DFG-Antrag auf Gewährung einer Sachbeihilfe für das Projekt

Algebraische Spezifikation + funktionale Programmierung = Umgebung für formale Softwareentwicklung

Bernd Krieg-Brückner
Till Mossakowski
Horst Herrlich
Fachbereich 3, Mathematik/Informatik
Universität Bremen

12 May 2000

Inhaltsverzeichnis

1 Allgemeine Angaben

Antrag auf Gewährung einer Sachbeihilfe, Neuantrag

1.1 Antragsteller

Bernd Krieg-Brückner, Prof. Dr. rer. nat.
Professor für Praktische Informatik mit den Schwerpunkten
Programmiersprachen, Übersetzer und Softwaretechnik
Universität Bremen
Studiengang Informatik des Fachbereichs Mathematik/Informatik

Dienstliche Adresse
Universität Bremen
Fachbereich 3, Mathematik/Informatik
Postfach 33 04 40, 28334 Bremen
Tel. (0421)218-3660, Fax (0421)218-3054
E-Mail bkb@informatik.uni-bremen.de

Till Mossakowski, Dr. Ing.
Wissenschaftlicher Mitarbeiter im DFG-Projekt MULTIPLE
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 till@informatik.uni-bremen.de

Horst Herrlich, Prof. Dr. rer. nat.
Professor für Mathematik mit den Schwerpunkten
Topologie und Kategorientheorie
Universität Bremen
Studiengang Mathematik des Fachbereichs Mathematik/Informatik

Dienstliche Adresse
Universität Bremen
Fachbereich 3, Mathematik/Informatik
Postfach 33 04 40, 28334 Bremen
Tel. (0421)218-2409, Fax: (0421)218-4856
E-Mail herrlich@math.uni-bremen.de

1.2 Thema

Kombination von algebraischer Spezifikation und funktionaler Programmierung als Umgebung für formale Softwareentwicklung.

1.3 Kennwort

HasCASL

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

4 Jahre

1.6 Antragszeitraum

2 Jahre

1.7 Gewünschter Beginn der Förderung

1. 10. 2000
Wir wären ggf. auch mit einer Bewilligung erst im Folgejahr einverstanden.

1.8 Zusammenfassung

Die Entwicklung einer international standardisierten Familie von Spezifikationssprachen zur formalen Entwicklung von Software ist das Ziel der *Common Framework Initiative* (CoFI) der IFIP WG 1.3. Die Entwicklung der zentralen Sprache CASL (Common Algebraic Specification Language) ist abgeschlossen. In diesem Projekt soll eine Erweiterung von CASL entwickelt werden mit dem Ziel einer Verbindung mit der funktionalen Programmiersprache Haskell. Dazu muß CASL um Logik höherer Stufe (mit Syntax, formaler Semantik und Werkzeugunterstützung) erweitert werden, so daß eine ausführbare Teilsprache zu der funktionalen Programmiersprache Haskell korrespondiert. Somit entsteht erstmals eine Entwicklungsumgebung zur Spezifikation und formalen Entwicklung von Software, die eine kohärente Entwicklung von formalen Spezifikationen und ausführbaren funktionalen Programmen in *einem* Rahmen erlaubt.

1.9 Andere DFG-Projekte

Für die Antragstellung relevant sind die in 2.2.2, 2.2.6 und 2.2.7 beschriebenen Projekte.

2 Stand der Forschung, eigene Vorarbeiten

2.1 Stand der Forschung

2.1.1 Formale Spezifikation

Für die Softwareentwicklung ist zur Verbesserung von Effizienz, Fehlervermeidung und Sicherheit der Einsatz strukturierender Methoden und Werkzeuge in allen Phasen der Entwicklung (Modellierung, Anforderungsdefinition, Entwurf) erforderlich. Für semi-formale Methoden wie UML [Obj99] gibt es inzwischen Werkzeuge zur Anbindung an Code-Generierung [I-L, Rat]; allerdings ist z.B. für sicherheitskritische Bereiche der Einsatz vollständig formaler Spezifikationssprachen erforderlich. Diese ermöglichen, auf Grundlage einer mathematisch orientierten Modellierung, zu einer präzisen Definition der Anforderungen an die Funktionalität eines Systems zu gelangen. Solche Spezifikationssprachen müssen durch eine mathematisch fundierte formale Semantik abgesichert sein; auf diesem Wege erhält man gleichzeitig eine angemessene Definition der Korrektheit einer Implementierung bzgl. einer Spezifikation. Die Korrektheit kann dann bewiesen oder durch die Benutzung von Transformationen garantiert werden.

Im Bereich der formalen Methoden zur Spezifikation und Entwicklung von *function strong* und *data strong* Programmen waren die Sprachen Larch

[GHG+93] und das VSE-2-Projekt [HMR+98] mit einer Reihe von industrierelevanten Fallstudien erfolgreich. VSE umfaßt im Datentyp-orientierten Bereich eine algebraischen Spezifikationssprache und die Anbindung einer imperativen Programmiersprache. Allerdings geht bei VSE-2 der Weg von der algebraischen Spezifikation zum ausführbaren Programm über die Zwischenstufe einer dynamischen Logik (auch bei Larch ist eine sog. Interface-Sprache erforderlich).

Eine wesentlich direktere Anbindung einer Programmiersprache versucht Extended ML [KST97]: hier ist es möglich, Spezifikationen und ML-Programme zu mischen und schrittweise in Richtung ausführbares Programm zu entwickeln. Dies wird durch die Wahl von ML ermöglicht: ML ist als *funktionale* Programmiersprache näher an der Spezifikationsebene als eine imperative Sprache. Die Spezifikationssprache Extended ML hat sich jedoch als zu kompliziert erwiesen, um in der Praxis anwendbar zu sein. Dies hängt vor allem damit zusammen, daß Extended ML auf die relativ komplexe Semantik der Programmiersprache ML aufsetzt. Als Folge gibt es keine praktisch brauchbare Werkzeugunterstützung für Extended ML.

HOLCF [Reg95, MNvOS99] ist als Einbettung von LCF [Pau87] in Isabelle/HOL [Pau94] eine handhabbarere Logik mit besserer Beweisunterstützung (durch Isabelle/HOL) als Extended ML. Allerdings ist die ausführbare Zielsprache LCF keine Programmiersprache, sondern eine Sprache zur Semantikdefinition von Programmiersprachen. So werden in HOLCF auch nur nur relativ einfache Formen von Rekursion und Pattern-Matching unterstützt. Zudem stehen als Strukturierungsmechanismen nur diejenigen des Theorembeweisers Isabelle zur Verfügung.

Einen anderen Weg geht CaféOBJ [DF98], eine direkt ausführbare Spezifikationssprache. Das Problem des Übergangs zu einer Programmiersprache stellt sich hier nicht, da CaféOBJ (bzw. die eng verwandte Sprache Maude [C+99]) selbst ausführbar ist. Die direkte Ausführbarkeit geht allerdings zu Lasten der Ausdrucksmächtigkeit; deshalb ist CaféOBJ für nichtalgorithmische problemorientierte Anforderungsspezifikationen ungeeignet.

Als internationaler Standard für algebraische Spezifikationsprachen wurde inzwischen im Rahmen der *Common Framework Initiative* [Mos97, CoF] der IFIP WG 1.3 die Sprache CASL (*Common Algebraic Specification Language*) entwickelt [CoF98], die sich durch eine relativ hohe Ausdrucksmächtigkeit und eine relativ einfache algebraische Semantik auszeichnet; s. hierzu auch 2.2.4. Dieser Standard soll künftig auch von den VSE-2 zugrundeliegenden Werkzeugen (KIV [R+98], Inka [AHMS99]) verwendet werden.

2.1.2 Haskell

Haskell [Tho96, Bir98] ist eine funktionale Programmiersprache, die sich dadurch auszeichnet, daß sie einerseits "rein" ist, d.h. keine nicht-funktionalen

Elemente enthält, andererseits aber auch für größere Software-Projekte [Kar98] eingesetzt worden ist. Haskell ist, ähnlich wie CASL, das Ergebnis von Standardisierungsbemühungen der in diesem Gebiet aktiven Forschergemeinschaft und wird daher durch eine Reihe ausgereifter Compiler unterstützt.

Neben Standardelementen funktionaler Sprachen wie Polymorphie, rekursiven Typen und Funktionen höherer Ordnung bietet Haskell Typklassen [HHPW96, Jon00], insbesondere Monaden, über die unter anderem die Ein-/Ausgabe realisiert wird. Zur Zeit ist mit Haskell98 ein stabiler Standard erreicht [JHA+99]. Wichtige Varianten sind Haskell B, das, wie auch der Glasgow Haskell Compiler und der Interpreter Hugs, existentielle Typen unterstützt [Läu96], und Concurrent Haskell [PGF96], eine rein funktionale Erweiterung von Haskell um nebenläufige Anteile; s. hierzu auch 2.2.5. Die Sprache O'Haskell [Nor99, Nor00] ist eine Erweiterung von Haskell um Nebenläufigkeit und Subtypen, wobei die Subtypen zur Modellierung von objekt-orientierten Vererbungsrelationen benutzt werden.

Haskell ist trotz seiner "puren" funktionalen Basis effizient ausführbar, und gerade wegen dieser Basis auch gut parallelisierbar [THM+96]. Zudem besitzt Haskell mit H/Direct [FLMJ99] eine moderne Fremdsprachenschnittstelle, die eine Einbindung über die Schnittstellendefinitionssprache IDL gestattet; hierdurch ist unter anderem eine Anbindung an C gegeben. Ferner existiert eine Übersetzung von Haskell in die *Java Virtual Machine* [Rei99, Wak98].

2.1.3 Logik höherer Stufe

Logik höherer Stufe ist im Vergleich zur Logik erster Stufe flexibler und erhöht die Wiederverwendbarkeit. Nicht zuletzt deshalb basieren alle erfolgreichen halbautomatischen Theorembeweiser der letzten Zeit (z.B. PVS [OSRSC99], Isabelle/HOL [Pau94], HOL [GM93], Coq [B+97]) auf Logik höherer Stufe, oder es wird zumindest an einer Unterstützung von Logik höherer Stufe gearbeitet (KIV [R+98], Inka [AHMS99]).

Die Literatur über die zugrundeliegenden Typentheorien höherer Stufe ist umfangreich, siehe z.B. [And86, Cro94, AL91, Jac99, LS86]. Insbesondere existiert eine weitverzweigte Ansammlung von Typentheorien unterschiedlicher Mächtigkeit; ein Teil dieser Theoriefamilie findet sich im sogenannten Lambda-Würfel wieder [Bar92] (s. Abbildung 1). Dieser systematisiert Erweiterungen des einfach getypten Lambda-Kalküls $\lambda\!\!\rightarrow$ um etwa Polymorphie ('2'), *Kinds* ('ω') oder abhängige Typen ('P'); an dieser Klassifikation orientiert sich auch das hier beantragte Projekt. Aufbauend auf diese typentheoretischen Ergebnisse sind in mehrere der existierenden Spezifikationssprachen Sprachelemente höherer Stufe aufgenommen worden, so etwa in RAISE [GHH+92], Spectrum [BFG+93] oder im Entwurf SPECTRAL (s.

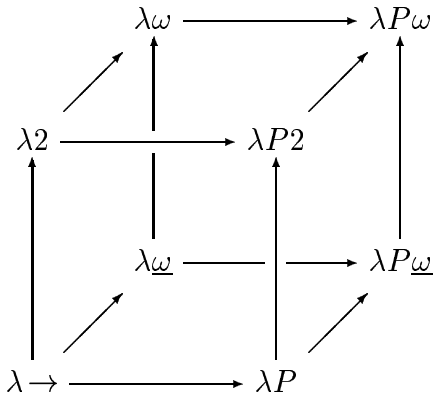

Abbildung 1: Der Lambda-Würfel

Abschnitt 2.2.1). Zu beobachten ist hier sowohl bei Spezifikations- als auch bei Beweisersprachen im allgemeinen entweder ein hoher Komplexitätsgrad, bedingt z.B. durch eine dreiwertige Logik oder eine verbandstheoretische Semantik (Spectrum, RAISE), oder eine weitgehende Beschränkung bei der Auswahl der Sprachkonzepte. So wird etwa in PVS auf Partialität und im eigentlichen Sinne polymorphe Typen verzichtet, in Isabelle/HOL auf Partialität und Subtypen und in RAISE auf Polymorphie und Prädikate.

2.2 Eigene Vorarbeiten

2.2.1 Sprachentwurf

B. Krieg-Brückner hat langjährige Erfahrungen im Entwurf von sowohl Programmier- als auch Spezifikationssprachen (Ada [IBH+79], PAnndA-S im ESPRIT Projekt PROSPECTRA [HKB93], SPECTRAL [KBS91]). Ideen aus dem Entwurf SPECTRAL sind zum Teil in den Entwurf der Sprache CASL (siehe 2.2.4) eingeflossen. Die in der Forschungsgruppe Krieg-Brückner entstandene Dissertation [Liu98] diskutiert einen Modellinferenzmechanismus für Klassenpolymorphie in Spectral.

2.2.2 Von der Spezifikation zum Programm

Im DFG-Projekt EXSPEC (Dr. Z. Qian, AG Krieg-Brückner, vgl. [QW92, QW94, Qia94, Qia95, Wan94]) ist untersucht worden, inwieweit sich die in den letzten Jahren neu entwickelten Spezifikationsparadigmen erster und höherer Stufe auf ihre ausführbaren Kerne mit relativ gutem operationalen Verhalten beschränken und sich aus solchen Kernen ein logisches Programmierparadigma mit Nebenbedingungen (*Constraints*) gewinnen läßt. Der An-

satz unterstützt das *rapid prototyping* von Spezifikationen und kann die Entwurfsphase von Programmen deutlich verkürzen.

Im Rahmen des BMBF-Projekts UniForM [Kar99, KB99, KPO+99] sind ferner auf der Basis von Logik höherer Stufe Entwurfstransformationen (zum Teil innerhalb von PROSPECTRA entwickelt; vgl. Abschnitt 2.2.1) implementiert worden [KSW96], die geeignet sind, Spezifikationen in ausführbare Spezifikationen zu überführen.

Das generische Bremer *Transformation Application System* (TAS) [LW] erlaubt die schrittweise Verfeinerung von Anforderungsspezifikationen zu (ausführbaren) Entwurfsspezifikationen mittels Transformationen. Es basiert auf dem generischen Theorembeweiser Isabelle [Pau94], der die logische Korrektheit und Konsistenz von Transformation und Entwicklung gewährleistet. Seine Generizität erlaubt es, das System schnell für den Einsatz im Zusammenhang mit CASL bzw. HasCASL zu konfigurieren.

In Verbindung zu TAS steht der DFG-Antrag AWE (C. Lüth, unter Mitwirkung von B. Krieg-Brückner), der auf die Wiederverwendung von Transformationen und Beweisen abzielt.

2.2.3 Grundlagen der Systemspezifikation

B. Krieg-Brückner war Koordinator der ESPRIT Basic Research Working Group COMPASS (A COMPrehensive Algebraic approach to System Specification and development, 1989-96) mit insgesamt 20 europäischen Partnern. Diese Arbeitsgruppe diente europäischen Wissenschaftlern aus dem Gebiet der eigenschaftsorientierten Spezifikationen als Forum zur Diskussion und Verbreitung theoretischer Fragestellungen und Ergebnisse [BKL+91, CGK+97, GC95, KB89, KB90, KBe91, KB96, KBQ96].

Fortgesetzt wird die Arbeit von COMPASS in der Working Group 1.3 (Foundations of Systems Specification) der International Federation of Information Processing (IFIP). Ein wichtiges Arbeitsergebnis der Working Group ist der State-of-the-Art-Report „Algebraic Foundations of Systems Specification“ [AKKB99], herausgegeben von E. Astesiano, H.-J. Kreowski und B. Krieg-Brückner. Der Band enthält zwei Kapitel von Koautoren aus der Forschungsgruppe Krieg-Brückner [CMR99, BKB99], deren Thematik in engem Zusammenhang zum beantragten Vorhaben steht.

2.2.4 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 im Rahmen der Working Group COMPASS (s. 2.2.3) und 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)). In diese Entwicklung sind weitere internationale Arbeitsgruppen einbezogen worden, um eine maximale Verbreitung zu erreichen. Seit 1998 wird die CoFI Working Group mit Reisemitteln durch die Europäische Union gefördert.

Die von der Arbeitsgruppe entwickelte Kernsprache CASL (*Common Algebraic Specification Language*) enthält bewußt nur bereits lange theoretisch untersuchte und gut verstandene Konzepte. Ein Ziel von CoFI ist es, um diese zentrale Sprache herum eine Familie von unterschiedlich komplexen Sprachen für verschiedene Zwecke zu entwickeln. Dies ermöglicht es, Sprachen und vor allem auch vorhandene Werkzeuge zueinander in Relation zu setzen und, nach geeigneter Übersetzung, in einem gemeinsamen Rahmen zu verwenden. Der Entwurf von CASL [CoF98] sowie die Definition von Teilsprachen [Mos97a] konnten 1998 abgeschlossen werden. CASL erweitert Prädikatenlogik erster Stufe um Subtypen und Partialität; dadurch werden Spezifikationen kompakter und lesbarer. Als nächstes sollen, neben der im Rahmen des hier beantragten Projektes geplanten erweiterten Sprache höherer Stufe, reaktive und objekt-orientierte Erweiterungen von CASL entwickelt werden.

B. Krieg-Brückner trägt innerhalb von CoFI zentrale Verantwortung als Leiter der CoFI-Sprachentwurfsgruppe. Seine Forschungsgruppe, insbesondere Dr. T. Mossakowski, hat an der Semantik von CASL mitgearbeitet [CoF99] und sowohl mit internen Papieren [Mos97a, Mos98d, RM99, RMS00, Mos97b, Mos98a, Mos98e, Mos98c] und internationalen Veröffentlichungen [CHKBM97, Mos98b, MKK98, ABKB$^+$, MHKB00, Mos00a, Mos00b, Mos00c, RSM00] als auch insbesondere durch die Implementierung von Werkzeugen (Parser, statischer Checker u.ä.; vgl. Abbildung 2), die innerhalb von CoFI mittlerweile einen De-facto-Standard darstellen, zur Entwicklung von CASL wesentlich beigetragen.

Zur Erweiterung von CASL in Richtung auf eine Integration von Sprachelementen höherer Stufe liegt ein unter Mitwirkung der Forschungsgruppe Krieg-Brückner entstandener erster Entwurf vor, in dem eine Syntax und Semantik von Funktions- und Prädikatentypen unter Beibehaltung von sowohl Partialität als auch Subtypen vorgeschlagen werden; die Definition der Semantik erfolgt hierbei durch schrittweise Reduktion auf einfachere Logiken [MHKB00]. Weitergehende, in der Funktionalen Programmierung wichtige Konzepte wie Polymorphie, abhängige Typen o.ä. sind hier noch nicht berücksichtigt.

2.2.5 Implementierungen in Haskell

In der Forschungsgruppe Krieg-Brückner ist 1998 im Rahmen einer Dissertation ein Projekt zur Werkzeugintegration in Haskell realisiert worden

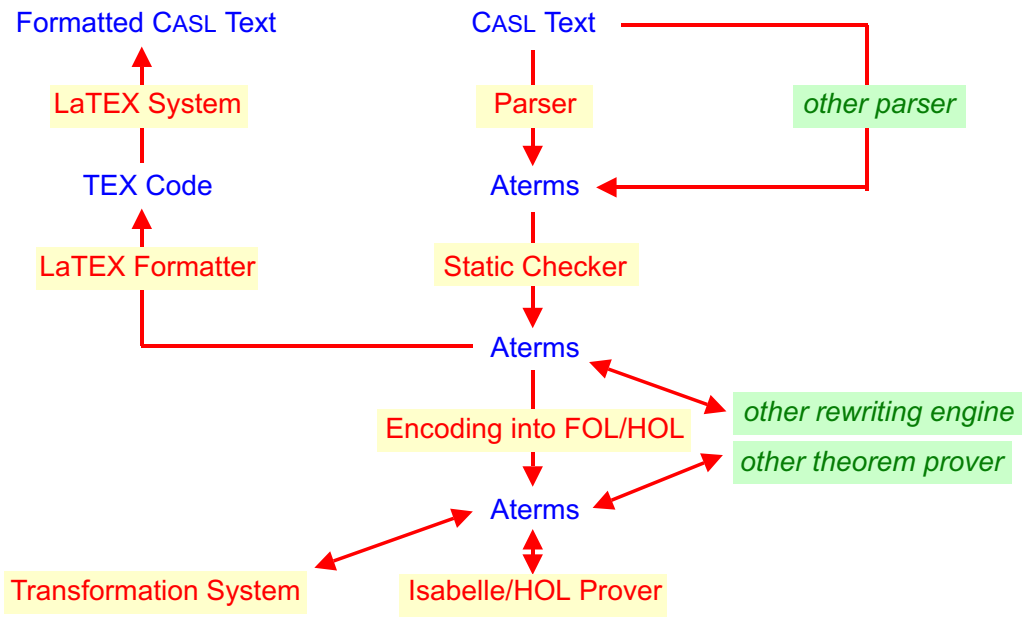

Abbildung 2: Architektur der Bremer CASL-Werkzeuge

[Kar98, Kar99, KB99]; Teil dieses Projekts ist das UniForM Concurrency Toolkit, bei dessen Entwicklung Concurrent Haskell zum Einsatz kam. Das entstandene System ist die zur Zeit vermutlich größte existierende Haskell-Anwendung (ca. 50.000 Programmzeilen); damit ist gezeigt worden, daß Haskell auch für umfangreiche Vorhaben und zur Systemprogrammierung geeignet ist.

2.2.6 Nebenläufige und objekt-orientierte Sprachen

Im DFG-Projekt COOFL (B. Krieg-Brückner, Dr. Markus Roggenbach; 11/98–10/01) wird untersucht, wie sich funktionale, nebenläufige und objekt-orientierte Spezifikation und Programmierung sowohl auf der Programmiersprachenebene als auch auf der Spezifikationssprachenebene miteinander vereinen lassen. Einerseits ist dieses Projekt orthogonal zum hier beantragten, da Spezifikation und Programmierung im wesentlichen separat betrachtet werden. Andererseits ist zu erwarten, daß sich die nebenläufigen und objekt-orientierten Erweiterungen direkt auf HasCASL übertragen lassen, insofern sie sich auf der Spezifikationsebene bewegen und sich dort auf (first-order) CASL beziehen.

2.2.7 Multi-Logik-Systeme

Im DFG-Projekt MULTIPLE (B. Krieg-Brückner, H.-J. Kreowski, T. Mossakowski, ab 7/00) wird ein Graph von Logiken entwickelt, der Teilsprachen

und Erweiterungen von CASL sowie exemplarisch andere Spezifikationssprachen umfaßt, mit denen bereits erfolgreich Anwendungen entwickelt wurden. Dieser Graph ist die Basis für eine heterogene Spezifikationssprache, in der Spezifikationen aus verschiedenen Logiken und Werkzeuge kombiniert werden können.

Die im hier beantragten Projekt zu entwickelnde Sprache HasCASL soll als CASL-Erweiterung in den Logik-Graph und die heterogene Sprache mit aufgenommen werden. Sie bildet dort den Weg in die funktionalen Programmiersprachen.

2.2.8 Kategorientheorie

In der Semantik von Spezifikationssprachen und von Logiken höherer Stufe spielt die Kategorientheorie eine zentrale Rolle (vgl. 2.1.3 sowie [Cro94, AL91, Jac99, LS86]). Obwohl die Arbeit am beantragten Projekt schwerpunktmäßig in der Forschungsgruppe Krieg-Brückner stattfinden wird, wird daher H. Herrlich zur Lösung solcher theoretischer Fragestellungen beitragen. Der vorgesehene wissenschaftliche Mitarbeiter Lutz Schröder wurde in der AG Herrlich mit einem Thema aus der Kategorientheorie promoviert [Sch99] und hat international sowohl über Kategorientheorie [HS99, SH00, Sch, SHa, SHb] als auch über CASL-bezogene Themen [RSM00, RMS00] veröffentlicht.

H. Herrlich hat durch langjährige und andauernde Forschungstätigkeit zum Aufbau der Kategorientheorie sowohl im Grundlagen- als auch im Anwendungsbereich wesentlich beigetragen. Zu nennen sind hier sowohl Standardlehrbücher wie [HS79, AHS90] als auch Beiträge zur Kategoriellen Topologie [Her68, AH86, AH90, BH98] und zur Kategoriellen Algebra [Her71, Her74, Her86, AHR88, AHT89, Her90, Her93] sowie der in engerem Zusammenhang mit der Thematik des beantragten Projekts stehende Konferenzband [EHKP89].

3 Ziele und Arbeitsprogramm

3.1 Ziele

In diesem Projekt soll eine Erweiterung von CASL entwickelt werden mit dem Ziel einer Anbindung der funktionalen Programmiersprache Haskell. Haskell bietet sich für die Verbindung mit einer algebraischen Spezifikationssprache insofern an, als es als rein funktionale Sprache (im Gegensatz etwa zu ML [Pau96]) linguistisch und semantisch nahe an den Paradigmen der algebraischen Spezifikation liegt; Haskell hat sich auch in dieser reinen Form als für größere Anwendungen geeignet erwiesen [Kar98], s. 2.2.5.

Funktionale Sprachen haben sich als Bestandteil der Lehre einschließlich des Grundstudiums breit durchgesetzt. Zudem ist unsere Erfahrung, daß

mit einer funktionalen Sprache etwa die dreifache Programmierproduktivität (verglichen mit imperativen Sprachen) erreicht werden kann. Dies führt dazu, daß auch für komplexe Aufgaben relativ schnell erste lauffähige Systeme erstellt werden können [Kar98, Mos00a]. Über Haskell-Schnittstellen zu imperativen Sprachen (vgl. Abschnitt 2.1.2) können dann später Teile in einer imperativen Sprache, etwa in C, optimiert oder hinzugefügt werden.

Sprachelemente von CASL sind Prädikatenlogik erster Stufe, Subtypen mit überladenen Operatoren, Partialität und Konstrukte zur Beschreibung termerzeugter Datentypen. Wichtige Bestandteile von Haskell fehlen jedoch in CASL. Um eine Verbindung mit Haskell zu ermöglichen, soll CASL um Logik höherer Stufe, Polymorphie, rekursive Typen, Typkonstruktoren und nicht-strikte Funktionen erweitert werden. Die so entstehende Sprache HasCASL soll mit Syntax und formaler Semantik beschrieben werden.

Um eine möglichst einfache Entwicklung von HasCASL-Spezifikationen zu funktionalen Programmen (ohne den Umweg über Zwischen-Logiken oder Interface-Sprachen) zu ermöglichen, soll eine Teilsprache von HasCASL ausgezeichnet werden, die ausführbar ist und die direkt zu der funktionalen Programmiersprache Haskell korrespondiert. Zentraler Forschungsgegenstand ist deswegen eine Konzeption von HasCASL, die die Einfachheit und Eleganz der algebraischen Semantik und zweiwertigen Logik von CASL mit einer möglichst weitgehenden Kompatibilität zur denotationalen cpo-basierten Semantik von Haskell verbindet.

Eine typische Entwicklung in HasCASL würde bei einer problemnahen, nicht-ausführbaren HasCASL-Anforderungsspezifikation starten, die dann schrittweise zu einer HasCASL- Entwurfsspezifikation verfeinert oder transformiert wird, bis schließlich die ausführbare Teilsprache erreicht wird, von der dann eine automatische Übersetzung nach Haskell möglich ist. Eine solche Methodik bietet gegenüber der strikten Trennung von Programmierung und Spezifikation, eventuell mit einer dazwischengeschalteten Interface-Spezifikation wie in Larch [GHG+93], den Vorteil, daß im Prinzip die gesamte Software-Entwicklung, ausgehend von der Anforderungsspezifikation und gegebenenfalls unter Verwendung geeigneter Transformationen, innerhalb einer Sprache stattfinden kann. Dies erlaubt insbesondere unvollständige Implementationen, bei denen nur Teile der Spezifikation durch ausführbare Abschnitte ersetzt werden, während andere Teile unimplementiert bleiben.

Somit entsteht eine Entwicklungsumgebung zur Spezifikation und formalen Entwicklung von Software, die erstmals eine kohärente Entwicklung von formalen Spezifikationen und ausführbaren funktionalen Programmen in *einem* Rahmen erlaubt. Daher ist die Entwicklung von Werkzeugen, die den Entwicklungsprozeß unterstützen, ein entscheidender und integraler Bestandteil des Projekts. Es sollen die bestehenden Analyse- und Beweiswerkzeuge für CASL auf HasCASL ausgedehnt werden, die Erkennung der ausführbaren Teilsprache (inklusive der entstehenden Beweisverpflichtungen zur Ausführ-

barkeit), die automatische Übersetzung nach Haskell und (zumindest prototypisch) halbautomatisch anzuwendende Transformationen für den Entwicklungsprozeß von der Anforderungs- zur Entwurfsspezifikation entwickelt werden. Sowohl Sprachentwurf als auch Werkzeuge sollen durch eine Fallstudie auf Praxistauglichkeit getestet werden.

3.2 Arbeitsprogramm

3.2.1 Zeitplan

Die geplanten Arbeiten sind in eine konzeptionell-theoretische und eine Werkzeugebene unterteilt, für die jeweils ein Mitarbeiter vorgesehen ist. Anhand des vorliegenden Entwurfs [MHKB00] kann mit den Arbeiten auf der Werkzeugebene ohne Verzögerung begonnen werden; der weitere Ablauf des Sprachentwurfs ist so geplant, daß die Werkzeuge stufenweise an neu entwickelte Erweiterungen angepaßt werden können (vgl. umseitiges Diagramm). Die Entwicklung einer an wesentlichen Bestandteilen von Haskell orientierten Sprache (HasCASL) einschließlich der Werkzeuganpassung soll innerhalb der ersten Zweijahresperiode abgeschlossen werden; für die zweite Periode sind die Fertigstellung der vollständigen Sprache (HasCASL B) sowie eine Fallstudie geplant.

3.2.2 Konzeptioneller Teil, Phase I

3.2.2.1 Ausführbarkeits-Begriffe Es soll als erstes ein geeigneter Ausführbarkeitsbegriff für Spezifikationssprachen entwickelt werden (zunächst basierend auf der Erweiterung von CASL um Konzepte höherer Stufe in [MHKB00]), um einen möglichst nahtlosen Übergang von einer ausführbaren Teilsprache nach Haskell zu ermöglichen. Dieser Ausführbarkeitsbegriff soll möglichst allgemeine Mechanismen zur Rekursionsbehandlung bieten. In Frage kommt hier eine Einschränkung der zulässigen Gleichungen auf die (syntaktisch leicht erkennbare) primitive Rekursion bzw. allgemeiner auf wohlfundierte Rekursion [MW, Win93], die wesentlich allgemeiner ist als die Rekursionsschemata etwa von HOLCF [Reg95]. Wohlfundierte Rekursion ist mit der Einführung geeigneter Annotationen verbunden, die entsprechende Beweisverpflichtungen erzeugen; dies erlaubt allerdings nur die Definition totaler Funktionen. Über allgemeine Rekursion definierte partielle Funktionen lassen sich behandeln, indem

- entweder gezeigt wird, daß eine ausführbare Spezifikation, aufgefaßt als Haskell-Programm, eine Fixpunkt-Semantik besitzt, die ein Modell der Spezifikation ist,

- oder mit Hilfe des in CASL vorhandenen *free*-Konstrukts (bei geeigneter Wahl des Homomorphiebegriffs) die Fixpunktsemantik direkt spezifiziert wird [Mos98d].

In jedem Falle ist eine solche Modellierung partiell rekursiver Funktionen als partielle Funktionen in CASL einfacher als die Einbindung spezialisierter Logiken wie LCF [Pau87].

Dieser Ausführbarkeitsbegriff soll dann die Basis für ausführbare Teilsprachen von HasCASL bzw. HasCASL B (siehe Abschnitt 3.2.3.4) bilden, die eine direkte (möglichst automatisierte) Übersetzung nach Haskell erlauben.

3.2.2.2 Polymorphie und Subtypen Ein wichtiges Sprachelement von Haskell sind Funktionen variablen („polymorphen“) Typs. Dieses Konzept ist von besonderer Anwendungsrelevanz, da die Wiederverwendbarkeit von Programmen bzw. Spezifikationen hierdurch in erheblichem Maße verbessert wird. Das Konzept der Subtypen führt ebenfalls zu kompakteren Spezifikationen bzw. Programmen und erlaubt Modellierung von objekt-orientierter Vererbung; Subtypen gibt es (in unterschiedlicher Form) sowohl in CASL als auch in der Haskell-Erweiterung O'Haskell. Deshalb soll CASL höherer Stufe als nächstes um Polymorphie erweitert werden. Dabei sind folgende Probleme zu lösen:

- Problematisch ist das in Haskell und ML vorhandene *where*- bzw. *let*-Konstrukt, das das Einsetzen von Elementen polymorphen Typs in weitere Ausdrücke erlaubt: im Rahmen einer zweiwertigen Logik führt dies zu Inkonsistenzen [Coq86]. Dieses Konstrukt soll daher bei der Anbindung von Haskell zunächst zugunsten einer rein parametrischen Polymorphie [Rey83] ausgeblendet werden; eventuell muß zu einem späteren Zeitpunkt eine Hilfskonstruktion eingeführt werden, etwa über eine durch partielle Funktionen in einen gesonderten booleschen Typ realisierte dreiwertige Logik für programmierungsnahe Zwecke.

- Die Kombination von Polymorphie und Subtypen ist nicht-trivial [HM95, Reh97]. HasCASL soll sich in dieser Hinsicht vor allem an den Forschungsergebnissen über die bereits implementierte Sprache O'Haskell [Nor99, Nor00] orientieren, so daß ein möglichst nahtloser Übergang von polymorphen HasCASL-Subtypen zu polymorphen O'Haskell-Subtypen möglich wird.

3.2.2.3 Rekursive Typen In funktionalen Programmiersprachen, insbesondere in Haskell, gibt es benutzerdefinierte rekursive Typen, um beliebige dynamische Datenstrukturen (z.B. Listen, Bäume) erzeugen zu können. Die in CASL erster Stufe vorhandenen Sprachkonstrukte erlauben ebenfalls

die Definition rekursiver Typen. Deshalb soll in diesem Arbeitsschritt CASL höherer Stufe in gleicher Weise mit rekursiven Typen ausgestattet werden. Insbesondere sollen wie in Haskell über Mechanismen wie Funktionenraumbildung u.ä. definierte rekursive Typen zugelassen werden. Im Rahmen einer zweiwertigen Logik (wie eben in CASL) führt dies jedoch zu Konsistenzproblemen. Für HasCASL bietet sich als Lösungsperspektive der Übergang zu einer geeignet abgeschwächten Henkin-Semantik [Hen49] an, unter der ggf. die cpo-basierte Semantik von Haskell als ein Henkin-Modell aufgefasst werden kann.

3.2.2.4 Typkonstruktoren Um Funktionen, die Standardkonstruktionen auf Typen repräsentieren (wie etwa einen Operator, der einem Typ den Typ der Listen von Elementen dieses Typs zuordnet), selbst wiederum mit einem Typ versehen zu können, ist es erforderlich, einen vorgegeben Typ einzuführen, der das „Universum aller Typen" darstellt; dieser Typ und darauf aufbauende höhere Typen (etwa Funktionstypen „Typ $\rightarrow$ Typ") heißen Arten (*Kinds*), ihre Elemente Typkonstruktoren. Typkonstruktoren spielen in Haskell eine zentrale Rolle, etwa auch im Zusammenhang mit Monaden (s.o. sowie [JHA+99]); ein entsprechendes Sprachkonstrukt soll daher auch in der geplanten CASL-Erweiterung gefunden werden. Es soll außerdem untersucht werden, ob die in CASL vorhandenen parametrisierten Spezifikationen für diesen Zweck sowie zur Modellierung der in Haskell ebenfalls wichtigen Typklassen bereits hinreichend sind (vgl. auch 2.2.1). Konstruktorklassen wären dann parametrisierte Spezifikationen mit polymorphen Typen im Parameter.

3.2.2.5 HasCASL Nach der Anwendung des in Abschnitt 3.2.2.1 entwickelten Ausführbarkeitsbegriffs auf die bisher entwickelte Sprache und der Auszeichnung der ausführbaren Teilsprache steht am Ende von Phase I eine erste Version von HasCASL, mit Polymorphie, Subtypen, rekursiven Typen, Typkonstruktoren und einer zu einer Haskell- bzw. O'Haskell-Teilmenge korrespondierenden Teilsprache, zur Verfügung.

3.2.3 Konzeptioneller Teil, Phase II

3.2.3.1 Nicht-strikte Funktionen Bedingt durch das in Haskell verfolgte Prinzip der „Lazy Evaluation" sind die in der Haskell-Semantik auftauchenden partiellen Funktionen nicht-strikt (d.h. undefinierte Argumente können definierte Resultate liefern), während die CASL-Semantik gegenwärtig nur strikte partielle Funktionen kennt. Deshalb soll HasCASL um nicht-strikte Funktionen erweitert werden. Hierzu kann ein weiterer, nicht-strikter partieller Funktionstyp zu HasCASL hinzugefügt werden, der semantisch voraussichtlich leicht durch den bestehenden strikten Typ codierbar wäre. Nicht-strikte Funktionen kommen in Haskell (neben der auch in

CASL bereits vorhandenden nicht-strikten Fallunterscheidung) vor allem in Zusammenhang mit Monaden [Mog91, Wad95] vor; Monaden sind wiederum entscheidend, um imperative Features wie Ein- und Ausgabe in eine rein funktionale Sprache wie Haskell integrieren zu können.

3.2.3.2 Existentielle Typen Existentielle Typen (oft „Summentypen" [Jac99]) sind ein geeignetes sprachliches Mittel zur Verkapselung der tatsächlichen Repräsentation abstrakter Datentypen [CW85, MP88]; implementiert ist dieses Prinzip beispielsweise in der funktionalen Programmiersprache Haskell B [Läu96], aber auch in den gängigen Haskell-Compilern wie etwa dem Glasgow Haskell Compiler [Uni]. Die existentielle Quantifizierung ermöglicht gewissermaßen die Zusicherung des Vorhandenseins von Operationen bestimmten Profils auf einem geeigneten repräsentierenden Typ, ohne diesen dabei tatsächlich festzulegen bzw. dem Benutzer zur Verfügung zu stellen; auf diesem Wege lassen sich auch objektorientierte Konzepte teilweise nachbilden [Jac99].

HasCASL soll daher um ein Konzept für existentielle Typen erweitert werden. Der Effekt existentieller Typen läßt sich auch durch das in CASL mögliche ausdrückliche Verstecken von Bezeichnern annähern; die Konstruktion über existentielle Typen ist aber möglicherweise natürlicher. Allerdings bringt die existentielle Quantifizierung von Typen ähnliche semantische Probleme mit sich wie die mit Polymorphie assoziierte Allquantifizierung. Letztlich soll die einfachste Lösung gefunden werden, die die oben beschriebene Funktion der existentiellen Typen komfortabel zur Verfügung stellt.

3.2.3.3 Teilsprachen Insbesondere für die Einbettung anderer Sprachen, einschließlich weiterer funktionaler Programmiersprachen, in HasCASL sind Teilsprachen von großer Bedeutung (Zu Teilsprachen von CASL erster Stufe s. [Mos00b].). Der Sprachentwurf wird von vornherein die Auszeichnung geeigneter Teilsprachen mit beeinhalten. In der Tat liegt eine Teilsprache in Gestalt des Entwurfs [MHKB00] im wesentlichen bereits vor; zum anvisierten Zeitpunkt werden ferner die in Phase I entworfene Sprache und ihr ausführbarer Anteil fertiggestellt sein. Weitere Teilsprachen sollen durch Einschränkungen des Typsystems ausgezeichnet werden. Ein besonderes Augenmerk ist hier auf vorhandene ähnliche Sprachen bzw. Typsysteme wie etwa die in Abschnitt 2.1.3 aufgeführten Spezifikationssprachen mit Bestandteilen höherer Stufe zu richten, die für Übersetzungen von und nach HasCASL in Frage kommen.

Bei der Definition von Teilsprachen soll allgemein auf die einfache, d.h. syntaktische, Erkennbarkeit sowie die semantisch treue Einbettbarkeit wichtiger Teilsprachen (die bedingt, daß in einer Teilsprache geschriebene Spezifikationen in Erweiterungen die gleiche Semantik behalten) geachtet werden.

Hierbei führt die letztere Forderung auf wesentliche semantische Probleme, bei deren Bewältigung kategorielle bzw. universell-algebraische Techniken erneut im Mittelpunkt stehen werden.

3.2.3.4 HasCASL B Am Ende von Phase II steht HasCASL B, eine Spezifikationssprache, die alle wesentlichen Konzepte von Haskell umfaßt, zur Verfügung.

3.2.4 Werkzeug-Teil, Phase I

3.2.4.1 Syntaktische und statische semantische Analyse Das bestehende Bremer CASL-Werkzeug erlaubt syntaktische sowie statische semantische Analyse von CASL-Spezifikationen erster Stufe [MKK98]. Dieses Werkzeug soll zunächst an die Spracherweiterung gemäß [MHKB00] und anschließend an HasCASL angepaßt werden. Dies erfordert neue Lösungen vor allem im Bereich des Mixfix-Parsing und der Typüberprüfung für Funktionen höherer Stufe und Typkonstruktoren in Zusammenspiel mit Subtypen. Außerdem soll eine Konsistenzprüfung rekursiver Typen im Rahmen der statischen Analyse implementiert werden.

3.2.4.2 Codierung in Isabelle/HOL; Verfeinerungsbeweise Die Beweisunterstützung für CASL-Spezifikationen erster Stufe ist durch Codierung syntaktisch und bezüglich der statischen Semantik korrekter Spezifikationen in Isabelle/HOL realisiert [MKK98]. Für HasCASL soll eine entsprechende Codierung in Isabelle/HOL zunächst theoretisch korrekt entwickelt und dann implementiert werden. Zudem sollen geeignete Sätze von Termersetzungsregeln sowie Reduktionstaktiken aufgestellt und implementiert werden, die erlauben, mit den in Isabelle/HOL nicht vorhandenen HasCASL-Konzepten wie Subtypen und Partialität effizient umzugehen.

Mit der Codierung von HasCASL in Isabelle/HOL kann dann sowohl geprüft werden, ob eine Spezifikation ihre impliziten Beweisverpflichtungen erfüllt, als auch, ob intendierte Folgerungen gelten, und es können Verfeinerungen zwischen Spezifikationen bewiesen werden.

3.2.4.3 Werkzeuge für die ausführbare Teilsprache Entscheidend für den Entwicklungsprozeß mit HasCASL ist der Übergang von einer allgemeinen HasCASL-Spezifikation zu einer Spezifikation in der ausführbaren Teilsprache. Deshalb soll ein Werkzeug zur Überprüfung primitiv-rekursiver, wohlfundiert-rekursiver und allgemein partiell rekursiver Funktionen (siehe Abschnitt 3.2.2.1) implementiert werden. Im Fall der primitiven Rekursion reicht hier eine schlichte Syntaxprüfung aus [BN98]; im wohlfundierten Fall ist, neben der Erzeugung von Beweisverpflichtungen über eine vom Benutzer vorgegebene wohlfundierte Ordnung, die halbautomatische Generierung

solcher Ordnungen mit Hilfe von Standardverfahren [Gie97] oder möglicherweise nach der in der innerhalb der Forschungsgruppe Krieg-Brückner verfaßten Dissertation [Sei93] entwickelten Methode vorgesehen. Die Beweisunterstützung für die Spezifikation von allgemein partiell rekursiven Funktionen mittels des free-Konstrukts in HasCASL verlangt in der ersten Stufe im wesentlichen die Möglichkeit von Induktionsbeweisen [AHMS99], während das entsprechende Problem in Logik höherer Stufe noch zu lösen ist.

3.2.4.4 Übersetzung HasCASL–Haskell Als letzter Baustein zur Unterstützung des Entwicklungsprozesses soll dann ein Übersetzer von ausführbarem HasCASL nach Haskell und zurück implementiert werden. Da die ausführbare HasCASL-Teilsprache eng an Haskell orientiert ist, sollte dies relativ einfach sein.

3.2.4.5 Werkzeug-Satz für HasCASL Am Ende von Phase I steht somit ein Werkzeug-Satz zur Verfügung, mit dem HasCASL-Spezifikationen entwickelt, überprüft, in Richtung ausführbare Teilsprache manuell verfeinert und schließlich nach Haskell übersetzt werden können.

3.2.5 Werkzeug-Teil, Phase II

3.2.5.1 Fallstudie In Phase II soll u.a. der in Phase I entwickelte Werkzeug-Satz mittels einer Fallstudie auf Praxis-Tauglichkeit getestet werden. Die Fallstudie soll einen Compiler für eine kleine Programmiersprache mit Parsing, statischer Analyse und Code-Erzeugung umfassen, oder aber, falls dies vom Umfang her durchführbar bzw. geeignet eingrenzbar erscheint, die HasCASL-Werkzeuge selbst (mit Parsing, statischer Analyse und Codierung). Die Fallstudie soll exemplarisch eine vollständige Entwicklung durchführen, von der Anforderungsspezifikation bis zum fertigen Programm; sie ist von zentraler Bedeutung für den Nachweis der Praxistauglichkeit des ganzes Ansatzes. Die dabei zu lösenden Probleme (insbesondere die Durchführung der notwendigen Beweise) sind von ihrem Umfang her nicht zu unterschätzen. Von der Fallstudie erhoffen wir uns zudem wichtige Impulse für die Verbesserung und Weiterentwicklung des Werkzeug-Satzes in Richtung einer praktisch anwendbaren Entwicklungsumgebung; insofern wird die Fallstudie in Phase II hinsichtlich Abfolge und Zeiteinteilung Priorität haben.

3.2.5.2 Transformationen für HasCASL Das der Fallstudie zugrundeliegende Paradigma ist die schrittweise Verfeinerung von Anforderungsspezifikationen zu (ausführbaren) Entwurfsspezifikationen (nicht etwa die nachträgliche Verifikation eines existierenden Programms). Diese Verfeinerung soll durch Transformationen unterstützt werden, indem sowohl bereits

existierende Transformationen für die Entwicklung eingesetzt als auch erfolgreiche Entwicklungsschritte als Transformationen für die spätere Wiederverwendung aufgehoben werden.

Zur Entwicklung der Transformationen soll das generische Bremer "Transformation Application System" (siehe Abschnitt 2.2.2) benutzt werden, das die oben skizzierten Möglichkeiten unterstützt; Voraussetzung hierfür ist die geplante HasCASL-Codierung in Isabelle/HOL (s. Abschnitt 3.2.4.2).

3.2.5.3 Werkzeug-Satz für HasCASL B Sobald der Sprachentwurf von HasCASL B fertiggestellt ist, sollen die in Phase I entwickelten Werkzeuge auf HasCASL B (d.h. um die Behandlung nicht-strikter Funktionen und existentieller Typen) erweitert werden. Am Ende von Phase II steht somit eine Entwicklungsumgebung zur Spezifikation und formalen Entwicklung von Software zur Verfügung, die in einem ersten Test ihre Praxistauglichkeit unter Beweis gestellt hat.

4 Beantragte Mittel

4.1 Personalbedarf

4.2 Wissenschaftliche Geräte

4.3 Verbrauchsmaterial und Versuchstiere

4.4 Reisen

4.5 sonstige Kosten

5 Voraussetzungen für die Durchführung des Vorhabens

5.1 Zusammensetzung der Arbeitsgruppe

In der Forschungsgruppe Krieg-Brückner im Studiengang Informatik des Fachbereichs Mathematik/Informatik an der Universität Bremen arbeiten Dr. Till Mossakowski (DFG-Projekt MULTIPLE), Dr. Markus Roggenbach (DFG-Projekt COOFL), Dr. Christoph Lüth (Isabelle/HOL, generische grafische Benutzerschnittstelle) und Dr. George Russell (UniForM Workbench) an Fragestellungen, die eng mit dem beantragten Projekt zusammenhängen. B. Krieg-Brückner garantiert für die Weiterbeschäftigung von Herrn T. Mossakowski auch über das Ende des DFG-Projekts MULTIPLE hinaus, sofern

es für das beantragte Projekt erforderlich ist.

5.2 Zusammenarbeit mit anderen Wissenschaftlern

1. Prof. Dr. Donald Sannella, Universität Edinburgh (Funktionale Programmierung, Sprachentwurf, Semantik)
2. Prof. Dr. Anne Haxthausen, Technische Universität Dänemark, Lyngby (CASL höherer Stufe)
3. Prof. Dr. Andrzej Tarlecki, Polnische Akademie der Wissenschaften, Warschau (Semantik)
4. Dr. Hélène Kirchner, LORIA-CNRS & INRIA Lorraine, Nancy (Werkzeugentwicklung)
5. Dr. Mark van den Brandt, CWI, Amsterdam (Werkzeugentwicklung)
6. Dr. Dieter Hutter, Heiko Mantel, Axel Schairer, Serge Autexier, DFKI GmbH, Saarbrücken (Werkzeugentwicklung)
7. PD Dr. Michael Kohlhase, Universität des Saarlandes, Saarbrücken (Werkzeugentwicklung, CASL höherer Stufe)

5.3 Auslandsbezug

Mit den unter 5.2 genannten Wissenschaftlern aus dem Ausland bestehen Kooperationen (u.a. Veröffentlichung gemeinsamer Arbeiten), die im Rahmen des beantragten Projekts fortgesetzt werden sollen. Zudem ergeben sich internationale Kooperationen, die mit dem beantragten Vorhaben in Zusammenhang stehen, aus der Mitarbeit in der CoFI Working Group.

5.4 Apparative Ausstattung

5.5 Laufende Mittel für Sachausgaben

5.6 Sonstige Voraussetzungen

6 Erklärungen

6.1

Ein Antrag auf Finanzierung dieses Vorhabens wurde bei keiner anderen Stelle eingereicht. Wenn wir einen solchen Antrag stellen, werden wir die Deutsche Forschungsgemeinschaft unverzüglich benachrichtigen.

6.2

Der Vertrauensmann der DFG an der Universität Bremen, Herr Prof. Dr. Walter R. Heinz, ist über die Antragstellung informiert.

7 Unterschrift

Bremen, den 12.05.2000

B. Krieg-Brückner T. Mossakowski H. Herrlich

8 Verzeichnis der Anlagen

Literatur

[AH86] J. Adámek, H. Herrlich. Cartesian closed categories, quasitopoi, and topological universes. *Comm. Math. Univ. Carol.* **27**, 235–257, 1986.

[AH90] J. Adámek, H. Herrlich. A characterization of concrete quasitopi by injectivity. *J. Pure Appl. Algebra* **68**, 1–9, 1990.

[AHR88] J. Adámek, H. Herrlich, J. Rosický. Essentially equational categories. *Cahiers Topol. Géom. Diff. Catég.* **29**, 175–192, 1988.

[AHS90] J. Adámek, H. Herrlich, G. E. Strecker. *Abstract and Concrete Categories.* Wiley Interscience, New York, 2nd edition, 1990.

[AHT89] J. Adámek, H. Herrlich, W. Tholen. Monadic decompositions. *J. Pure Appl. Algebra* **59**, 111–123, 1989.

[And86] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Academic press, 1986.

[AL91] A. Asperti, G. Longo. *Categories, Types, and Structures.* Foundations of Computing Series. The MIT Press, Cambridge, Massachusetts, 1991.

[ABKB$^+$] E. Astesiano, M. Bidoit, B. Krieg-Brückner, P. D. Mosses, D. Sannella, A. Tarlecki. CASL - the common algebraic specification language. Submitted to *Theoretical Computer Science.*

[AKKB99] E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner. *Algebraic Foundations of Systems Specification.* Springer, 1999. 616 pages.

[AHMS99] S. Autexier, D. Hutter, H. Mantel, A. Schairer. System description: inka 5.0 — A logic voyager. *Lecture Notes in Computer Science* **1632**, 207–211, 1999.

[BN98] F. Baader, T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, 1998.

[Bar92] H. P. Barendregt. Lambda calculi with types. In D. M. Gabbay, S. Abramski, T. S. E. Maibaum, eds., *Handbook of Logic in Computer Science.* Oxford University Press, Oxford, 1992.

[B$^+$97] B. Barras, et al. The Coq Proof Assistant Reference Manual: Version 6.1. Technical Report RT-0203, Inria, Institut National de Recherche en Informatique et en Automatique, 1997.

[BKB99] D. Basin, B. Krieg-Brückner. Formalization of the development process. In E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner, eds., *Algebraic Foundations of Systems Specification*, 521–562. Springer, 1999.

[BH98] L. Bentley, H. Herrlich. Morita-extensions and nearness-completions. *Topol. Appl.* **82**, 59–65, 1998.

[BKL$^+$91] M. Bidoit, H.-J. Kreowski, P. Lescannes, F. Orejas, D. Sannella. *Algebraic System Specification and Development. A Survey and Annotated Bibliography, Lecture Notes in Computer Science* **501**. Springer Verlag, 1991.

[Bir98] R. Bird. *Introduction to Functional Programming using Haskell, second edition.* Printice Hall Europe, London, 1998.

[BFG+93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, K. Stølen. The requirement and design specification language Spectrum, an informal introduction, version 1.0. Technical report, Institut für Informatik, Technische Universität München, March 1993.

[CW85] L. Cardelli, P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys* **17**(4), 471–522, December 1985.

[CGK+97] M. Cerioli, M. Gogolla, H. Kirchner, B. Krieg-Brückner, Z. Qian, M. Wolf. *Algebraic System Specification and Development. A Survey and Annotated Bibliography, 2nd Edition*, *BISS monographs* **3**. Shaker Verlag, 1997.

[CHKBM97] M. Cerioli, A. Haxthausen, B. Krieg-Brückner, T. Mossakowski. Permissive subsorted partial logic in CASL. In Michael Johnson, ed., *Algebraic Methodology and Software Technology: 6th International Conference, AMAST 97*, *Lecture Notes in Computer Science* **1349**. Springer-Verlag, 1997.

[CMR99] M. Cerioli, T. Mossakowski, H. Reichel. From total equational to partial first order logic. In E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner, eds., *Algebraic Foundations of Systems Specifications*, 31–104. Springer Verlag, 1999.

[C+99] M. Clavel, et al. The Maude system. In P. Narendran, M. Rusinowitch, eds., *Proceedings of the 10th International Conference on Rewriting Techniques and Applications (RTA-99)*, 240–243, Trento, Italy, July 1999. Springer-Verlag LNCS 1631. System Description.

[CoF] CoFI. The Common Framework Initiative for algebraic specification and development, electronic archives. Notes and Documents accessible by WWW[1] and FTP[2].

[CoF98] CoFI Language Design Task Group. CASL – The CoFI Algebraic Specification Language – Summary. Documents/CASL/Summary, in [CoF], October 1998.

[CoF99] CoFI Semantics Task Group. CASL – The CoFI Algebraic Specification Language – Semantics. Note S-9 (version 0.95), in [CoF], March 1999.

[Coq86] T. Coquand. An analysis of girard's paradox. In *Proceedings Symposium on Logic in Computer Science*, 227–236, 1986.

[Cro94] R. L. Crole. *Categories for Types.* Cambridge Mathematical Textbooks. Cambridge, Cambridge, UK, 1994.

[1] http://www.brics.dk/Projects/CoFI
[2] ftp://ftp.brics.dk/Projects/CoFI

[DF98] R. Diaconescu, K. Futatsugi. *CafeOBJ Report.* AMAST series. World Scientific, Singapore, 1998.

[EHKP89] H. Ehrig, H. Herrlich, H.-J. Kreowski, G. Preuss, eds. *Categorical Methods in Computer Science*, Berlin, 1989.

[FLMJ99] S. Finne, D Leijen, E. Meijer, S. Peyton Jones. *H/Direct*: A binary foreign language interface for haskell. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98), ACM SIGPLAN Notices* **34(1)**, 153–162. ACM, June 1999.

[GHH+92] C. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. Bendix Nielson, S. Prehn, K. R. Wagner. *The Raise Specification Language.* Prentice Hall, New York, 1992.

[Gie97] J. Giesl. Termination of nested and mutually recursive algorithms. *Journal of Automated Reasoning* **19**(1), 1–29, August 1997.

[GC95] M. Gogolla, M. Cerioli. What is an abstract data type after all? *Lecture Notes in Computer Science* **906**, 499–523, 1995.

[GM93] M. J. C. Gordon, T. M. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logics.* Cambridge University Press, 1993.

[GHG+93] J. V. Guttag, J. J. Horning, S. J. Garland, K. D. Jones, A. Modet, J. M. Wing. *Larch: Languages and Tools for Formal Specification.* Texts and Monographs in Computer Science. Springer-Verlag, 1993.

[HHPW96] C. V. Hall, K. Hammond, S. L. Peyton Jones, P. L. Wadler. Type classes in Haskell. *ACM Transactions on Programming Languages and Systems* **18**(2), 109–138, March 1996.

[Hen49] L. Henkin. The completeness of the first-order functional calculus. *Journal of Symbolic Logic* **14**(3), 159–166, 1949.

[Her68] H. Herrlich. *Topologische Reflexionen und Coreflexionen*, *Lect. Notes Math.* **78**. Springer Verlag, Berlin, 1968.

[Her71] H. Herrlich. A characterization of k-ary algebraic categories. *Manuscripta Math.* **4**, 277–284, 1971.

[Her74] H. Herrlich. Regular categories and regular functors. *Canad. J. Math.* **26**, 709–720, 1974.

[Her86] H. Herrlich. Essentially algebraic categories. *Quaestiones Math.* **9**, 245–262, 1986.

[Her90] H. Herrlich. Remarks on categories of algebras defined by a proper class of operations. *Quaestiones Math.* **13**, 385–393, 1990.

[Her93] H. Herrlich. On the failure of Birkhoff's theorem for locally small based equational categories of algebras. *Cahiers Topol. Géom. Diff. Catég.* **34**, 185–192, 1993.

[HS99] H. Herrlich, L. Schröder. Composing special epimorphisms and retractions. *Cahiers Topol. Gom. Diff* **40**, 221–226., 1999.

[HS79] H. Herrlich, G. E. Strecker. *Category Theory.* Heldermann Verlag, Berlin, 2nd edition, 1979.

[HM95] M. Hoang, J. C. Mitchell. Lower bounds on type inference with subtypes. In *Conference Record of POPL '95: 22nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 176–185, New York, January 1995. ACM.

[HKB93] B. Hoffmann, B. Krieg-Brückner. *Program development by specification and transformation: the PROSPECTRA methodology, language family, and system*, *Lecture Notes in Computer Science* **680**. Springer-Verlag, New York, 1993.

[HMR+98] D. Hutter, H. Mantel, G. Rock, W. Stephan, A. Wolpers, M. Balser, W. Reif, G. Schellhorn, K. Stenzel. VSE: Controlling the complexity in formal software developments. In *Proceedings of the International Workshop on Applied Formal Methods*, Boppard, Germany, 1998.

[I-L] I-Logix, Inc. Rhapsody overview. http://www.ilogix.com/rhover_c.htm.

[IBH+79] J. D. Ichbiah, J. G. P. Barnes, J. C. Heliard, B. Krieg-Brückner, O. Roubine, B. A. Wichmann. Reference manual and rationale for the Ada programming language. *ACM SIGPLAN Notices* **14**(6), June 1979.

[Jac99] B. Jacobs. *Categorical Logic and Type Theory.* Studies in Logic and the Foundations of Mathematics. Elsevier, Amsterdam, 1999.

[Jon00] M. P. Jones. Type classes with functional dependencies. In G. Smolka, ed., *Programming Languages and Systems, ESOP2000*, number 1782 in Lecture Notes in Computer Science, Berlin, 2000. Springer.

[JHA+99] S. P. Jones, J. Hughes, L. Augustsson, D. Barton, B. Boutel, W. Burton, J. Fasel, K. Hammond, R. Hinze, P. Hudak, T. Johnsson, M. Jones, J. Launchbury, E. Meijer, J. Peterson, A. Reid, C. Runciman, P. Wadler. Haskell 98: A non-strict, purely functional language, 1999. http://www.haskell.org/onlinereport.

[KST97] S. Kahrs, D. Sannella, A. Tarlecki. The definition of extended ML: A gentle introduction. *Theoretical Computer Science* **173**(2), 445–484, 1997.

[Kar98] E. W. Karlsen. *Tool integration in a functional programming language.* PhD thesis, University of Bremen, 1998. Revised Version 1999.

[Kar99] E.W. Karlsen. The UniForM Workbench, a higher-order tool integration framework. In D. Hutter, W. Stephan, P. Traverso, M. Ullmann, eds., *Applied Formal Methods - FM-Trends 98. International Workshop on Current Trends in Applied Formal Methods*, *Lecture Notes in Computer Science* **1641**, 266–280. Springer, 1999.

[KSW96] Kolyang, T. Santen, B. Wolff. Correct and user-friendly implementations of transformation systems. In M. C. Gaudel, J. Woodcock, eds., *Formal Methods Europe '96*, number 1051 in Lecture Notes in Computer Science, 629–648. Springer, 1996.

[KB89] B. Krieg-Brückner. COMPASS, a COMPrehensive Algebraic approach to System Specification and development (ESPRIT basic research working group 3264): Objectives, state of the art, references. Informatik Bericht 6/89, Universität Bremen, 1989.

[KB90] B. Krieg-Brückner. COMPASS, a COMPrehensive Algebraic approach for System Specification and development, ESPRIT basic research working group 3264. *EATCS Bulletin* **40**, 144–157, 1990.

[KB96] B. Krieg-Brückner. Seven years of COMPASS. In M. Haveraaen, O. Owe, O. J. Dahl, eds., *Recent Trends in Data Type Specification*, *Lecture Notes in Computer Science* **1130**, 1–13. Springer-Verlag, 1996.

[KB99] B. Krieg-Brückner. Uniform perspectives for formal methods. In D. Hutter, W. Stephan, P. Traverso, M. Ullmann, eds., *Applied Formal Methods - FM-Trends 98. International Workshop on Current Trends in Applied Formal Methods*, *Lecture Notes in Computer Science* **1641**, 251–265. Springer, 1999.

[KBe91] B. Krieg-Brückner, D. Plump (eds.). COMPASS, a COMPrehensive Algebraic approach to System Specification and development (ESPRIT basic research working group 3264): Final report. Informatik Bericht 7/91, Universität Bremen, 1991.

[KPO+99] B. Krieg-Brückner, J. Peleska, E.-R. Olderog, D. Balzer, A. Baer. The UniForM Workbench, a universal development environment for formal methods. In *FM99: World Congress on Formal Methods*, *Lecture Notes in Computer Science* **1709**, 1186–1205. Springer-Verlag, 1999.

[KBQ96] B. Krieg-Brückner, Z. Qian. Object-oriented functional programming and type reconstruction. In M. Haveraaen, O. Owe, O. J. Dahl, eds., *Recent Trends in Data Type Specification*, *Lecture Notes in Computer Science* **1130**, 458–477. Springer-Verlag, 1996.

[KBS91] B. Krieg-Brückner, D. Sannella. Structuring specifications in-the-large and in-the-small: Higher-order functions, dependent types and inheritance in Spectral. In *Proc. TAPSOFT 91*, *Lecture Notes in Computer Science* **494**, 313–336. Springer Verlag, 1991.

[LS86] J. Lambek, P. J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge University Press, 1986.

[Läu96] K. Läufer. Type classes with existential types. *Journal of Functional Programming* **6**(3), 485–517, May 1996.

[Liu98] J. Liu. *Subclass inheritance in Spectral and higher order pattern narrowing.* PhD thesis, University of Bremen, 1998.

[LW] C. Lüth, B. Wolff. TAS — a generic transformation system. In *Theorem Proving in Higher Order Logics TPHOLS 2000*, Lecture Notes in Computer Science. To appear.

[MW] T. Meyer, B. Wolff. Correct code-generation in a generic framework. University of Bremen; submitted for publication.

[MP88] J. C. Mitchell, G. D. Plotkin. Abstract types have existential type. *ACM Trans. Prog. Lang. and Sys.* **10**(3), 470–502, July 1988. Proc. 12th ACM Symposium on Principles of Programming Languages, 85.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation* **93**, 55–92, 1991.

[Mos97a] T. Mossakowski. Sublanguages of CASL. Note L-7, in [CoF], December 1997.

[Mos97b] T. Mossakowski. Subsorting and structured specification. Note S-3, in [CoF], March 1997.

[Mos98a] T. Mossakowski. Cocompleteness of the CASL signature category. Note S-7, in [CoF], February 1998.

[Mos98b] T. Mossakowski. Colimits of order-sorted specifications. In F. Parisi Presicce, ed., *Recent Trends in Algebraic Development Techniques. Proc. 12th International Workshop*, *Lecture Notes in Computer Science* **1376**, 316–332. Springer, 1998.

[Mos98c] T. Mossakowski. Standard annotations for parsers and static semantic checkers – a proposal. Note T-6 (revised), in [CoF], September 1998.

[Mos98d] T. Mossakowski. Two "functional programming" sublanguages of CASL. Note L-9, in [CoF], March 1998.

[Mos98e] T. Mossakowski. Version control and registration for CASL libraries. Note M-5, in [CoF], September 1998.

[Mos00a] T. Mossakowski. CASL: From semantics to tools. In S. Graf, M. Schwartzbach, eds., *TACAS 2000*, *Lecture Notes in Computer Science* **1785**, 93–108. Springer-Verlag, 2000.

[Mos00b] T. Mossakowski. Relating CASL with other specification languages: the institution level. *Theoretical Computer Science*, 2000. To appear.

[Mos00c] T. Mossakowski. Specification in an arbitrary institution with symbols. In C. Choppy, D. Bert, eds., *Recent Developments in Algebraic Development Techniques, 14th International Workshop, WADT'99, Bonas, France*, *Lecture Notes in Computer Science* **1827**. Springer-Verlag, 2000. To appear.

[MHKB00] T. Mossakowski, A. Haxthausen, B. Krieg-Brückner. Subsorted partial higher-order logic as an extension of CASL. In C. Choppy, D. Bert, eds., *Recent Developments in Algebraic Development Techniques, 14th International Workshop, WADT'99, Bonas, France*, *Lecture Notes in Computer Science* **1827**. Springer-Verlag, 2000. To appear.

[MKK98] T. Mossakowski, Kolyang, B. Krieg-Brückner. Static semantic analysis and theorem proving for CASL. In F. Parisi Presicce, ed., *Recent Trends in Algebraic Development Techniques. Proc. 12th International Workshop*, *Lecture Notes in Computer Science* **1376**, 333–348. Springer, 1998.

[Mos97] P. D. Mosses. CoFI: The Common Framework Initiative for Algebraic Specification and Development. In *TAPSOFT '97: Theory and Practice of Software Development*, *LNCS* **1214**, 115–137. Springer-Verlag, 1997. Documents/Tentative/Mosses97TAPSOFT, in [CoF].

[MNvOS99] O. Müller, T. Nipkow, D. von Oheimb, O. Slotosch. HOLCF = HOL + LCF. *Journal of Functional Programming* **9**, 191–223, 1999.

[Nor99] J. Nordlander. Pragmatic subtyping in polymorphic languages. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, *ACM SIGPLAN Notices* **34(1)**, 216–227. ACM, June 1999.

[Nor00] J. Nordlander. Polymorphic subtyping in O'Haskell. In *Proceedings of the APPSEM Workshop on Subtyping and Dependent Types in Programming*, 2000. To appear.

[Obj99] Object Management Group, Inc. *OMG Unified Modeling Language Specification, Version 1.3.* June 1999. http://www.omg.org.

[OSRSC99] S. Owre, N. Shankar, J. M. Rushby, D. W. J. Stringer-Calvert. *PVS Language Reference.* SRI International, Menlo Park, 1999. http://pvs.csl.sri.com.

[Pau87] L. C. Paulson. *Logic and Computation : Interactive Proof with Cambridge LCF*, *Cambridge Tracts in Theoretical Computer Science* **2**. Cambridge University Press, 1987.

[Pau94] L. C. Paulson. *Isabelle - A Generic Theorem Prover.* Number 828 in Lecture Notes in Computer Science. Springer Verlag, 1994.

[Pau96] L. C. Paulson. *ML for the Working Programmer.* Cambridge University Press, New York, NY, second edition, 1996.

[PGF96] S. Peyton Jones, A. Gordon, S. Finne. Concurrent Haskell. In ACM, ed., *Conference record of POPL '96, 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: papers presented at the Symposium: St. Petersburg Beach, Florida, 21–24 January 1996*, 295–308, New York, NY, USA, 1996. ACM Press.

[QW92] Z. Qian , K. Wang. Higher-order E-unification for arbitrary theories. In Krzysztof Apt, ed., *Proceedings of the Joint International Conference and Symposium on Logic Programming*, 52–66, Washington, USA, 1992. The MIT Press.

[Qia94] Z. Qian. Higher-order equational logic programming. In ACM, ed., *Proceedings of 21st Annual ACM SIGACT-SIGPLAN Symposium on*

Principles of Programming Languages, 254–267, New York, NY, USA, 1994. ACM Press.

[Qia95] Z. Qian. Unification of higher-order patterns in linear time and space. *Journal of Logic and Computation* **6**(3), 315–341, 1995.

[QW94] Z. Qian, K. Wang. Modular AC unification of higher-order patterns. In *Proc. Int. Conf. on Constraints in Computational Logics*, *Lecture Notes in Computer Science* **845**, 105–120. Springer, 1994.

[Rat] Rational Software Corporation. Rational rose. http://www.rational.com/products/rose/.

[Reg95] F. Regensburger. HOLCF: Higher order logic of computable functions. *Lecture Notes in Computer Science* **971**, 293–307, 1995.

[Reh97] J. Rehof. Minimal typings in atomic subtyping. In *Proceedings POPL '97, 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 278–291. ACM Press, January 1997.

[R+98] W. Reif, et al. Structured specifications and interactive proofs with KIV. In Wolfgang Bibel, Peter H. Schmidt, eds., *Automated Deduction: A Basis for Applications. Volume II, Systems and Implementation Techniques.* Kluwer Academic Publishers, Dordrecht, 1998.

[Rei99] C. Reinke. Towards a Haskell/Java connection. *Lecture Notes in Computer Science* **1595**, 200–215, 1999.

[Rey83] J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, ed., *Information Processing 83*, 513–523, Amsterdam, 1983. Elsevier Science Publishers B. V. (North-Holland).

[RM99] M. Roggenbach, T. Mossakowski. Proposal of some annotations and literal syntax in CASL. Note L-11, in [CoF], March 1999.

[RMS00] M. Roggenbach, T. Mossakowski, L. Schröder. Basic datatypes in CASL. Note L-12, version 0.4, in [CoF], March 2000.

[RSM00] M. Roggenbach, L. Schröder, T. Mossakowski. Specifying real numbers in CASL. In Christine Choppy, Didier Bert, eds., *Recent Developments in Algebraic Development Techniques, 14th International Workshop, WADT'99*, *Lecture Notes in Computer Science* **1827**. Springer, 2000. To appear.

[Sch] L. Schröder. Isomorphisms and splitting of idempotents in semicategories. *Cahiers Topol. Gom. Diff.* To appear.

[Sch99] L. Schröder. *Composition graphs and free extensions of categories.* PhD thesis, University of Bremen, 1999. In German. Also: Logos Verlag, Berlin, 1999.

[SHa] L. Schröder, H. Herrlich. Free adjunction of morphisms. *Appl. Cat. Struct.* To appear.

[SHb] L. Schröder, H. Herrlich. Free factorizations. *Appl. Cat. Struct.* To appear.

[SH00] L. Schröder, H. Herrlich. Abstract initiality. *Comm. Math. Univ. Carol.* **41**, 2000.

[Sei93] R. Seifert. *Property analysis of term rewriting systems.* PhD thesis, Universität Bremen, 1993.

[Tho96] S. Thompson. *Haskell: The Craft of Functional Programming.* Addison-Wesley, Harlow, England, 1996.

[THM+96] P. W. Trinder, K. Hammond, J. S. Mattson Jr., A. S. Partridge, S. L. Peyton Jones. GUM: a portable parallel implementation of Haskell. In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation*, 79–88, Philadelphia, Pennsylvania, 21–24 May 1996.

[Uni] University of Glasgow. The Glasgow Haskell Compiler. http://www.haskell.org/ghc.

[Wad95] P. Wadler. Monads for functional programming. In J. Jeuring, E. Meijer, eds., *Advanced Functional Programming*, *Lecture Notes in Computer Science* **925**, 24–52. Springer, 1995.

[Wak98] D. Wakeling. A Haskell to Java Virtual Machine code compiler. *Lecture Notes in Computer Science* **1467**, 39–52, 1998.

[Wan94] K. Wang. *Higher-Order Constraint Logic Programming.* PhD thesis, Universität Bremen, 1994.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages.* MIT Press, Cambridge, Massachusetts, 1993.