

Antrag auf Projektförderung durch die FNK

## **Monadische Berechnungslogiken in HOL**

Lutz Schröder  
Fachbereich 3, Mathematik/Informatik  
Universität Bremen

27. Oktober 2004

# Inhaltsverzeichnis

<b>1</b>	<b>Allgemeine Angaben</b>	<b>1</b>
1.1	Antragsteller . . . . .	1
1.2	Thema . . . . .	1
1.3	Kennwort . . . . .	1
1.4	Fachgebiet und Arbeitsrichtung . . . . .	1
1.5	Voraussichtliche Gesamtdauer . . . . .	1
1.6	Antragszeitraum . . . . .	2
1.7	Gewünschter Beginn der Förderung . . . . .	2
1.8	Zusammenfassung . . . . .	2
<b>2</b>	<b>Stand der Forschung, eigene Vorarbeiten</b>	<b>2</b>
2.1	Stand der Forschung . . . . .	2
2.1.1	Monadische Berechnungslogik . . . . .	2
2.1.2	Programmverifikation in Isabelle/HOL . . . . .	4
2.1.3	Coalgebraische Spezifikation und Modallogik . . . . .	4
2.1.4	Beweisabstraktion . . . . .	4
2.2	Eigene Vorarbeiten . . . . .	4
2.2.1	Die Common Framework Initiative . . . . .	4
2.2.2	Integrierte funktionale Spezifikation und Programmierung . . . . .	5
2.2.3	Coalgebraische Modallogik . . . . .	6
2.2.4	Beweisunterstützung in Isabelle/HOL . . . . .	6
2.2.5	Monadische Berechnungslogiken . . . . .	6
<b>3</b>	<b>Ziele und Arbeitsprogramm</b>	<b>7</b>
3.1	Ziele . . . . .	7
3.2	Arbeitsprogramm . . . . .	9
3.2.1	Logikentwurf . . . . .	9
3.2.2	Logikimplementierung . . . . .	10
3.2.3	Dissertationsthemen . . . . .	11
3.2.4	Zeitplan . . . . .	11
<b>4</b>	<b>Beantragte Mittel</b>	<b>11</b>
4.1	Personalbedarf . . . . .	11
4.2	Wissenschaftliche Geräte . . . . .	13
4.3	Verbrauchsmaterial . . . . .	13
4.4	Reisen . . . . .	13
4.5	Publikationskosten . . . . .	13
4.6	sonstige Kosten . . . . .	13
<b>5</b>	<b>Voraussetzungen für die Durchführung des Vorhabens</b>	<b>13</b>
<b>6</b>	<b>Erklärungen</b>	<b>14</b>

<b>7</b>	<b>Unterschrift</b>	<b>14</b>
<b>8</b>	<b>Verzeichnis der Anlagen</b>	<b>15</b>

# 1 Allgemeine Angaben

Antrag auf Gewährung einer Projektförderung (Neuantrag)

## 1.1 Antragsteller

**Lutz Schröder**, Prof. Dr. rer. nat.  
Professor (Vertretung) für theoretische Informatik  
geb. 15.08.1970, Deutscher  
Universität Bremen  
Studiengang Informatik des Fachbereichs Mathematik/Informatik

### *Dienstliche Adresse*

Universität Bremen  
Fachbereich 3, Mathematik/Informatik  
Postfach 33 04 40, 28334 Bremen  
Tel. (0421) 218-4683, Fax: (0421) 218-3054  
E-Mail lschrode@informatik.uni-bremen.de

### *Privatadresse*

Emmastr. 273  
28213 Bremen  
Tel. (0421) 35 41 06

## 1.2 Thema

Weiterentwicklung der monadischen dynamischen Logik und ihrer halbautomatischen Beweisunterstützung in Isabelle/HOL.

## 1.3 Kennwort

HOL-MDL

## 1.4 Fachgebiet und Arbeitsrichtung

Fachgebiet: Theoretische und Praktische Informatik  
Arbeitsrichtung: Theorie der Programmierung, Semantik,  
formale Methoden und Werkzeuge,  
sichere Systeme, korrekte Programmentwicklung

## 1.5 Voraussichtliche Gesamtdauer

3 Jahre

## 1.6 Antragszeitraum

3 Jahre

## 1.7 Gewünschter Beginn der Förderung

1. 2. 2004

## 1.8 Zusammenfassung

Mit zunehmender Delegation kritischer Aufgaben an Software wird die formale Verifikation von Programmen ein immer zentraleres Thema. Während die Spezifikation und Verifikation seiteneffektfreier (funktionaler oder logischer) Programme relativ gut erforscht ist, bereitet die entsprechende Behandlung seiteneffektbehafteter Programme (die weiterhin in der Praxis wesentlich gängiger sind) nach wie vor große Schwierigkeiten, die nicht zuletzt von der mangelnden Anpassungsfähigkeit der bisher verwendeten Formalismen herrühren. Im hier beantragten Projekt soll ein vom Antragsteller und anderen entwickelter Ansatz zur logischen Unterstützung der imperativen Programmierung vertieft werden, der auf einer Repräsentation generischer Seiteneffekte in an sich seiteneffektfreiem Rahmen durch sogenannte Monaden basiert. Es hat sich gezeigt, dass sich klassische Formalismen zur Spezifikation imperativer Programme wie insbesondere Floyd-Hoare-Logik und dynamische Logik im Rahmen der monadischen Darstellung sehr viel allgemeiner fassen lassen, so dass man Kalküle erhält, die auf grundverschiedene Seiteneffekte wie etwa Ein-/Ausgabe, Ausnahmen, dynamische Referenzen und Nichtdeterminismus anwendbar sind. Im Rahmen dieses Projekts sollen einerseits die bisher beschriebenen monadischen Kalküle hinsichtlich ihres Anwendungsspektrums weiter verbreitert und die Erkenntnisse über die entsprechenden Beweissysteme vertieft werden, sowie andererseits eine Beweisunterstützung im halbautomatischen Theorembeweiser Isabelle/HOL implementiert werden.

## 2 Stand der Forschung, eigene Vorarbeiten

### 2.1 Stand der Forschung

#### 2.1.1 Monadische Berechnungslogik

Die Funktional-imperative Programmierung mittels Monaden spielt in modernen funktionalen Programmiersprachen, insbesondere Haskell, eine große Rolle. Hauptziel des hier beantragten Projekts ist der Ausbau des logischen Rahmenwerks zur Spezifikation von monadischen Programmen.

**Monaden in der funktionalen Programmierung** In [26] wird vorgeschlagen, Monaden zur verkapselten Darstellung generischer Seiteneffekte zu verwenden; so lassen sich etwa Nicht-Terminierung, Ausnahmen, Nichtdeterminismus, globaler Speicher oder Ein-/Ausgabe-Effekte als Monaden verstehen. Der Grundgedanke besteht darin, den Ergebnistyp von Funktionen im Bedarfsfall vom eigentlichen Wertetyp auf einen Typ von Berechnungen über diesem Wertetyp zu liften. Moggi zeigt, dass die (aus dem Kontext der kategoriellen Algebra stammende) kategorielle Grundlage zu einer Sprache mit Bindungskonstrukt (`let`) korrespondiert. Aufbauend auf [60] sind diese Ideen im Design von Haskell umgesetzt, in Gestalt sowohl einer Typklasse von Monaden als auch verschiedener vorgefertigter Instanzen dieser Klasse. In [14] wird die Semantik eines Java-Fragments als Monade beschrieben. Die für monadische Programmierung zur Verfügung stehende Syntax erlaubt einen imperativ anmutenden Programmierstil, der dennoch den Hauptvorteil funktionaler Programmierung bewahrt, d.h. mathematischer Analyse ohne weiteres zugänglich bleibt.

**Monadische Hoare-Logik** Der naheliegende Gedanke, den Floyd-Hoare-Kalkül zur Verifikation imperativer Programme auf monadische Programme auszudehnen, hat in der Literatur bisher nur wenig Berücksichtigung gefunden. Ein knapp gehaltener Ansatz in dieser Richtung findet sich bereits in [26], bleibt aber auf einer sehr spezialisierten Ebene, die im wesentlichen auf deterministische Speicheroperationen hinausläuft und insofern über den klassischen Kalkül eigentlich nicht hinausgeht. Ähnliche Ideen sind in ausgearbeiteterer Form in [7] dargestellt. Interessanterweise erlaubt das Haskell-Testwerkzeug QuickCheck in seiner neuesten Version das automatische Testen monadischer Hoare-Tripel [3]. Für letztere wird eine Semantik skizziert, die allerdings ad hoc gehalten und insbesondere mit den Schlußregeln des konventionellen Hoare-Kalküls im allgemeinen nicht verträglich ist. Über diese Vorarbeiten gehen die im Rahmen des Bremer DFG-Projekts HasCASL erzielten Ergebnisse (s. 2.2.5) deutlich hinaus.

**Auswertungslogik** Als Verallgemeinerung der dynamischen Logik auf Berechnungen mit expliziten Rückgabewerten (die eigentliche dynamische Logik kennt nur Zustandsmanipulationen) ist in [39] die sogenannte *Auswertungslogik* (Evaluation Logic) eingeführt worden. Diese Logik ist zugeschnitten auf Beweise über auf Monaden basierende funktional-imperative Programme und dabei ausdrucksstärker als die Hoare-Logik, bringt allerdings auch größere semantische Schwierigkeiten mit sich. Die in [39] angegebene Semantik basiert auf Hyperdoktrinen über monadischen Kategorien und ist zusätzlich über die Interpretation der dynamischen Modaloperatoren parametrisiert. Dies verlagert einen Großteil des Korrektheitsproblems auf die Ebene der Modelle, d.h. man hat insgesamt nur wenig echte Generizität; z.B. muss für die Zustandsmonade die Zustandsabhängigkeit von Formeln explizit in das Modell eingebaut werden. Als Lösung

dieses Problems wird in [27] eine allein auf die Monade bezogene Semantik vorgeschlagen; diese Semantik hat allerdings einen grundlegend anderen, um nicht zu sagen der ursprünglichen Semantik widersprechenden Charakter, indem hier die für Modallogik typische Lokalität (Zustandsabhängigkeit) aufgegeben wird. Ein herausragendes Ergebnis des Bremer DFG-Projekts HasCASL ist die Überbrückung dieser semantischen Lücke durch eine axiomatische Semantik (s. 2.2.5).

### **2.1.2 Programmverifikation in Isabelle/HOL**

Der halbautomatische Theorembeweiser Isabelle/HOL ist unter anderem geeignet zur Verifikation einfacher (insbesondere terminierender) funktionaler Programme [34]. Ferner liegen Arbeiten über die Spezifikation der operationalen Semantik imperativer und objektorientierter Programmiersprachen sowie über entsprechende spezialisierte Floyd-Hoare-Kalküle in Isabelle/HOL vor [33, 59].

### **2.1.3 Coalgebraische Spezifikation und Modallogik**

In jüngerer Zeit hat sich die Coalgebra als Rahmen für die abstrakte Behandlung reaktiver Systeme [47] zunehmend durchgesetzt. Hierbei ist die Coalgebra tatsächlich im wörtlichen Sinne dual zur universellen Algebra, wie sie in der Spezifikation funktionaler Systeme verwendet wird. Es hat sich herausgestellt, dass die coalgebraische Entsprechung der Gleichungslogik gerade die Modallogik ist [12, 20, 18, 36, 37, 38], die sich in der Tat auch als zur reaktiven Spezifikation unter anderem objektorientierter Programme geeignet erwiesen hat [19, 46]. Wichtig für das hier beantragte Projekt ist hierbei u.U. insbesondere die axiomatische Behandlung generischer Modaloperatoren gemäß [36, 37]; s. 2.2.3 und 3.2.1.

### **2.1.4 Beweisabstraktion**

In Rahmen des Bremer DFG-Projekts AWE ist ein Abstraktionsmechanismus zur Erleichterung der Wiederverwendung von Beweisen in Isabelle implementiert worden [15]. Für die Zwecke des hier beantragten Projekts ist dabei insbesondere von Interesse, dass auf diesem Wege auch Abstraktionen über syntaktische Entitäten ermöglicht werden, für die Isabelle selbst keine Quantifikation bietet. Dies betrifft vor allem Typkonstruktoren (Isabelle verfügt zwar über einen Typklassenmechanismus für einfache Typen, aber nicht über sogenannte Konstruktorklassen); s. hierzu 3.2.2.

## **2.2 Eigene Vorarbeiten**

### **2.2.1 Die Common Framework Initiative**

Angesichts der Tatsache, dass es zur erfolgreichen Anwendung von algebraischen Spezifikationssprachen in der Industrie und zur Verbreitung von Werkzeugen und

Entwicklungsumgebungen auf lange Sicht eines internationalen Standards bedarf, wurde 1995 auf Initiative der IFIP Working Group 1.3 beschlossen, ein solches einheitliches Rahmenwerk für Spezifikationssprachen zu entwerfen (*Common Framework Initiative for Algebraic Specification and Development* (CoFI) [4]). Von 1998 bis 2001 wurde die CoFI Working Group mit Reisemitteln durch die Europäische Union gefördert.

Referenzpunkt des von der Arbeitsgruppe entwickelten Rahmenwerks ist die Kernsprache CASL (*Common Algebraic Specification Language*), die bewußt nur bereits relativ gut verstandene Konzepte enthält. Die Basislogik von CASL erweitert die Prädikatenlogik erster Stufe um Subtypen, Partialität, und term erzeugte Datentypen. Ein Ziel von CoFI ist es, um diese zentrale Sprache herum eine Familie von unterschiedlich komplexen Sprachen für verschiedene Zwecke zu entwickeln, so daß Sprachen und vor allem auch vorhandene Werkzeuge zueinander in Relation gesetzt und in einem gemeinsamen Rahmen verwendet werden können; dies wird dadurch erleichtert, daß die Strukturierungsmechanismen der Sprache logikunabhängig gehalten sind.

Die Bremer CoFI-Gruppe, der gegenwärtig unter anderem B. Krieg-Brückner, T. Mossakowski und der Antragsteller angehören, spielt innerhalb der Gesamtinitiative eine aktive Rolle. Der Antragsteller hat mit internationalen Veröffentlichungen [16, 22, 43, 44, 45, 58] zur Entwicklung von CASL wesentlich beigetragen, insbesondere hinsichtlich Methodologie und Semantik. Die Leitung der CoFI-Werkzeugentwurfsgruppe liegt in Bremen. Insbesondere wird in Bremen im Rahmen des DFG-Projekts MULTIPLE der heterogene CASL-Werkzeugsatz entwickelt [29, 30], der verschiedene Logiken und die zugehörigen Werkzeuge integriert; die im Rahmen des hier beantragten Projekts entwickelten Logiken und Beweiswerkzeuge werden sowohl wichtiger Teil dieses Logiksystems sein als auch vom heterogenen Rahmenwerk profitieren.

### 2.2.2 Integrierte funktionale Spezifikation und Programmierung

Zur durchgehenden formalen Spezifikation und Entwicklung funktionaler Programme bedarf es einer Sprache, die zum einen das komplexe Typsystem moderner funktionaler Programmiersprachen wie etwa Haskell möglichst weitgehend unterstützt und zum anderen über einen geeigneten Ausführbarkeitsbegriff verfügt, der allgemeine Rekursion erlaubt, ohne dabei einfach auf syntaktische Verwechslung zu bauen. Eine solche Sprache wird im DFG-Projekt HasCASL („Kombination von algebraischer Spezifikation und funktionaler Programmierung als Umgebung für formale Softwareentwicklung“) entwickelt, das seit Juli 2001 in Bremen unter Leitung des Antragstellers durchgeführt wird (Laufzeit der zweiten Phase bis 2007). Die Sprache HasCASL baut auf Moggis partiellen  $\lambda$ -Kalkül [24, 25] auf, unter Ausnutzung der aus ihm erwachsenden internen Logik [51]; dieser Grundkalkül wird verschiedentlich syntaktisch erweitert sowie mit einem mächtigen Typklassenmechanismus versehen, der ähnlich wie in Haskell eine differenzierte

Typisierung spezialisierter polymorpher Funktionen erlaubt. Die so entstehende Sprache ist hinreichend mächtig, um die Kodierung eines Ausführbarkeitsbegriffs in vollständigen Halbordnungen analog HOLCF [41] zu gestatten.

Der HASCASL Sprachentwurf wird in [53] beschrieben; eine ausführlichere Version ist in Vorbereitung [52]. Die vollständige Sprachdefinition findet sich in [57], auf dessen Basis HASCASL von der IFIP WG1.3 als offizielle CASL-Erweiterung angenommen worden ist. Die Semantik von HASCASL basiert auf einer Weiterentwicklung der Semantik des partiellen  $\lambda$ -Kalküls, die in [48, 49, 51] dargestellt ist.

### 2.2.3 Coalgebraische Modallogik

Die algebraisch-coalgebraische Spezifikationssprache CoCASL erweitert die Sprache CASL (s. 2.2.1) um Konstrukte zur Spezifikation von reaktiven Prozesstypen [31]. Unterstützt werden derartige Prozesstypen insbesondere durch eine Modallogik (vgl. 2.1.3), die sich gegenüber bisher verwendeten coalgebraischen Modallogiken (etwa [13, 19, 46]) dadurch auszeichnet, dass für eine sehr allgemeine Klasse von Prozesstypen Modaloperatoren auf einheitliche Weise extrahiert werden. Diese Methode wird in [50] verallgemeinert und in Beziehung gesetzt zur bisherigen rein axiomatischen Behandlung von Modaloperatoren [36, 37]. Diese Extraktionsmethode bzw. eine geeignete Modifikation derselben könnte u.U. zur Verbesserung der Ausdruckmächtigkeit der monadischen dynamischen Logik herangezogen werden; s. 3.2.1.

### 2.2.4 Beweisunterstützung in Isabelle/HOL

Die Bremer CoFI-Arbeitsgruppe verfügt über Erfahrung in der Implementierung von sowohl Spracheinbettungen nach Isabelle/HOL als auch taktischer Beweisunterstützung für eingebettete Sprachen. Insbesondere sind in Bremen auf diesem Wege Beweisumgebungen für den gesamten Sprachumfang von CASL [28] sowie für Teilsprachen von HASCASL [9], CoCASL [10] und ModalCASL [30] erstellt worden. Hierbei hat sich Isabelle durch seine gute Unterstützung für Taktikimplementierungen, seine relative Sprachmächtigkeit sowie seine Architektur, die, gegeben die Korrektheit eines kleinen Systemkerns, die Korrektheit des Gesamtsystems garantiert, als besonders geeignetes Beweiswerkzeug für ein breites Spektrum an Logiken erwiesen.

### 2.2.5 Monadische Berechnungslogiken

Das Thema der Spezifikation von durch monadische Programmierung realisierten funktional-imperativen Programmen war bisher nur in Anfängen bearbeitet (s. 2.1.1). Im Rahmen des Bremer DFG-Projekts HasCASL (s. 2.2.2) ist sowohl eine Floyd-Hoare-Logik [54] als auch eine dynamische Logik (oder Auswertungslogik) [56] für monadische Programme entwickelt worden, wobei die Nachteile

der bisherigen Ansätze (s. 2.1.1) vermieden werden konnten. Insbesondere wurde sowohl für die Floyd-Hoare-Logik als auch für die Auswertungslogik eine direkt über einer Monade definierte Semantik angegeben; dabei konnte im letzteren Falle durch eine axiomatische Methode der lokale, d.h. zustandsabhängige Charakter der Semantik erhalten werden. Die entstehenden Logiken eignen sich gut zur modularen Spezifikation von Monaden (s. auch [21]); als Anwendungsbeispiel wird mit Hilfe der jeweiligen Kalküle in [54, 56] die partielle Korrektheit bzw. die Terminierung für Dijkstras nichtdeterministische Implementierung des euklidischen Algorithmus [6] bewiesen. Dieser Ansatz ist weiterentwickelt worden um Korrektheitsaussagen für irreguläre Terminierung, was Korrektheitsbeweise für Programme erlaubt, die, wie etwa in Java üblich, gezielt mit Ausnahmemechanismen arbeiten [55].

Diese Ergebnisse stellen einen wesentlichen Fortschritt in Hinsicht auf methodologische Fragen der Softwarespezifikation im allgemeinen dar, da hiermit ein Rahmenwerk zur Verfügung steht, in dem imperative Programmierparadigmen ohne den Umweg über Interfacesprachen einer formal fundierten Spezifikation und Entwicklung zugänglich werden. Der Ausbau dieser Methoden ist Gegenstand des hier beantragten Projekts.

## 3 Ziele und Arbeitsprogramm

### 3.1 Ziele

Die zunehmende Übernahme kritischer Funktionen durch Software, etwa im Bereich der Steuerung sicherheitskritischer Systeme oder auch der Durchführung hoheitlicher Aufgaben, bringt wachsende Ansprüche an die Korrektheit solcher Software mit sich. Während traditionell die Korrektheit von Software allenfalls durch Tests von notwendigerweise begrenzter Aussagekraft sichergestellt wird, wird in Zukunft die mathematisch rigide Verifikation von Software einen immer breiteren Raum einnehmen.

Hierbei ergibt sich hinsichtlich der bei der Softwareentwicklung zu verwendenden Programmiersprachen und -paradigmen ein Konflikt zwischen der formalen Verifikation von leichter zugänglichen, aber weniger verbreiteten Sprachen auf der einen Seite und formal weniger gut handhabbaren, aber breit durchgesetzten Sprachen auf der anderen Seite. In die erstere Kategorie fallen insbesondere logische und funktionale Programmiersprachen, in die zweite die meisten imperativen oder objektorientierten Sprachen. Die Entwicklung logischer Rahmenwerke zur Verifikation von imperativen Sprachen stellt insofern eine akute Herausforderung dar. Hierzu existiert eine gewisse Tradition in Gestalt von Programmlogiken wie der Floyd-Hoare-Logik [11] oder der dynamischen Logik [40], die in vielfacher Form weiterentwickelt worden sind [5, 17]. Als erhebliches Hindernis erweist sich dabei, dass Programmlogiken nur schwer an konkrete Programmiersprachen

anzupassen oder um neue Sprachaspekte zu erweitern sind.

Es hat sich in neuerer Zeit gezeigt, dass das imperative Programmierparadigma weitgehend in funktionale Sprachen integrierbar ist. Das geeignete Mittel hierfür ist die Verkapselung von Seiteneffekten durch sogenannte Monaden; s. 2.1.1. Da allgemeine Monaden in diesem Sinne generische Seiteneffekte repräsentieren, öffnet dies einen Weg zur angemessenen allgemeinen Behandlung von seiteneffektbehafteten Programmen auch auf der Ebene von Programmlogiken. Erste Ergebnisse in dieser Hinsicht sind in 2.2.5 zusammengefasst. Ziel des hier beantragten Projekts ist der Ausbau der so entstandenen monadischen Berechnungslogiken sowohl auf theoretischer Ebene als auch hinsichtlich Werkzeugunterstützung insbesondere durch halbautomatische Beweiser.

**Verbreiterung der logischen Basis** Die in [54, 56] eingeführten monadischen Berechnungslogiken sind aus technischen Gründen nicht durchweg von der gewünschten Allgemeinheit. So werden bestimmte Formen sehr „wilder“ Seiteneffekte, insbesondere die in Fortsetzungsmonaden kodierbaren Sprünge und nebenläufigen Prozesse u.ä., nicht oder nur ungenügend abgedeckt. Weitere offene Fragen betreffen die Natur der verwendeten Modaloperatoren, insbesondere vor dem Hintergrund jüngster Ergebnisse über coalgebraische Modaloperatoren [36, 50]. Es steht zu erwarten, dass etwa die Expressivität von Instanzen der generischen Logiken für probabilistische Programme durch eine allgemeinere Definition der verwendeten dynamischen Modaloperatoren zu verbessern ist.

**Vertiefung der logischen Basis** In [54, 56] werden für die dort eingeführten monadischen Berechnungslogiken jeweils Korrektheitssätze für die angegebenen Beweissysteme aufgestellt. Für die Entwicklung von Beweisunterstützung sind weiterführende metatheoretische Fragen wie etwa nach der Vollständigkeit dieser Beweissysteme und nach der Ausdrucksmächtigkeit der verwendeten logischen Sprachen relevant; die Bearbeitung solcher Probleme wird voraussichtlich auch eine Ergänzung und Verbesserung der logischen Systeme nach sich ziehen.

**Beweisunterstützung und Fallstudien** Die Anwendbarkeit logischer Formalismen in der praktischen Softwareentwicklung setzt zwingend die Verfügbarkeit geeigneter automatischer oder halbautomatischer Beweisunterstützung voraus. Die Implementierung einer Beweisumgebung für die bereits existierenden bzw. im Rahmen des Projekts zu entwickelnden monadischen Berechnungslogiken im halbautomatischen Theorembeweiser Isabelle/HOL ist eins der Hauptziele dieses Projekts. Innerhalb einer solchen Beweisumgebung sollen als Fallstudien Korrektheitsbeweise für funktional-imperative Programme mit verschiedenen Arten und Graden von Seiteneffekten geführt werden, die als Test für die Anwendbarkeit des logischen Rahmenwerks dienen und gegebenenfalls richtungsweisend für dessen Ausbau sind.

Am Ende des Projekts steht eine logische Umgebung zur Verfügung, die ein breites Spektrum an Seiteneffekten abdeckt und über automatische und halbautomatische Beweismechanismen verfügt. Diese Umgebung eignet sich sowohl als Basis für die axiomatische Behandlung der Semantik imperativer und objektorientierter Programmiersprachen als auch für die direkte Verifikation imperativer und nebenläufiger Programme unter weitestgehender Verkapselung von Seiteneffekten.

## 3.2 Arbeitsprogramm

Die im Projekt anfallenden Arbeiten teilen sich dem Charakter nach in Arbeiten an der logischen Basis, d.h. Weiterentwicklung existierender monadischer Berechnungslogiken und Erstellung neuer Formalismen, und Arbeiten an der Beweisunterstützung. Mit der Arbeit an der Beweisunterstützung kann, aufbauend auf die Resultate von [54, 56], unmittelbar begonnen werden; die Beweisumgebung kann dann entsprechend dem Fortgang der Entwurfsarbeit erweitert werden.

### 3.2.1 Logikentwurf

**Axiomatisierung und Extraktion von Modalitäten** In [54, 56] wird auf fest definierte Weise ein globalisierter Modaloperator aus einer gegebenen Klasse von Seiteneffekten (d.h. aus einer gegebenen Monade) extrahiert; hierauf werden alle weiteren Definitionen aufgebaut. Wie in 2.1.3 und 2.2.3 dargelegt, lassen sich in der teilweise verwandten coalgebraischen Modallogik allgemeine Modaloperatoren sowohl in axiomatischer Weise einführen als auch vollständig klassifizieren; dies führt unter anderem auf gewichtete Modalitäten etwa der Natur „mit Wahrscheinlichkeit mindestens  $p$ “ oder „mit höchstens  $n$  Ausnahmen“, die mit der bisherigen Extraktionsmethode nicht erreicht werden, die aber für Expressivitätsresultate etwa in probabilistischen Systemen von zentraler Bedeutung sind. Ein analoges Programm soll zur Verbesserung der Ausdrucksstärke auch für monadische Berechnungslogiken durchgeführt werden; eine axiomatische Behandlung von Modalitäten dürfte ferner zu einem leichteren Beweis nicht nur von Expressivitätsresultaten, sondern auch von Vollständigkeitsaussagen führen.

**Beweissysteme** Wie bereits angedeutet, fehlen (relative) Vollständigkeitsätze für monadische Berechnungslogiken bisher völlig. Die in [54, 56] angegebenen Beweissysteme zielen auch auf Vollständigkeit nicht in erster Linie ab. Insofern sollen im Rahmen dieses Projekts Vollständigkeitsresultate angestrebt werden, aller Voraussicht nach unter Ergänzung der bestehenden Kalküle. Unabhängig von solchen Resultaten werden die Kalküle ferner in Fallstudien auf Anwendbarkeit getestet und gegebenenfalls auf Basis der bestehenden Semantik entsprechend erweitert. Von Interesse ist ferner die Erfassung des lokalen Schließens im Sinne

sogenannter *frame rules* [35], die eventuell in Verbindung mit Kommutativitätsbedingungen an monadische Programme [8, 56] verallgemeinerbar sind.

**Korrektheit von zustandslosen Effekten** Die bisher entworfenen monadischen Berechnungslogiken sind insofern der traditionellen Programmlogik „zu“ verwandt, als sich ihre Aussagen (Vor- und Nachbedingungen bzw. modale Aussagen) nur auf entweder Berechnungsergebnisse oder eine abstrahierte Form von internem Zustand beziehen. Seiteneffekte, die auf diese Größen ohne Einfluß bleiben, wie etwa monadisch modellierte Ausgabe, lassen sich mit den bisherigen logischen Mitteln somit nicht vollständig erfassen, obwohl die entsprechenden Monaden grundsätzlich in das vorhandene Rahmenwerk passen. Hier soll eine möglichst allgemeine Lösung in Gestalt einer Erweiterung des logischen Formalismus gefunden werden; ein erster Schritt in dieser Richtung sind eventuell indizierte Modalitäten (s.o.).

**Berechnungslogik für die Fortsetzungsmonade** Die in [56] eingeführte monadische dynamische Logik benötigt noch gewisse Einschränkungen an die Natur der zulässigen Seiteneffekte. Insbesondere ist die monadische dynamische Logik nicht anwendbar auf die sogenannte Fortsetzungsmonade (*continuation monad*), die in der Semantik von ML und generell in der Semantik von Programmiersprachen (s. etwa [42]) sowie auch in der Behandlung von Nebenläufigkeit [2] eine wichtige Rolle spielt. Ähnliches gilt für die in [54] eingeführte monadische Hoare-Logik, die zwar im Prinzip auf beliebige Monaden anwendbar ist, aber im Falle der Fortsetzungsmonade stark an Expressivität verliert. Die Ausdehnung der monadischen Berechnungslogik auf derartige Fälle erfordert offenbar grundsätzlich neue Ideen, etwa in Gestalt neuer berechnungslogischer Formalismen spezieller oder allgemeiner Natur oder in der Form geeigneter axiomatischer Abschwächungen der existierenden Formalismen. Eine mögliche Perspektive besteht hierbei in einer monadischen Generalisierung des sogenannten Assume-Guarantee-Schließens [23].

### 3.2.2 Logikimplementierung

**Monaden in Isabelle** Anders als etwa Haskell oder HASCASL bietet Isabelle keine sogenannten Konstruktorklassen, d.h. Klassen parametrisierter Typen. Infolgedessen lassen sich allgemeine Tatsachen über Monaden in Isabelle zwar beweisen, anschließend aber nicht auf konkrete Monaden anwenden. Diese Schwierigkeit soll mit Hilfe des im DFG-Projekt AWE entwickelten Abstraktionsmechanismus (s. 2.1.4) gelöst werden.

**Partielle Funktionen** Die übliche Repräsentation partieller Funktionen in Isabelle durch unterspezifizierte total Funktionen ist semantisch inkorrekt und inso-

fern für die Implementierung monadischer Berechnungslogiken, in denen Partialität eine semantisch wichtige Rolle spielt, nicht zufriedenstellend. Erste Experimente zu einer Fehlwertkodierung partieller Funktionen höherer Stufe finden sich in [9]; dies soll zu einer vollen Kodierung des klassischen Fragments des partiellen  $\lambda$ -Kalküls erweitert werden. Die Herausforderung liegt hier vor allem im Entwurf geeigneter Beweisstrategien auf einer Ebene oberhalb der Kodierung, d.h. ohne explizite Erwähnung von Fehlwerten.

**Implementierung der monadischen Kalküle** Die Hauptaufgabe im Implementierungsanteil des Projekts liegt in der tatsächlichen Ausprogrammierung der Kalküle aus [54, 56] und der im Rahmen des Projekts zu entwerfenden weiteren bzw. erweiterten monadischen Berechnungslogiken in Isabelle/HOL. Dies beinhaltet im einzelnen die Erstellung einer Theorie von Monaden, die Definition von Syntax und Semantik der besagten Logiken, den Beweis der jeweiligen Schlußregeln, und die Entwicklung taktischer Beweisunterstützung auf der Basis dieser Schlußregeln mit dem Ziel einer möglichst weitgehenden Beweisautomatisierung.

### 3.2.3 Dissertationsthemen

Aus dem oben erläuterten Arbeitsprogramm ergeben sich mehrere mögliche Dissertationsthemen. Auf der Implementierungsseite bietet sich hier insbesondere die Erstellung der taktischen Beweisunterstützung an, die das Potential für erheblichen Erkenntnisgewinn in sich trägt. Auf der Grundlagenseite sind vor allem die Erweiterung des Formalismus auf die Fortsetzungsmonade sowie der Beweis von Vollständigkeitsresultaten für monadische Berechnungslogiken zu nennen.

### 3.2.4 Zeitplan

Ein detaillierter Ablaufplan ist in Abb. 3.2.4 dargestellt.

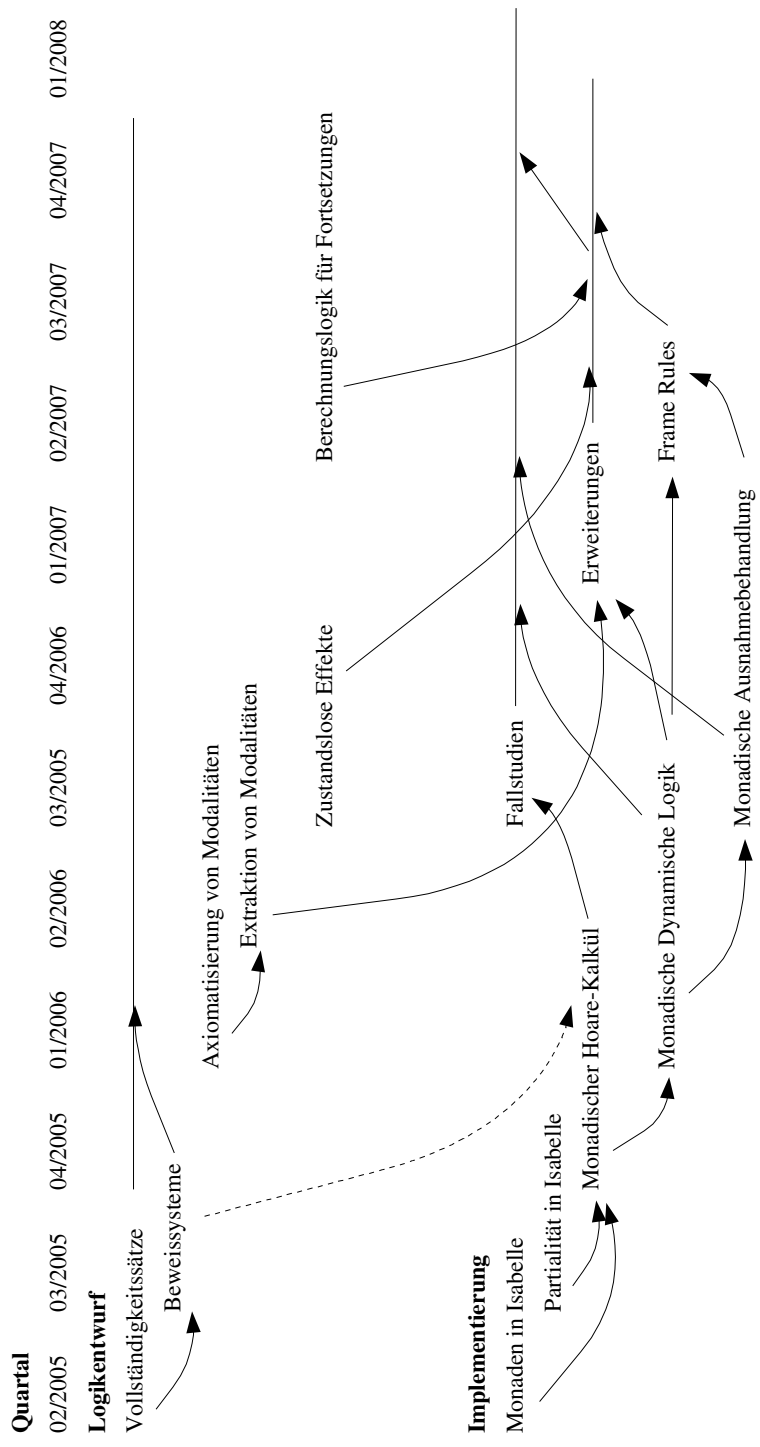
## 4 Beantragte Mittel

### 4.1 Personalbedarf

Beantragt wird eine halbe wissenschaftliche Mitarbeiterstelle (BAT IIa) für die Dauer von drei Jahren. Für die Besetzung der Stelle ist Frau Sonja Gröning vorgesehen, die zur Zeit ihre Diplomarbeit in Informatik fertigstellt, betreut vom Antragsteller und T. Mossakowski. Ferner werden für Programmieraufgaben Hilfskraftmittel im Umfang von 10 Wochenstunden beantragt, d.h.

**15600,- EUR.**

*Zeitplan „Monadische Berechnungslogiken in HOL“*



## **4.2 Wissenschaftliche Geräte**

Entfällt.

## **4.3 Verbrauchsmaterial**

Entfällt.

## **4.4 Reisen**

Entfällt

## **4.5 Publikationskosten**

Entfällt

## **4.6 sonstige Kosten**

Für die einzustellende Mitarbeiterin wird zur Durchführung der anfallenden Implementierungsaufgaben ein leistungsfähiger Arbeitsplatzrechner benötigt, der insbesondere über genügend Hauptspeicher verfügt; hierfür werden

**2000,- EUR**

beantragt.

# **5 Voraussetzungen für die Durchführung des Vorhabens**

## **5.1 Zusammensetzung der Arbeitsgruppe**

In der Arbeitsgruppe Krieg-Brückner, der ab 1.8.2005 auch der Antragsteller wieder angehört, arbeiten Dr. Till Mossakowski (DFG-Projekt MULTIPLE), Dr. Christoph Lüth (Isabelle/HOL, generische grafische Benutzerschnittstelle, transformationelle Entwicklung), Dr. George Russell (UniForM Workbench) und Dr. Paolo Torrini (DFG-Projekt SafeRobotics; modale Logiken) an Fragestellungen, die eng mit dem beantragten Projekt zusammenhängen.

## **5.2 Zusammenarbeit mit anderen Wissenschaftlern**

1. Prof. Dr. Donald Sannella und Dr. David Aspinall, Universität Edinburgh (Funktionale Programmierung, Sprachentwurf)
2. Dr. Burkhard Wolff, Universität Freiburg (Isabelle)

3. Dr. Dieter Hutter, Serge Autexier, DFKI GmbH, Saarbrücken (Werkzeugentwicklung)

### **5.3 Auslandsbezug**

Internationale Kooperationen, die mit dem beantragten Vorhaben in Zusammenhang stehen, ergeben sich aus der Mitarbeit in der CoFI Working Group und der IFIP WG 1.3.

### **5.4 Apparative Ausstattung**

Entfällt.

### **5.5 Laufende Mittel für Sachausgaben**

Laufende Kosten für Schreibbedarf etc. werden im Rahmen vorhandener Haushaltsmittel des Antragstellers bzw. der AG Krieg-Brückner von der Universität Bremen zur Verfügung gestellt.

### **5.6 Sonstige Voraussetzungen**

Schreib- und Organisationsarbeiten können vom Sekretariat des Antragstellers bzw. der AG Krieg-Brückner in angemessenem Umfang übernommen werden.

## **6 Erklärungen**

### **6.1**

Ein Antrag auf Finanzierung dieses Vorhabens wurde bei keiner anderen Stelle eingereicht.

## **7 Unterschrift**

Bremen, den 28.10.2004

Lutz Schröder

## 8 Verzeichnis der Anlagen

- Lebenslauf
- Publikationsverzeichnis
- Sonderdruck [56] und Publikationsmanuskripte [54, 55]
- Personalfragebogen Sonja Gröning

# Literatur

- [1] M. Bidoit and P. D. Mosses. *CASL User Manual*, volume 2900 of *LNCS, IFIP Series*. Springer, 2003.
- [2] K. Claessen. A poor man’s concurrency monad. *J. Funct. Programming*, 9:313–323, 1999.
- [3] K. Claessen and J. Hughes. Testing monadic code with QuickCheck. In *Haskell Workshop*, pages 65–77. ACM, 2002.
- [4] CoFI. The Common Framework Initiative for algebraic specification and development, electronic archives. Notes and Documents accessible by WWW<sup>1</sup> and FTP<sup>2</sup>.
- [5] P. Cousot. Methods and logics for proving programs. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 15, pages 841–993. Elsevier, 1990.
- [6] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.
- [7] J.-C. Filliâtre. Proof of imperative programs in type theory. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Types for Proofs and Programs*, volume 1657 of *LNCS*, pages 78–92. Springer, 1999.
- [8] C. Führmann. Varieties of effects. In M. Nielsen and U. Engberg, editors, *Foundations of Software Science and Computation Structures, FOSSACS 2002*, volume 2303 of *LNCS*, pages 144–158. Springer, 2002.
- [9] S. Gröning. Beweisunterstützung für HASCASL in Isabelle/HOL. Master’s thesis, Universität Bremen. In preparation.
- [10] D. Hausmann, T. Mossakowski, and L. Schröder. Iterative coinduction for CoCASL in Isabelle/HOL. Technical report, Universität Bremen, 2004.
- [11] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12:576–580, 1969.
- [12] B. Jacobs. Towards a duality result in the modal logic of coalgebras. In H. Reichel, editor, *Coalgebraic Methods in Computer Science, CMCS 2000*, volume 33 of *Electron. Notes Theoret. Comput. Sci.* Elsevier, 2000.
- [13] B. Jacobs. Many-sorted coalgebraic modal logic: a model-theoretic study. *Theor. Inform. Appl.*, 35:31–59, 2001.
- [14] B. Jacobs and E. Poll. Coalgebras and Monads in the Semantics of Java. *Theoret. Comput. Sci.*, 291:329–349, 2003.
- [15] E. Johnsen and C. Lüth. Theorem reuse by proof term transformation. In K. Slind, A. Bunker, and G. Gopalakrishnan, editors, *International Conference on Theorem Proving in Higher-Order Logics, TPHOLs 2004*, volume 3223 of *LNCS*, pages 152–167. Springer, 2004.
- [16] B. Klin, P. Hoffman, A. Tarlecki, L. Schröder, and T. Mossakowski. Checking amalgamability conditions for CASL architectural specifications. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science, MFCS 2001*, volume 2136 of *LNCS*, pages 512–523. Springer, 2001.
- [17] D. Kozen and J. Tiuryn. Logics of programs. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 14, pages 789–840. Elsevier, 1990.
- [18] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. PhD thesis, Universität München, 2000.
- [19] A. Kurz. Specifying coalgebras with modal logic. *Theoret. Comput. Sci.*, 260:119–138, 2001.
- [20] A. Kurz. Logics admitting final semantics. In M. Nielsen and U. Engberg, editors, *Foundations of Software Science and Computation Structures, FOSSACS 2002*, volume 2303 of *LNCS*, pages 238–249. Springer, 2002.

---

<sup>1</sup><http://www.brics.dk/Projects/CoFI>

<sup>2</sup><ftp://ftp.brics.dk/Projects/CoFI>

- [21] C. Lüth and N. Ghani. Monads and modularity. In A. Armando, editor, *Frontiers of Combining Systems, FroCoS 2002*, volume 2309 of *LNAI*, pages 18–32. Springer, 2002.
- [22] C. Lüth, M. Roggenbach, and L. Schröder. CCC - the CASL consistency checker. In J. Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques, 17th International Workshop, WADT 2004, selected papers*, LNCS. Springer, 2004. To appear.
- [23] J. Misra and K. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7:417–426, 1981.
- [24] E. Moggi. Categories of partial morphisms and the  $\lambda_p$ -calculus. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Programming*, volume 240 of *LNCS*, pages 242–251. Springer, 1986.
- [25] E. Moggi. *The Partial Lambda Calculus*. PhD thesis, University of Edinburgh, 1988.
- [26] E. Moggi. Notions of computation and monads. *Inform. and Comput.*, 93:55–92, 1991.
- [27] E. Moggi. A semantics for evaluation logic. *Fund. Inform.*, 22, 1995.
- [28] T. Mossakowski. *CASL Tools*. Chapter 11 of [1].
- [29] T. Mossakowski. HETCASL - heterogeneous specification. Language summary, 2004.
- [30] T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2004.
- [31] T. Mossakowski, L. Schröder, M. Roggenbach, and H. Reichel. Algebraic-co-algebraic specification in CoCASL. *Journal of Logic and Algebraic Programming*. To appear.
- [32] P. D. Mosses, editor. *CASL Reference Manual*, volume 2960 of *LNCS, IFIP Series*. Springer, 2004.
- [33] T. Nipkow. Hoare logics in Isabelle/HOL. In *Proof and System-Reliability*, pages 341–367. Kluwer, 2002.
- [34] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- [35] P. O’Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *Computer Science Logic (CSL 01)*, volume 2142 of *LNCS*, pages 1–19. Springer, 2001.
- [36] D. Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame J. Formal Logic*. To appear.
- [37] D. Pattinson. *Expressivity Results in the Modal Logic of Coalgebras*. PhD thesis, University of Munich, 2001.
- [38] D. Pattinson. Semantical principles in the modal logic of coalgebras. In A. Ferreira and H. Reichel, editors, *Symposium on Theoretical Aspects of Computer Science, STACS 2001*, volume 2010 of *LNCS*, pages 514–526. Springer, 2001.
- [39] A. Pitts. Evaluation logic. In *Higher Order Workshop, Workshops in Computing*, pages 162–189. Springer, 1991.
- [40] V. Pratt. Semantical considerations on Floyd-Hoare logic. In *Foundations of Computer Science*, pages 109–121. IEEE, 1976.
- [41] F. Regensburger. HOLCF: Higher order logic of computable functions. In E. Schubert, P. Windley, and J. Alves-Foss, editors, *Theorem Proving in Higher Order Logics, TPHOLs 1995*, volume 971 of *LNCS*, pages 293–307, 1995.
- [42] J. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.
- [43] M. Roggenbach and L. Schröder. Towards trustworthy specifications I: Consistency checks. In M. Cerioli and G. Reggio, editors, *Recent Trends in Algebraic Specification Techniques, 15th International Workshop, WADT 2001*, volume 2267 of *LNCS*. Springer, 2001.

- [44] M. Roggenbach, L. Schröder, and T. Mossakowski. Specifying real numbers in CASL. In D. Bert, C. Choppy, and P. D. Mosses, editors, *Recent Developments in Algebraic Development Techniques, 14th International Workshop, WADT 1999, selected papers*, volume 1827 of *LNCS*, pages 146–161. Springer, 2000.
- [45] M. Roggenbach, L. Schröder, and T. Mossakowski. Libraries. 2004. Part V of [32].
- [46] J. Rothe, H. Tews, and B. Jacobs. The Coalgebraic Class Specification Language CCSL. *J. Universal Comput. Sci.*, 7:175–193, 2001.
- [47] J. Rutten. Universal coalgebra: A theory of systems. *Theoret. Comput. Sci.*, 249:3–80, 2000.
- [48] L. Schröder. Classifying categories for partial equational logic. In R. Blute, editor, *Category Theory and Computer Science, CTCS 2002*, volume 69 of *Electron. Notes Theoret. Comput. Sci.* Elsevier, 2003.
- [49] L. Schröder. Henkin models of the partial  $\lambda$ -calculus. In M. Baaz and J. Makowsky, editors, *Computer Science Logic, CSL 2003*, volume 2803 of *LNCS*, pages 498–512. Springer, 2003.
- [50] L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. Technical report, Universität Bremen, 2004.
- [51] L. Schröder. The logic of the partial  $\lambda$ -calculus with equality. In J. Marcinkowski and A. Tarlecki, editors, *Computer Science Logic, CSL 2004*, volume 3210 of *LNCS*, pages 385–399. Springer, 2004.
- [52] L. Schröder and T. Mossakowski. HASCASL: Integrated specification and development of functional programs. In preparation.
- [53] L. Schröder and T. Mossakowski. HASCASL: towards integrated specification and development of functional programs. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methodology And Software Technology, AMAST 2002*, volume 2422 of *LNCS*, pages 99–116. Springer, 2002.
- [54] L. Schröder and T. Mossakowski. Monad-independent Hoare logic in HASCASL. In M. Pezzè, editor, *Fundamental Approaches to Software Engineering, FASE 2003*, volume 2621 of *LNCS*, pages 261–277. Springer, 2003.
- [55] L. Schröder and T. Mossakowski. Generic exception handling and the Java monad. In C. Rattray, S. Maharaj, and C. Shankland, editors, *Algebraic Methodology and Software Technology, AMAST 2004*, volume 3116 of *LNCS*, pages 443–459. Springer, 2004.
- [56] L. Schröder and T. Mossakowski. Monad-independent dynamic logic in HASCASL. *Journal of Logic and Computation*, 14:571–619, 2004. Earlier version appeared in M. Wirsing, D. Pattinson, and R. Hennicker (eds.), *Recent Developments in Algebraic Development Techniques, 16th International Workshop, WADT 2002, selected papers*, volume 2755 of *LNCS*, Springer, 2002, pp. 425–441.
- [57] L. Schröder, T. Mossakowski, and C. Maeder. HASCASL – Integrated functional specification and programming. Language summary. Available at [http://www.informatik.uni-bremen.de/agbkb/forschung/formal\\_methods/CoFI/HasCASL](http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/HasCASL), 2003.
- [58] L. Schröder, T. Mossakowski, A. Tarlecki, P. Hoffman, and B. Klin. Amalgamation in the semantics of CASL. *Theoret. Comput. Sci.* To appear.
- [59] D. von Oheimb. Hoare logic for Java in Isabelle/HOL. *Concurrency and Computation: Practice and Experience*, 13:1173–1214, 2001.
- [60] P. Wadler. How to declare an imperative. *ACM Computing Surveys*, 29:240–263, 1997.