ist definiert durch

$$R[(z, q1, q2), (s, b1, b2, x1, x2, y1, y2)] \leftrightarrow$$

$$
\begin{array}{lll}
& & z = s = Leer \\
\vee & & z = s = Eins \\
& \wedge & (q1 = b1 \vee q1 = x1) \\
\vee & & z = s = Zwei \\
& \wedge & (q1 = b1 \vee q1 = x1) \\
& \wedge & (q2 = b2 \vee q2 = x2) \\
\vee & & s = Wait \\
& \wedge & z = Wait1 \\
& \wedge & x1 = 0 \\
& \wedge & b1 = q1 \\
\vee & & s = Wait \\
& \wedge & z = Wait2 \\
& \wedge & y1 = q1 \\
\vee & & s = Trans \\
& \wedge & z = Wait3 \\
& \wedge & x1 = 0 \wedge x2 = 0 \\
& \wedge & b1 = q1 \wedge b2 = q2 \\
\vee & & s = Trans \\
& \wedge & z = Wait4 \\
& \wedge & y2 = q2 \wedge y1 = q1 \\
& \wedge & b2 = q2
\end{array}
$$

SFB 342:	Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994 auf besondere Anforderung

342/01/95 A	Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A	Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A	Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A	Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A	Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A	Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A	Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A	Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A	Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A	Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A	Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A	Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A	Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A	Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A	Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik
342/17/95 A	Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A	Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications
342/19/95 A	Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A	Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication

Reihe A

342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control

342/22/95 A Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes

342/23/95 A Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation

342/24/95 A M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems

342/01/96 A Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries

342/02/96 A Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs

342/03/96 A Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes

342/04/96 A Thomas Huckle: Efficient Computation of Sparse Approximate Inverses

342/05/96 A Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification

342/06/96 A Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components

342/07/96 A Richard Mayr: Some Results on Basic Parallel Processes

342/08/96 A Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht

342/09/96 A P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme

342/10/96 A Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments

342/11/96 A Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions

342/12/96 A Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete

342/13/96 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks

342/14/96 A Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

342/15/96 A Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus

342/16/96 A Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen

342/17/96 A Richard Mayr: Model Checking PA-Processes

342/18/96 A Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States

342/01/97 A Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsfähigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken

Reihe A

342/02/97 A	Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations
342/03/97 A	Bernhard Mitschang: Technologie für Parallele Datenbanken - Bericht zum Workshop
342/04/97 A	nicht erschienen
342/05/97 A	Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik
342/06/97 A	Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump
342/07/97 A	Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming
342/08/97 A	Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement
342/09/97 A	Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic
342/10/97 A	Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus
342/11/97 A	Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity
342/12/97 A	Christian B. Czech: Architektur und Konzept des Dycos-Kerns
342/13/97 A	Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow
342/14/97 A	Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level
342/15/97 A	Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen
342/16/97 A	Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung
342/17/97 A	Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing
342/18/97 A	Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations
342/19/97 A	Frank Wallner: Model Checking LTL Using Net Unfoldings
342/20/97 A	Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –
342/21/97 A	Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving
342/22/97 A	T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)
342/23/97 A	Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP
342/24/97 A	Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II
342/25/97 A	Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS
342/26/97 A	Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration

Reihe A

342/27/97 A	Max Breitling: Formalizing and Verifying TimeWarp with FOCUS
342/28/97 A	Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)
342/29/97 A	Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems
342/01/98 A	A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): „Anwendungsbezogene Lastverteilung“, ALV'98
342/02/98 A	Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in FOCUS
342/03/98 A	Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten
342/04/98 A	Stefan Bischof, Ernst-W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions
342/05/98 A	St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results
342/06/98 A	Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems
342/07/98 A	Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS
342/08/98 A	Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement
342/09/98 A	Javier Esparza, Editor: Proceedings of INFINITY'98
342/10/98 A	Richard Mayr: Lossy Counter Machines
342/11/98 A	Thomas Huckle: Matrix Multilevel Methods and Preconditioning
342/12/98 A	Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning
342/13/98 A	Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time
342/01/99 A	Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS