Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

F.W. von Henke and H. Rueß (Eds.)

Abtl. Künstliche Intelligenz
Universität Ulm
D – 89069 Ulm, Germany
{vhenke,ruess}@informatik.uni-ulm.de

Zusammenfassung

This report contains summaries and abstracts of talks presented at the type theory workshop at the University of Ulm in April 1994.

1 Vorwort

Das Arbeitstreffen Typtheorie fand am 14.-15. April an der Universität Ulm statt, und wurde von der Abtl. Künstliche Intelligenz organisiert und von der GI-Fachgruppe 0.1.7 (Spezifikation und Semantik) unterstützt. Der Zweck dieses Treffens war in erster Linie, einen Überblick über laufende Arbeiten und Interessen im Umfeld der Typtheorie in Deutschland (und näherer Umgebung) zu gewinnen und das Interesse an möglichen weiteren gemeinsamen Aktivitäten zu erkunden.

Es gab insgesamt 23 Teilnehmer an diesem Arbeitskreis und 17 Präsentationen aus den Teilbereichen

- Beweistheoretische Untersuchungen
- Implementierungsaspekte
- Maschinelle Deduktion
- Metasprachen
- Querbezüge zu Spezifikation und Programmierung
- Semantik

Der vorliegende Bericht enthält Zusammenfassungen dieser Präsentationen und ist auch über *World Wide Web*

```
http://www.informatik.uni-ulm.de/fakultaet/abteilungen/ki
/Forschung/Typentheorie/Ulm/workshop94.html
```

und *ftp*

```
ftp.informatik.uni-ulm.de:/pub/papers/uib/UIB-94-08.dvi.Z
```

verfügbar.

Die Organisatoren bedanken sich bei den Teilnehmern recht herzlich für die rege Beteiligung und das fruchtbare Arbeitsklima.

F.W. von Henke
H. Rueß

2 Zusammenfassung

Unification in a Sorted λ-Calculus with Term Declarations and Function Sorts

Michael Kohlhase
Universität Saarbrücken

The introduction of sorts to first-order automated deduction has brought greater conciseness of representation and a considerable gain in efficiency by reducing search spaces. This suggests that sort information can be employed in higher-order theorem proving with similar results.

This paper develops a sorted λ-calculus suitable for automatic theorem proving applications by extending the simply typed λ-calculus with a higher-order sort concept that includes term declarations and functional base sorts. The term declaration mechanism studied here is powerful enough to subsume subsorting as a derived notion and therefore gives a justification for the special form of subsort inference.

We present a set of transformations for sorted (pre-) unification and prove the nondeterministic completeness of the algorithm induced by these transformations.

A long version of this abstract will appear in the proceedings of the *Deutsche Jahrestagung für Künstliche Intelligenz KI'94*, Saarbrücken, Springer Verlag, 1994.

Martin–Löf Typentheorien mit Wohlordnungstypen und Universen

Michael Rathjen
Universität Münster

Gegenstand des Vortrags ist die beweistheoretische Stärke von Martin–Löf Typentheorien mit Wohlordnungstypen und Universen.

In einer gemeinsamen Arbeit mit E. Griffor wurde bewiesen, daß die Theorie mit einem Universum die Konsistenz eines recht starken Teilsystems der Zahlentheorie zweiter Stufe beweist, und zwar des Systems mit $Delta^1_2$–Komprehension und Bar–Induktion. Da Martin–Löf mit seiner Typentheorie eine Grundlegung der konstruktiven Mathematik beabsichtigte, stellt sich heraus, daß für ein starkes klassisches System die Konsistenz konstruktiv bewiesen werden kann.

Subtyping and Update

Martin Hofmann
University of Edinburgh

We propose an extension of the polymorphic lambda calculus (with records) with subtyping under which whenever S is a subtype of T in addition to the implicit coercion form S to T there is a function $put[S,T] : S \to T \to S$ allowing to overwrite the "T-part " in an element of S by an element of T. It turns out that in this calculus the contravariant subtyping rules for arrow types and polymorphic types become unsound, but all the other usual subtyping rules including record extension carry over. We give a realizability model for this calculus and an equational theory which is sound for this interpretation.

The "put"-functions provided by this calculus can be used to update records and also to model inheritance in object-oriented programming. Using the equational theory one can show that proofs of certain specifications can be "inherited" as well. We give an example using points and coloured points to demonstrate this.

Typ-Klassen und implizite Umwandlungen im Kontext der Computeralgebra

Andreas Weber[1]
Universität Tübingen

Typ-Klassen wurden durch Haskell für funktionale Sprachen eingeführt und wurden systematisch untersucht (etwa seit 1989). Für das Computeralgebra-System SCRATCHPAD II (AXIOM) ist unter dem Namen „Categories“ schon seit etwa 1981 ein den Typ-Klassen entsprechendes Konzept eingeführt worden. Jedoch gab es nur sehr wenige systematische Untersuchungen über die AXIOM-categories. In [2] wurde mit einer systematischen Untersuchung begonnen und es wurde gezeigt, daß der Typinferenz-Algorithmus von Haskell auch für viele Fälle der AXIOM „Categories“ anwendbar ist.

Ein weiteres für die Computeralgebra wichtiges Konzept, das zu den Typ-Klassen orthogonal ist, ist das der impliziten Umwandlungen. Dieses Konzept fehlt weitgehend in funktionalen Sprachen, da vor allem die Kombination mit Funktionen höherer Ordnung zu großen technischen Schwierigkeiten führt. In [2, 1] wurden viele für die Computeralgebra wichtige Fälle von impliziten Umwandlungen untersucht und semantische Kohärenzeigenschaften bewiesen. Ein Typinferenz-Algorithmus für dieses System von impliziten Umwandlungen wurde in [3, 2] angegeben.

Literatur

[1] Weber, A. On coherence in computer algebra. In *Design and Implementation of Symbolic Computation Systems — International Symposium DISCO '93* (Gmunden, Austria, Sept. 1993), A. Miola, Ed., vol. 722 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 95–106.

[2] Weber, A. *Type Systems for Computer Algebra.* Dissertation, Fakultät für Informatik, Universität Tübingen, July 1993.

[3] Weber, A. Algorithms for type inference with coercions. In *Proc. Symposium on Symbolic and Algebraic Computation (ISSAC '94)* (Oxford, July 1994), Association for Computing Machinery. To appear.

[1]Ein Teil der Arbeit wurde im Rahmen des DFG-Projekts Lo 231/5-1 durchgeführt.

Typen und Strukturen in der Computer Algebra

Stephan Missura
ETH Zürich

Computeralgebra beschäftigt sich mit Datenstrukturen wie beliebig langen Zahlen, Galois-Körpern, Polynomen, rationalen Funktionen, ...und Algorithmen wie schnelle Integeroperationen, ggT-Berechnungen und Faktorisieren von Polynomen, symbolisches Integrieren usw.

Daher ist die Computeralgebra

- nahe bei der konstruktiven Mathematik und Logik,
- auf "natürliche" Art getypt,
- geeignetes Umfeld für (getypte) funktionale Sprachen.

Jedoch sind weitverbreitete Systeme wie Mathematica oder Maple ungetypt und teilweise nicht higher-order! Das System Axiom, als einziges unter den kommerziellen Systemen statisch getypt, bietet zudem keine polymorphen und algebraischen Typen an, besitzt keine Logikkomponente, und identifiziert Typen mit Strukturen (analog zu Haskell und im Gegensatz zu SML). Diese Identifikation verhindert das Kreieren von mehrsortigen Strukturen und unterschiedlichen Strukturen mit gleichem Träger, so dass das "Parametrisierte Programmieren" wie in SML oder OBJ nur schwer möglich ist.

Grundsätzlich kann gesagt werden, dass die in der Computeralgebra verwendeten Sprachen meistens ungetypt sind, sowie keinen richtigen Logikteil besitzen und ohne formale Grundlage dastehen.

Daher kann die Computeralgebra und ihre Sprachen von Typsystemen bzw. -theorien, funktionalen Sprachen, sowie logischen Environments profitieren.

Für die Anwender von Computeralgebra ist jedoch

- "syntactic sugar" wie Coercions, Kombinieren von Theorien, Operatorenüberladung etc.,
- effiziente Implementation der in der Computeralgebra benötigten Algorithmen und Datenstrukturen und
- eine geeignete (partielle?) Logik, unterstützt durch Theorem-Prover

nötig.

Higher Order Structured Presentations In the Logical Framework

Junbo Liu
Universität Bremen

The algebraic techniques of structuring specification in the large developed in a logic-independent way based on institutions [GB84], like combining two specifications, renaming part of the signature of a specification, hiding information and abstraction by parameterization and instantiation, is reformulated by considering consequence relations instead of satisfaction relations in [FS87, HST89, Liu94] and extended to higher order case in the sense that morphisms between specifications have been considered as first- class citizens". In doing so, we can have not only higher order parameterizations but also same structuring techniques applicable to morphims.

Logical framework like LF is considered then as a way to decribing a class of logics that is representable with it [HST89], that is, a logical framework is considered as a family of consequence relations indexed by signatures. An object logic is representable in the logical framework if there is an uniform encoding of its signatures and indexed consequence relations in the logical framework. By considering a logical framework as the class of object logical systems we can instantiate logic-independent structuring presentations to this concrete class of logics, that is, we define signatures, morphisms between signatures as well as signature indexed consequence relations. Finally we obtain the useful structured presentations and structured proof search that supporting modular proof constructions for the logics representable by the logic framework.

Literatur

[HST89] R. Harper, D. Sannella, A. Tarlecki. Structure and representation in LF. In Proc. of the Symposium on Logic in Computer Science, page 226 - 237, 1989.

[FS87] J. Fiadeiro, A. Sernadas. Structuring theories on consequence. In Proc. of 5th Workshop on Specification of ADT, 1987.

[GB84] J. Goguen, R. Burstall. Introducing Institutitons. In Logics of Programs, page 221-256, Springer-Verlag, 1984

[Liu94] J. Liu. A semantic Basis for Logic-Independent Transformations. In F. Orejas (eds.) Proc. of 9th WADT-COMPASS Workshop (will appear in LNCS 1994).

Formalisierung schematischer Algorithmen

Axel Dold
Universität Ulm

Diese Arbeit beinhaltet die Formalisierung und Verifizierung eines generischen Software-Entwicklungsschrittes mittels Funktionen höherer Ordnung in einem typtheoretischen Rahmen am Beispiel des schematischen Algorithmus *Globale Suche*. Wir zeigen, daß sowohl der Entwicklungsschritt selbst als auch dessen Korrektheit in ein und demselben (typtheoretischen) Formalismus (QED) [2, 3, 5, 6] dargestellt werden kann. Die Anwendungsmöglichkeit dieses Entwicklungsschrittes wird anhand eines Beispiels (*Binäre Suche*) verdeutlicht.

Die hier vorgestellte Arbeit basiert im wesentlichen auf den Arbeiten von Doug Smith [7, 8, 9], der zahlreiche Algorithmentheorien untersucht und in seinem KIDS-System, welches eine sehr leistungsstarke Umgebung zur transformationellen Software-Entwicklung zur Verfügung stellt, implementiert hat. Teilaspekte werden dabei allerdings nur informell erfaßt. Diese Dinge werden hier genauer formalisiert. Eine ähnliche Formalisierung wurde von Christoph in seiner Habilitationsschrift [1] entwickelt.

Die Durchführung der Verifikation des Entwicklungsschrittes erfolgt mit Hilfe des Spezifikations-und Beweissystems PVS [4], welches mittels eines Gentzen-Beweisers für Logik höherer Ordnung adäquate Beweisunterstützung bietet.

Literatur

[1] C. Kreitz. *Metasynthesis - Deriving Programs that Develop Programs.* Technical report AIDA-93-03, Fachgebiet Intellektik, Technische Hochschule Darmstadt, 1993.

[2] Z. Luo. *An Extended Calculus of Construction.* Technical report CST-65-90, University of Edinburgh.

[3] Ch.E. Ore. *The Extended Calculus of Constructions (ECC) with Inductive Types.* Information and Computation, (99):231–264, 1992.

[4] S. Owre and J. M. Rushby and N. Shankar. *PVS: A Prototype Verification System.* 11th International Conference on Automated Deduction (CADE), Saratoga 1992, LNCS 607.

[5] H. Rueß. *Report on the specification language* QED. Korso Paper 1993, Universität Ulm.

[6] D. Schwier. *Type checking the specification language* QED. Korso Paper 1993, Universität Ulm.

[7] Douglas R. Smith. *Applications of a strategy for designing divide-and-conquer algorithms.* Science of Computer Programming, (8):213-229, 1987.

[8] Douglas R. Smith. *Structure and Design of Global Search Algorithms.* Technical report KES.U.87.1, Kestrel Institute, Palo Alto, CA, 1987.

[9] Douglas R. Smith and Michael R. Lowry. *Algorithm theories and design tactics.* Science of Computer Programming, (14): 305-321, 1990.

Typentheorie als Grundlage verifizierter und effizienter Systeme zur Synthese von Programmen

Christoph Kreitz
TH Darmstadt

Aus theoretischer Sicht ist Typentheorie ein geeigneter Formalismus für die Konstruktion korrekter Programme aus logischen Spezifikationen. Beweiseditoren für die Typentheorie können genutzt werden, um interaktiv oder unterstützt durch Taktiken Programme zu erstellen. Für praktische Anwendungen sind diese Werkzeuge jedoch bisher nicht verwendbar. Bei komplexeren Beispielen können voll-automatische Synthesen nicht durchgeführt werden. Andererseits aber ist das Abstraktionsniveau der elementaren Inferenzen so niedrig, daß es für Nicht-Logiker kaum zu verstehen ist, was eine benutzergesteuerte Programmentwicklung praktisch unmöglich werden läßt.

Dieses Problem kann dadurch gelöst werden, daß auf der Basis von Typentheorie eine *formale* Theorie der Programmierung entwickelt wird, welche in der Begriffsbildung den aus Standardlehrbüchern bekannten Programmierkonzepten entspricht und ein logisches Schließen auf diesem Niveau unterstützt. Da man sich auf diese Art von den Eigenarten einer speziellen Formulierung der Typentheorie lösen kann, werden Inferenzen und Synthesen einfacher, "natürlicher" und verständlicher. Durch die Abstützung auf Typentheorie bleiben sie aber dennoch vollständig formal und verifizierbar korrekt. Die Formalisierung der Theorie der Programmierung bildet daher das Fundament einer praktisch verwendbaren Inferenzmaschine für wissensbasierte Software-Entwicklung.

Neben der Formulierung von Faktenwissen über verschiedene Standardanwendungsbereiche (das sogenannte *Objektwissen*) bildet die Formalisierung deklarativen Wissens über Programmentwicklungsmethoden (*Metawissen*) einen zentralen Bestandteil dieser Theorie. Letztere ermöglicht es, eine verifizierte Implementierung von Syntheseverfahren auf der Basis von Theoremen durchzuführen, welche

die zentralen Eigenschaften in einer Art formalisieren, die der heute üblichen Beschreibungsform auf dem Papier sehr nahe kommt. Fehler beim Übergang von der Beschreibung der Strategie zur Codierung sind somit ausgeschlossen. Zudem erlaubt die deklarative Charakterisierung auf hohem Abstraktionsniveau den Vergleich und die Integration von Verfahren, die aufgrund der der verschiedenartigen Formalismen, in denen sie entworfen wurden, bisher nur schwer verglichen werden konnten.

Auch wenn die Entwicklung einer vollständigen formalen Theorie der Programmierung ein Langzeitprojekt ist, bei dem vor allem auch Wert auf die Festlegung von Standards für Begriffsbildungen gelegt werden muß, ist ihr zentraler Kern bereits erstellt und durch die Untersuchung einiger Syntheseverfahren exemplarisch getestet worden. Im Vortrag wurde dies illustriert anhand eines Theorems über die konstruktive Äquivalenz der drei Hauptparadigmen der Programmsynthese (Beweise als Programme, Transformationen, Algorithmenschemata) und der Formalisierung einer Strategie zum Entwurf von Globalsuch-Algorithmen aus ihren Spezifikationen.

Programming + Verification = Progification

Thorsten Altenkirch
Chalmers University of Technology

Eine der Ideen auf der die Typtheorie ala Martin-Löf basiert ist die Identifikation von Aussagen und Typen (oder Mengen): *Propositions as Types*. Dennoch basieren viele typtheoretische Formalisierungen auf der klassischen Trennung von Aussagen und Mengen, ein Beispiel dafür sind Luo's Kalküle ECC und UTT [Luo90, Luo92], die im LEGO system [LP92] implementiert sind.

Wir argumentieren hier, daß *Propositions as Types* ein adäquates Paradigma für die Programmverifikation ist und belegen das anhand von Beispielen im ALF-System [AGNvS94]. Die Entwicklung eines korrekten Programmes und der Nachweis seiner Korrektheit wird reduziert auf das *Programmieren* in einem System mit *dependent Types* (*Progification*). Als ein wesentliches Hilfsmittel für eine konzise Präsentation derartiger Programme erweist sich das von Coquand vorgeschlagenene und im ALF System implementierte *pattern matching* für *dependent types* [Coq92, CS93].

Obwohl die verwendete Typtheorie total ist, lassen sich allgemein rekursive Programme darstellen, wobei wir die induktive Präsentation des Wohlfundiertheitprädikates ausnutzen [Nor88]. Am Beispiel der in ALF verifizierten *merge-sort* Funktion beobachten wir, daß sich aus der typtheoretischen Präsentation mit Hilfe von statischen Optimierungen der übliche Code für *merge-sort* gewinnen läßt.

Literatur

[AGNvS94] Thorsten Altenkirch, Veronica Gaspes, Bengt Nordström, and Björn von Sydow. A user's guide to ALF. Draft, May 1994.

[Coq92] Thierry Coquand. Pattern matching with dependent types. In *Proceeding from the logical framework workshop at Båstad*, June 1992.

[CS93] Thierry Coquand and Jan M. Smith. What is the status of pattern matching in type theory? *Draft*, 1993.

[LP92] Zhaohui Luo and Robert Pollack. The LEGO proof development system: A user's manual. LFCS report ECS-LFCS-92-211, University of Edinburgh, 1992.

[Luo90] Zhaohui Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990.

[Luo92] Zhaohui Luo. A unifying theory of dependent types: the schematic approach. Technical Report ECS-LFCS-92-202, LFCS, 1992.

[Nor88] Bengt Nordström. Terminating General Recursion. *BIT*, 28(3):605–619, October 1988.

Lego – Ein homogenes Werkzeug zur Entwicklung korrekter imperativer Programme

Thomas Schreiber
University of Edinburgh

Wir zeigen, wie sich – unterstützt durch **Lego** – ausgehend von einer formalen Spezifikation ein korrektes imperatives Programm entwickeln läßt. Wir verwenden Hoare Logik als unser Verifikationskalkül und greifen auf "deliverables" zurück um das angestrebte Programm zusammen mit seinem Korrektheitsbeweis in einer homogenen Umgebung zu konstruieren.

Im Gegensatz zu früheren Studien im Edinburgh Logical Framework streben wir eine möglichst natürliche Codierung an um von der ausdrucksstarken Tyentheorie die Lego zur Verfügung stellt in unseren Beweisen zu profitieren. Anstelle der traditionell verwendeten Logik 1. Ordnung greifen wir auf die logische Ebene des Lego-Systems zurück (intuitionistische Logik höherer Ordnung). Außerdem arbeiten wir mit einem Hoare Kalkül ohne explizite Konsequenzregel, was uns mit Hilfe von deliverables eine transparente Handhabung von Hoare Tripeln ermöglicht.

Verifikation objekt-orientierter Programme mit Lego

Wolfgang Naraschewski
Universität Erlangen

Objekt-orientierte Programmiersprachen unterstützen *informelles* Schließen über Programme, da auf Objekte nur mit Hilfe von Methoden zugegriffen werden kann und deshalb für die Verifikation lediglich das Verhalten der Objekte bezüglich der Methodenaufrufe betrachtet werden muß. Interne Details der Implementierung werden somit nach außen verborgen.

Es wurde untersucht, ob die Vorteile, die die objekt-orientierte Programmierung für informelle Schlüsse bietet, auch von der *formalen* Verifikation ausgenützt werden werden können. Als formalen Rahmen haben wir den Calculus of Constructions (CC) gewählt, da man in diesem Kalkül sowohl funktionale Programme definieren als auch Beweise über sie führen kann. Außerdem existiert für CC maschinelle Unterstützung (LEGO), so daß die Definitionen und Beweise implementiert werden konnten. Die in CC definierbaren Programme sind zwar nicht à priori objekt-orientiert, aber Pierce und Turner haben in [PT92] gezeigt, daß objekt-orientierte Begriffe wie *Objekte*, *Klassen* und *Vererbung* in LEGO kodiert werden können.

Bei der Kodierung von Pierce und Turner ist das funktionale Verhalten der Objekte durch die Typen der Objekte (d.h. die Signatur der Methoden) noch nicht festgelegt. Somit sind Aussagen über *alle* Objekte eines bestimmten Typs nicht sinnvoll. Um dennoch Beweise führen zu können, darf lediglich über einen Teil dieser Objekte quantifiziert werden. Wir beschränkten uns darauf, jeweils nur Aussagen über Instanzen einer Klasse, d.h. Objekte mit der gleichen Implementierung, zu machen. Allerdings ist dafür die Kenntnis der Details der Implementation nötig. Die Vorteile, die die objekt-orientierte Programmierung bei informellen Schlüssen bietet, können mit dieser Beweistechnik also nicht auf die formale Verifikation übertragen werden.

Um Aussagen über alle Objekte eines Typs machen zu können, dürfen verschiedene Objekte des gleichen Typs nicht nur die gleiche Signatur besitzen, sondern sie müssen auch ähnliches Verhalten zeigen. Da das funktionale Verhalten bei der Kodierung von Pierce und Turner durch die Typen nicht festgelegt werden kann, müssen die Typen um eine Spezifikation erweitert werden. Objekte dieser neu definierten Typen beinhalten neben den Operationen auch Beweise, daß die Operationen die Spezifikation erfüllen.

Literatur

[PT92] Benjamin Pierce and David Turner. Object-oriented programming without recursive types. Technical Report ECS-LFCS-92-225, University of Edinburgh, August 1992. Principles of Programming Languages (POPL '93).

Konzepte der Spezifikationssprache *QED*

Detlef Schwier
Universität Ulm

Im Rahmen des KORSO-Projekts wird ein Spezifikations- und Verfikationssystem, das auf Luo's Typtheorie *ECC* basiert, entwickelt. Es wurden eine Reihe zusätzlicher syntaktischer Konstrukte in den Kalkül aufgenommen, um den speziellen Erfordernissen von Programmverifikationen gerecht zu werden. Ein Feature des Systems ist die automatische Erzeugung von Beweisverpflichtungen, wie es beim "klassischen" Spezifikationssystemen *PVS* der Fall ist. Für die bequeme Modellierung von Datentypen wurde ein "Datatyp"-Konstrukt aufgenommen, das auch rekursive Datentypen ermöglicht.

Beweisverpflichtungen ergeben sich meistens aus "coercions", mit denen der Typ eines Ausdrucks festgelegt wird. Für den semantischen Subtyp $ST = \{x : A \mid P(x)\}$ zum Beispiel erzeugt $a :: ST$ die Beweisverpflichtung $P(a)$. Erzeugen bedeutet hier, daß der Ausdruck a in das Subtypeelement (semantisch ein Paar) a[$:P(a)] umgewandelt wird. Die Metavariable $ kennzeichnet die Stelle an der der Beweiser einen Term vom Typ $P(a)$ einfügen muß. Mit einem ähnlichen Mechanismus werden ganze Spezifikationen, Implementierungen und Verfeinerungen von Spezifikationen behandelt.

Rewriting of Typed Terms applied to Category Theory and implemented in Elf

Wolfgang Gehrke
Johannes Kepler Universität Linz

Checking commutativity of diagrams in category theory is a task which arises permanently. Diagrams are a pictorial encoding of equations. In this context we will concentrate on tractable problems. The talk gives an outline how type theory is used to handle such equations in categories. On the one hand dependent types give a nice framework to represent morphisms in a category which are indexed by the corresponding source and target object. On the other hand we want to investigate a proof for equalities which is represented as a term where the type is the equality under consideration.

This approach can be implemented straightforward in Elf (a programming language based on the Edinburgh Logical Framework). Elf gives the right frame to express rewriting for typed terms (here morphisms in a category). Additionally the notion of critical overlaps is easily expressible in Elf. As an example monads (a basic notion from category theory) are implemented and it is shown automatically that the Kleisli construction gives a category. As a next step we would like to investigate the relationship of cartesian closed categories and monads which is exploited in purely functional programming.

Synthetische Domains in Typtheorie

Bernhard Reus (in Zusammenarbeit mit Thomas Streicher)
Ludwig-Maximilian-Universität München

Die Idee der synthetischen Domaintheorie besteht darin, Domains nicht als Mengen mit Struktur, sondern als Mengen mit gewissen Eigenschaften, aus denen dann die Ordnungsrelation definiert werden kann, zu axiomatisieren. In der Literatur gibt es dazu verschiedene Anstze von Hyland, Taylor, Phoa etc. Wir haben nun eine rein logische Axiomatisierung der *complete extensional pers* von Freyd et al. aufgestellt (siehe auch W. Phoa). Ein großer Vorteil der synthetischen Domaintheorie ist, daß alle Funktionen stetig sind. Dies ist jedoch inkonsistent mit klassischer Logik und so muß die Axiomatisierung in einer intuitionistischen Logik höherer Stufe erfolgen. Wir wählten den *extended calculus of constructions*, da er eine entsprechend reichhaltige Logik und in LEGO eine nutzbare Implementierung besitzt. Dies zeigt, daß man von wenigen Axiomen ausgehend die elementare (konstruktive) Domaintheorie entwickeln kann. Die Stärke der abhängigen Typen erlaubt es sogar, Funktoren zu modellieren und damit die inverse Limeskonstruktion für rekursiven Bereichsgleichungen durchzuführen. Die erhaltene Logik soll zu einer Art "verbessertem LCF" ausgebaut werden und letztlich in Verbindung mit Modultypen zur Verifikation modularer funktionaler Programme mit allgemeiner Rekursion dienen.
Ausgewählte Referenzen:

Literatur

[Freyd et al. 92] P. Freyd, P. Mulry, G. Rosolini, and D. Scott: *Extensional PERs*. Information and Computation 98, p. 211 -227 , 1992.

[Hyland 91] J.M.E. Hyland: *First Steps towards Synthetic Domain Theory*. In A. Carboni (ed.): Conference on Category Theory '90, volume 1488 of Lecture Notes in Mathematics. Springer Verlag, Berlin, 1991.

[Phoa 90] W. K.-S. Phoa: *Domain Theory in Realizability Toposes*. Ph. D. thesis, Trinity College, Cambridge, 1990.

[Reus, Streicher 93] . Reus, T. Streicher: *Naive Synthetic Domain Theory – A logical approach*. Unpublished manuscript, Universität München.
(ftp aus `ftp.informatik.uni-muenchen.de`
in Katalog `pst\papers\streicher+reus`))

[Rosolini 86] G. Rosolini: *Continuity and effectiveness in topoi*. Ph. D. Thesis, Carnegie-Mellon University, CMU-CS-86-123, 1986.

[Taylor 91] P. Taylor: *The Fixed Point Property in Synthetic Domain Theory.* Proc. of Logic in Computer Science, 1991, p. 152 - 160.

Modified Realizability Modelle für Intensionale Typtheorie

Thomas Streicher
LMU München

Basierend auf Kreisel's Modified Realizability wird ein Modell für eine imprädikative Version der Martin-Löf'schen Typtheorie konstruiert, in dem alle bekannten Sequenzen refutiert werden, die in der extensionalen Typtheorie (mit Identity Reflection Rule) herleitbar sind, sich aber einem formalen Beweis in der intensionalen Typtheorie (ohne Identity Reflection Rule) entziehen. Insbesondere werden folgende Kriterien für die Intensionalität eines Modelles angegeben und verifiziert:

(C1) $A : Set, x, y : A, z : Id(A, x, y) \vdash x = y : A$
gilt nicht
(C2) $A : Set, B : A \to Set, x, y : A, z : Id(A, x, y) \vdash B(x) = B(y)$
gilt nicht
(C3) Wenn $\vdash p : Id(A, t, s)$, dann $\vdash t = s : A$.

Es wird gezeigt, daß bei geeigneter Interpretation die Uniqueness of Elimination (d.h. η-Regeln für Eliminatoren) für alle induktiv definierten Typen nicht gelten. Beispielsweise gibt es in dem Modified Realizability Modell mehrere verschiedene Funktionen vom leeren Typ nach dem Typ N der natürlichen Zahlen. Es kann gezeigt werden, daß die extensional gleichen Funktionen $f, g : N \to N$, die wie folgt definiert sind :

$f(n) = n,$
$g(0) = 0 \quad g(n+1) = g(n) + 1,$

im Modell verschieden sind, weil sie sich auf sogenannten potentiellen Elementen, die nicht durch Terme denotiert werden können, verschieden verhalten (auf den aktualen Objekten hingegen verhalten sie sich aber gleich !).

Die Kategorie *mr-Set* der "modified realizability sets" erhält man aus der Kategorie der "realizability sets", indem Objekte definiert als Tripel $(X, X_{act}, \Vdash)$, so daß $(X, \Vdash)$ ein Realizability Set ist und X_{act} eine Teilmenge sogenannter "aktualer" Objekte von X ist, welche von den realisierbaren Morphismen erhalten werden müssen.

Die potentiellen, nicht-aktualen Elemente spielen dabei die Rolle von Nichtstandard- oder Errorelementen, die es gestatten extensional gleiche Funktionen (d.h. Funktionen die sich auf aktualen Obkjekten gleich verhalten) zu separieren. Insbesondere kann man nachweisen, daß in dem Modell das Extensionalitätsprinzip schon für Funktionen von N nach N nicht gilt.

Mithilfe des Modified Realizability Modells ist es also gelungen, Probleme der intensionalen Typtheorie, die operationaler Natur sind und aus Implementierungszwängen erwachsen (Entscheidbarkeit der Typisierbarkeit), mithilfe rein denotationaler semantischer Methoden zu analysieren.

Konstruktion semantischer Bereiche aus algebraischen Spezifikationen

Peter Kempf, Gunther Schmidt, Michael Winter
Universität der Bundeswehr München

Bei der theoretischen Untersuchung von Softwareentwicklung werden unter anderem Interpretationsmittel für Funktionen höherer Ordnung und algebraische Spezifikationen benötigt. Zur semantischen Beschreibung von Funktionen höherer Ordnung verwendet man Konstrukte wie algebraische cpo's. Algebraische Spezifikationen werden mit Hilfe der Termalgebra und einer Kongruenz, die durch die Gesetze induziert wird, interpretiert. Will man nun semantische Bereiche direkt aus algebraischen Spezifikationen gewinnen, so muß die Informationsordnung mit der induzierten Kongruenz verträglich sein. Frühere Arbeiten (siehe z.B. Möller (1985), Nelson (1982)) führten dabei eine Ordnung auf der dividierten Termalgebra ein und erhielten mittels Idealvervollständigung ein Modell. Wir zeigen, daß diese Methode zu ungewollten Effekten führt. Durch unseren Ansatz über die sogenannten vollständigen und approximationserhaltenden Kongruenzen auf Bereichen mit Keimen (siehe Schmidt (1989), Gunter (1985)) lassen sich diese Effekte vermeiden. Mit Hilfe dieser Theorie konstruieren wir aus einer algebraischen Spezifikation mit einer erweiterten Herleitungslogik über den inversen Limes direkt einen semantischen Bereich, der das initiale Modell in der Klasse der induktiv geordneten Modelle dieser Spezifikation ist.

Kategorielle Beschreibung des getypten Lambdakalküls

Lukas Alexander Erren
Technische Universität Berlin

Auf der Grundlage des getypten Lambdakalküls nach Barendregt 1984 hat die Arbeit zum Ziel, getypte Lambdakalküle als Modelle zu CCC's zu interpretieren. Dies wird dadurch erreicht, daß ein alternativer Termaufbau für Lambdaterme definiert wird, welcher zur Konstruktion geschlossener Ausdrücke ohne freie Variablen auskommt. Mithilfe des alternativen Aufbaus wird in einem einfachen Schritt ein zum getypten Lambdakalkül äquivalentes getyptes Kombinatorensystem konstruiert, und dieses als mengenbewertete CCC gedeutet. Die hier vorgestellten Arbeiten sind im Rahmen einer Studienarbeit an der TU Berlin entstanden, und wurden betreut von Martin Große-Rhode.

Frameworks: type-theoretic and other

Sean Matthews
Max-Planck-Institut für Informatik, Saarbrücken

There has over the last twenty years, been a lot of interest in the use of type theories as logical frameworks. However another older tradition, though less familiar perhaps to computer scientists, is that of Post and Smullyan. The most recent incarnation of this approach is the theory FS0, suggested by Feferman, which as a theory of recursively enumerable classes of s-expressions, rather than the more common Sigma-1 induction arithmetic, tries to remedy the deficiencies of earlier proposals in the tradition. FS0 is a framework that has different strengths and weaknesses than type theories, being intended primrarily for meta-theoretic analysis of implemented logics and theories (it is also not so heavily biased towards natural deduction style presentations of logics. On the other hand, it does not provide, as a primitive notion, substitution facilities that are distinctive of the type theoretic approach: the user must formalise mechanisms like this himself.

I have been working with FS0 to see how well, in practice, it works as a framework for doing metatheory.

Literatur

[1] Matthews, Sean, Smaill, Alan and Basin, David. Experience with FS0 as a Framework Theory, in *Logical Environments*, eds. Gerard Huet and Gordon Plotkin, Cambridge University Press, England.

[2] Matthews, Sean. A theory and its Metatheory in FS0, in 'What is a Logical System?', eds. Dov Gabbay and Franz Guenthner, Oxford University Press, England (to appear).

Program Synthesis as Higher-Order Resolution

David Basin
Max-Planck-Institut für Informatik, Saarbrücken

I show how higher-order resolution as embodied in Paulson's Isabelle system can be used to synthesize programs. Programs are built incrementally by reasoning about their relationship to their specification in an appropriate programming-logic. The program begins as a higher-order meta-variable and is incrementally instantiated by higher-order resolution during proof. This approach seems general enough to formalize many proposed and implemented "deductive synthesis" logics and strategies and has the advantages that it cleanly separates the underlying programming logic, derived rules used for synthesis, and possible heuristics for proof control. I will illustrate these ideas with an example in logic program synthesis where logic programs are build by applying derived rules in first-order logic. I will explain as well how this kind of proof can be automated.

Meta-Programmierung in Typtheorie

Harald Rueß
Universität Ulm

Wir präsentieren Techniken zur Formalisierung und Anwendung von Software-Entwicklungsschritten. Viele bekannte Transformationsregeln lassen sich durch typtheoretische Konzepte wie Abstraktion höherer Ordnung und abhängige Typen darstellen und als korrekt nachweisen. Als Beispiel verifizieren wir die Transformation *Split of Postcondition*, und zeigen, wie man solche Entwicklungsschritte in einem auf schrittweise Verfeinerungen basierenden Beweisersystem höherer Ordnung (z.B. Coq, Lego) auf konkrete Probleme anwenden kann (deduktive Programmsynthese).

Jedoch benötigen manche Entwicklungs- und Beweisschritte explizite Analyse und Manipulation der syntaktischen Kategorien des Kalküls. Um diese Formalisierungen durchführen zu können, reflektieren wir den Objektkalkül (CC) in sich selbst; diese Repräsentation konstitutiert die Meta-Ebene. In einem nächsten Schritt verbinden wir Objekt- und Meta-Ebene mit Hilfe von Reflektionsprinzipien, die uns erlauben, Ergebnisse zwischen Objekt- und Metaebene auszutauschen. In dieser Meta-Architektur formalisieren wir einen Taktik-Mechanismus, und weisen ihn als korrekt nach. Darüber hinaus lassen sich Entscheidungsprozeduren und Simplifikationsprozeduren in diesen Taktik-Mechanismus korrekt einbetten, ohne daß die jeweiligen Beweisterme explizit konstruiert werden müssen.

3 Teilnehmerliste

Thorsten **Altenkirch**
Chalmers University
Dept. of Computer Science
41296 Gothenburg, Schweden
alti@cs.chalmers.se

David **Basin**
Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken
basin@mpi-sb.mpg.de

Axel **Dold**
Universität Ulm
Abtl. Künstliche Intelligenz
Oberer Eselsberg
89069 Ulm
dold@informatik.uni-ulm.de

Lukas A. **Erren**
TU Berlin
Stargarder Str. 10
10437 Berlin
diogenes@cs.tu-berlin.de

Carsten **Führmann**
Universität Erlangen-Nürnberg
Hopfenleithe 21
91085 Weisendorf
cnfuehrm@
cip.informatik.uni-erlangen.de

Wolfgang **Gehrke**
RISC
Johannes-Kepler-Universität Linz
A-4040 Linz
wgehrke@risc.uni-linz.ac.at

Friedrich **von Henke**
Universität Ulm
Abtl. Künstliche Intelligenz
Oberer Eselsberg
89069 Ulm
vhenke@informatik.uni-ulm.de

Martin **Hofmann**
University of Edinburgh
Kings Building JCMB Room 1403
Mayfield Road
Edingburgh EH9 3JZ, Schottland
mxh@dcs.ed.ac.uk

Christian **Horn**
Fachhochschule Furtwangen
FB Allgemeine Informatik
Gerwigstr. 11
78120 Furtwangen
ch@ai-lab.fh-furtwangen.de

Peter **Kempf**
Universität der Bundeswehr
Fakultät für Informatik
Werner-Heisenberg-Weg 39
85577 Neubiberg
peter@informatik.unibw.muenchen.de

Michael **Kohlhase**
Fachbereich Informatik
Universität des Saarlandes
Postfach 151 150
66041 Saarbrücken
kohlhase@cs.uni-sb.de

Christoph **Kreitz**
TH Darmstadt
FB Informatik/FG Intellektik
Alexanderstraße 10
64283 Darmstadt
kreitz@intellektik.informatik.th-darmstadt.de

Junbo **Liu**
Universität Bremen
Fachbereich 3 (Informatik)
Postfach 330 440
28334 Bremen
liu@informatik.uni-bremen.de

Jacques **Loeckx**
Fachbereich 14 (Informatik)
Universität des Saarlandes
Postfach 151 150
66041 Saarbrücken
loeckx@cs.uni-sb.de

Sean **Matthews**
Max-Planck-Insitut für Informatik
Im Stadtwald
66123 Saarbrücken
sean@mpi-sb.mpg.de

Stephan **Missura**
ETH Zürich
Institut für Theoretische Informatik
IFW B48.2
CH-8092 Zürich
missura@inf.ethz.ch

Wolfgang **Naraschewski**
Universität Erlangen-Nürnberg
Himmelreichstraße 8
96120 Bischberg
wgnarasc@
cip.informatik.uni-erlangen.de

Michael **Rathjen**
Universität Münster
Inst. für Mathematische Logik
Einsteinstraße 62
48149 Münster
rathjen@math.uni-muenster.de

Bernhard **Reus**
Universität München
Institut für Informatik
Leopldstr. 11b
80802 München
reus@informatik.uni-muenchen.de

Harald **Rueß**
Universität Ulm
Abtl. Künstliche Intelligenz
Oberer Eselsberg
89069 Ulm
ruess@informatik.uni-ulm.de

Thomas **Schreiber**
University of Edinburgh
Kings Building JCMB
Mayfield Road
Edingburgh EH9 3JZ, Schottland
tms@dcs.ed.ac.uk

Detlef **Schwier**
Universität Ulm
Abtl. Künstliche Intelligenz
Oberer Eselsberg
89069 Ulm
schwier@informatik.uni-ulm.de

Thomas **Streicher**
Universität München
Institut für Informatik
Leopldstr. 11b
80802 München
streiche@informatik.uni-muenchen.de

Martin **Strecker**
Universität Ulm
Abtl. Künstliche Intelligenz
Oberer Eselsberg
89069 Ulm
strecker@informatik.uni-ulm.de

Terry **Stroup**
Universität Erlangen-Nürnberg
Himmelreichstrae 8
96120 Bischberg
terry@cip.informatik.uni-erlangen.de

Andreas **Weber**
Wilhelm-Schickard-Inst. für Informatik
Auf dem Sand 13
72076 Tübingen
weber@informatik.uni-tuebingen.de

Michael **Winter**
Universität der Bundeswehr
Fakultät für Informatik
Werner-Heisenberg-Weg 39
85577 Neubiberg
trash@hermes.unibw-muenchen.de