Kalkülbasierte OO-Sprachen

Uwe Nestmann* Martin Steffen†

Januar 1994

IMMD VII, Universität Erlangen-Nürnberg
Interner Bericht

Zusammenfassung

Der Text stammt aus einem Entwurf für das SFB-C2-Projekt. Er beschreibt Motivation und Stand der Kunst in dem Bereich formaler Methoden für Objektorientierte Sprachen, speziell im Bereich der Kalküle für OO.

1 Einleitung

Verifikation von Programmen ist nur möglich, wenn die Programmiersprache mit einer formalen Semantik ausgestattet ist. Für *objektorientierte Programmiersprachen,* insbesondere für parallele bzw. verteilte Systeme, gilt dies in gleicher Weise. Im TP C2 verfolgen wir die Strategie der Semantikdefinition durch natürliche Abbildungen in bekannte, wohluntersuchte Basiskalküle. Der folgende Überblick skizziert die gegenwärtig existierenden Ansätze in diesem Gebiet und zeigt auf, welche Art von Verifikation damit bisher ermöglicht wurde.

*University of Erlangen-Nürnberg, IMMD VII, Martensstr. 3, 91058 Erlangen, Germany. Electronic mail: nestmann@informatik.uni-erlangen.de.

†University of Erlangen-Nürnberg, IMMD VII, Martensstr. 3, 91058 Erlangen, Germany. Electronic mail: steffen@informatik.uni-erlangen.de.

2 Semantik verteilter objektorientierter Programme

Semantik durch Basiskalküle

Eine Möglichkeit, die Semantik einer Sprache zu festzulegen, ist die Semantik durch *Übersetzung* in einen zugrundeliegenden Basiskalkül. Die auf diese Weise erhaltenen Semantiken erscheinen oft einfacher als übliche denotationelle Semantiken. Dies weist darauf hin, daß wenige einfache Grundkonzepte – Benutzung von Namen und privaten Verbindungen, geeignete Behandlung von Gültigkeitsbereichen, Replikation – zur uniformen Darstellung von Objekten, Methoden und Klassen ausreichen. Ein weiterer Vorteil besteht darin, daß diese Vorgehensweise einzelne Sprachmerkmale – und gerade im Bereich objektorientierter Programmierung besteht alles andere als Einigkeit darüber, was nun als der Kanon objektorientierter Features auszusehen sei – in Isolation zu betrachten gestattet, Sprachmerkmale, die jeweils durch eine bestimmte strukturelle Übersetzung festgelegt sind. Darüberhinaus sind Erweiterungen der Zielsprachen oder Änderungen ihrer Semantik leichter durchführbar, da nur die Kodierung geändert werden muß. Dies ist vor allem dann gewinnbringend, wenn der zugrundeliegenden Kalkül nicht alleine die Semantik bereitstellt, sondern durch seine lauffähige *Im-*

plementierung auch ein experimentelles Testbett für den Sprachentwurf.

Der Nutzen in Hinblick auf die *Verifikation* liegt darin, daß zum einen die Semantik, auf die sich die Verifikation ja beziehen muß, in wohlverstandenen Basiskalkülen handhabbarer erscheint. Zum anderen und vor allem, daß man, sofern nur die Kodierung natürlich genug ist, sich die Theorie des zugrundeliegenden Kalküls auf der Zielsprachebene zunutze machen kann; im Falle einer prozeßtheoretischen Semantik beispielsweise die entsprechenden algebraischen Gesetze. Bei einem zugrundeliegenden typentheoretische Kalkül bekommt man theoretische Eigenschaften wie z.B. Typsicherheit oder Werkzeuge wie Typechecker durch die Kodierung gratis auch für die Zielsprache.

Sequentielle OO-Sprachen & Typentheorie

Die Entwicklung ausdrucksstarker, statischer Typsysteme stellt einen wichtigen Beitrag zur Entwicklung und Fundierung moderner Programmiersprachen dar. Funktionale Programmiersprachen waren aufgrund ihrer konzeptionellen Klarheit traditionell das Gebiet der weitestentwicketen Typsysteme. Objektorientierte Sprachen bieten erweiterte Möglichkeiten wie *Objektkapselung*, *Klassen*, *Vererbung*, *Methodenaufrufe* [Weg87], die sich in rein funktionalen Sprachen nicht finden lassen. Da sich die Untersuchung statischer Typsysteme auf dem Gebiet funktionaler Programmiersprachen am weitesten fortgeschritten ist, ist ein wichtiger Forschungsrichtung, die Entwicklung OO-Formalismen und Sprachmerkmalen auf Basis typisierter λ-Kalküle meist höherer Ordnung. Ausgangspunkt vieler Untersuchungen war die einflußreiche Arbeit [CW85], bzw. der Kalkül $F_{\leq}$, eine Erweiterung des λ-Kalküls zweiter Ordnung [Gir72, Rey74] um Untertypen und mit beschränkter Quantifizierung. Gemeinsam diesen Arbeiten ist, daß Objekte in irgendeiner Form durch *Records* ihrer Methoden modelliert werden. Da sich der ursprüngliche Kalkül $F_{\leq}$ bald als zu schwach herausstellte [RT88], um zustandsverändernde Funktionen in Zusammenhang mit beschränkter Quantifikation und Objektkapselung zu modellieren, zielt eine Reihe von Erweiterungen darauf ab, die sich zum Teil widersprechenden Anforderungen für *Kapselung*, *Vererbung*, *Polymorphie*, *Untertypen* und *späte Bindung* bzw. *Nachrichtenaustausch*, um nur einige zu nennen, unter einen Hut zu bringen. Mehrere Arbeiten konzentrierten sich darauf, die Einschränkung auf Records fester Länge durch Integration geeigneter Manipulationsmöglichkeiten für Records —Hinzufügen und Entfernen eines Feldes, überschreiben eines existierenden Feldes, Konkatenation zweier Verbünde— zu überwinden, so z.B. [Car88] [CM91] [CHC90] [Rém92] [Wan91] [Wan94] [Car92]. Der Nachteil ist die relative Komplexität der Kalküle, ihrer Modelle sowie ihrer Metatheorie, was u.a. darauf zurückzuführen ist, daß man hierbei zur Typisierung von Objekten auf rekursive Typen angewiesen ist. Ein anderer, wichtiger Ansatzpunkt versucht daher, ohne rekursive Typen auszukommen [PT94] [PT92] was zu einer deutlichen Vereinfachung der Kalküle führt. Die Idee ist, daß die Typen der Objekte existentielle Typen seien, womit ihr interner Zustand nur durch die Methodenschnittstelle des Objektes erreichtbar ist. Die Verwendung existentieller Typen zur Kapselung, wenn auch nicht im objektorientierten Rahmen, stammt von [MP88]. Im Mittelpunkt dieser Bemühungen steht das System $F^{\omega}_{\leq}$ [Car90] [CM91] [BM92]. Darüberhinausgehende Erweiterungen dieses Systems um Schnittypen (*intersection types*) [Pie97] [CP96] dienen der Modellierung von *Mehrfachvererbung*. Ein anderer Weg um mit den Problem der zustandsverändernder Methoden umzugehen wird in [HP95a] eingeschlagen: Ein positives Fragment von $F_{\leq}$ wird verwendet, um Zuständsveränderung modellieren zu können. Um ebenfalls eine Erweiterung des λ-Kalküls, in diesem Falle des einfach typisierten λ-Kalküls um abhängi-

ge Typen, wird in [CGL95] [Cas93] untersucht. Dadurch wird eine Trennung von Typen zur Compilezeit und Typen zur Laufzeit erreicht, um *overloading* und *multiple dispatch* wie in CLOS [KG89] ausdrücken zu können. Diese Arbeit wurde weiterentwickelt in [Tsu94]. In [CCHO89] [Bru94] [CHC90] [Mit90] werden Objekte durch Records und mit Hilfe sogenannter *F-beschränkter* Quantifikation dargestellt. Dieser Ansatz führte zu bereits relativ komplexen, objektorientierten Sprachentwürfen [Bru94].

Eine Spielart der Vererbung nicht über Klassen, sondern durch *Delegation* und *Methodenspezialisierung* wird in [Mit90] [FHM94] formalisiert.

Von höherer Warte als die genannten Ansätze aus wird die Formalisierung in [HP92] [HP95b] angegangen, indem keine konkrete Kodierung objektorientierter Eigenschaften oder ein speziell entwickelter Kalkül präsentiert wird, sondern abstrakte Forderungen an das Verhalten von Objekten und Klassen formuliert werden. Dabei wird gezeigt, daß eine Reihe der oben genannten Kodierungen diesen Anforderungen genügt.

Eine aktuelle Zusammenstellung der wichtigsten Arbeiten zur Semantik objektorientierter Sprachen auf typentheoretischem Gebiet findet sich in [GM94], ein Überblick über den Stand der Forschung in [DT88] oder, aktueller, in [FM94].

3 Verifikation verteilter objektorientierter Programme

Verifikation mit direkter Semantik

Eine Reihe von Untersuchungen bemüht sich darum, konventionelle Methoden der Verifikation imperativer Programme auf *sequentielle* objektorientierte Sprachen zu übertragen. Ein Ausgangspunkt dabei sind dabei die Arbeiten zu sogenannten Verfeinerungskalkülen. Diese stellen eine Erweiterung von Dijktras *guarded-command* Sprache [Dij76] dar, und zwar um die Möglichkeit der Verfeinerung, mit der man ausgehend von der abtrakten Spezifikation durch eine Reihe von Transformationen ein ausführbares Programm erhalten kann [Bac78][Mor88]. Neuere Arbeiten versuchen, diese Idee der transformationellen Verfeinerung auf objektorientierte Sprachen zu übertragen. In [Lea88] werden Bedingungen an die Untertyprelation formuliert, die *modulare* Beweise in Zusammenhang mit Untertyppolymorphie und *late-binding* von Methoden ermöglichen. Allerdings ist die Anwendung dieses Verifikationskalküls auf Objekte mit unveränderlichem Zustand in einer Sprache ohne Wertzuweisungen beschränkt. In [Utt92] [UR93] wird dies auf Objekte mit veränderlichem Zustand erweitert, allerdings um den Preis, auf Datenverfeinerung zu verzichten. Alles in allem stehen die Bemühungen, insbesondere was verteilte Programmierung in diesem Gebiet betrifft, erst am Anfang.

Beweissysteme für **parallele**, objektorientierte Sprachen existieren bisher immer noch lediglich für die POOL-Familie [Ame86]. Die Einführung von Parallelität verkompliziert die Verifikation noch beträchtlich, insbesondere in Zusammenhang mit der Möglichkeit dynamischer Prozeßerzeugung, für die in [dB91] eigens dazu erforderliche Beweisregeln entwickelt wurden. Allerdings sind derartige Beweissysteme nicht kompositionell, da es für die Dekomposition des Beweisproblems bei dynamischer Konfigurierung der Objekte bisher keine Ansätze gibt. Auch für die Behandlung von Vererbung und Untertypen in diesen Sprachen in Bezug auf Verifikation gibt es noch keine zufriedenstellende Lösung. Die Anwendung auf größere Programme, bestehend aus einer Anzahl interagierender, paralleler Objekte, blieb bisher aus.

Verifikation im Basiskalkül

Im Rahmen der **Typentheorie** von $F^{\omega}_{\leq}$ wurden Eigenschaften eines Objektmodells in

dem Sinne verifiziert, daß das aufeinanderfolgende Aufbrechen der Kapselung und Wiederzusammenfügen eines Objekts den ursprünglichen Zustand stets wiederherstellt. Dieser Beweis wurde sowohl für die Kodierung mit rekursiven Typen als auch für die mit existentiellen Typen erbracht. Damit wurde gezeigt, daß diese Eigenschaften von Objekten sehr präzise von einem Standard-Typkalkül formuliert werden kann und nicht als primitives Konzept der Sprache eingeführt werden müssen.

Aktive Objekte, deren Semantik mittels mobiler **Prozeßtheorie** im π-Kalkül beschrieben wurden, waren aufgrund der Aktualität der Forschung ebenfalls bisher nur Gegenstand einiger weniger Untersuchungen zur Verifikation. In [Wal94] und [LW95] wurden erste formale Verifikationsergebnisse erzielt. Beide behandeln ein Programmtransformationsproblem nach [Jon94], bei welchem die Korrektheit der Transformation auf der Objektebene durch Argumente auf der π-Kalkül-Ebene verifiziert wird. Konkrete Ergebnisse waren bisher für eingeschränkte, reguläre Objektstrukturen möglich und führten mittlerweile zu einer Erweiterung der Theorie der mobilen Prozesse um den Begriff der Konfluenz.

Formale Programmentwicklung in Typentheorie

Daß formale Programmentwicklung am günstigsten *schrittweise* durch eine Folge von horizontalen und vertikalen Verfeinerungen vollzogen wird, ist heutzutage, zumindest theoretisch, ein Gemeinplatz. Insbesondere im Rahmen der Theorie der Algebraischen Spezifikation wurden hier viele konkurrierende Ansätze verwirklicht (als aktuelle Sammlung von Arbeiten s. [BKL$^+$91]), die bei der Darstellung der Beziehung zwischen *Programmen* und ihren *Spezifikationen* zumeist auf allgemeiner **Mengenlehre** beruhen. Ein Manko dieser Ansätze ist, daß die *Beweise,* daß Programme ihren Spezifikationen genügen, nicht innerhalb dieses formalen Rahmens vertreten sind.

Typentheorie dagegen erlaubt die Integration von Programmen, Spezifikationen *und* Beweisen in demselben formalen Rahmen. Ermöglicht wird dies dadurch, daß man die Typen, ist ihre Theorie nur mächtig genug, als Formeln einer —meist höheren— Logik betrachten kann. Entsprechend dieser Analogie von Formeln und Typen, auch als *Curry-Howard-Isomoprhie* [How80] [CF58] [GLT89] bekannt, sind die „Bewohner“ der Typen sowohl Programme als auch Beweise, daß die Programme ihre Spezifikation erfüllen. Verifikation wird zum *type-checking.*

Ein weiterer Vorteil dabei ist, daß innerhalb dieses Rahmens mit Hilfe eines typentheoretischen Werkzeugs wie beispielsweise Lego die Beweise Hand in Hand mit dem Programm aus der Spezifikation entwickelt und maschinengestützt sofort auf ihre Korrektheit streng formal überprüft werden können. Daher spricht man auch häufig von **Ko-Entwicklung von Programmen mit ihren Beweisen**. Solche Verfahren sind, wohlgemerkt, nicht vollautomatisch, sondern werden interaktiv vom mathematisch ausgebildeten Programmentwickler eingesetzt.

Literatur

[Ame86] Pierre America. A proof theory for a sequential version of POOL. ESPRIT Project 415 Document 188, Philips Research Laboratories, Eindhoven, the Netherlands, 1986.

[Bac78] Ralph-Johan R. Back. *On the Correctness of Refinement in Program Development.* PhD thesis, Department of computer Science, University of Helsinki, 1978.

[BKL$^+$91] M. Bidoit, H.-J. Kreowski, P. Lescanne, F. Orejas, and D. Sannella, editors. *Algebraic System Speci-*

fication and Development, volume 501 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[BM92] Kim Bruce and John Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In POPL'92 [POP92], pages 316–327.

[Bru94] Kim B. Bruce. A paradigmatic object-oriented programming language: Design, static typing and semantics. *Journal of Functional Programming*, 4(2), April 1994. A preliminary version appeared in POPL 1993 under the title "Safe Type Checking in a Statically Typed Object-Oriented Programming Language", and as Williams College Technical Report CS-92-01.

[Car88] Luca Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988. A preliminary version appeared in [KMP84].

[Car90] Luca Cardelli. Notes about $F^{\omega}_{<:}$. Unpublished manuscript, October 1990.

[Car92] Luca Cardelli. Extensible records in a pure calculus of subtyping. Technical Report 81, Digital Equipment Corporation, System Research, 1992. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

[Cas93] Guiseppe Castagna. A metalanguage for typed object-oriented languages. In R. K. Shyamasundar, editor, *Proceedings of FSTTCS '93*, volume 761 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1993.

[CCHO89] Peter Canning, William Cook, Walt Hill, and Walter Olthoff. Interfaces for strongly-typed object-oriented programming. In *Object Oriented Programing: Systems, Languages, and Applications (OOPSLA)*, pages 457–467, 1989.

[CF58] Haskell B. Curry and Robert Feys. *Combinatory Logic*. North-Holland, 1958.

[CGL95] Guiseppe Castagna, Giorgio Ghelli, and Guiseppe Longo. A calculus for overloaded functions with subtyping. *Information and Computation*, 117(1):115–135, February 1995. A Preliminary Version appeared in ACM Conference on LISP and Functional Programming, June 1992 (pp. 182–192) and as LIENS Rapport de Recherche, LIENS-92-4, 1992.

[CHC90] William R. Cook, Walter L. Hill, and Peter S. Canning. Inheritance is not subtyping. In POPL'90 [POP90], pages 125–135. Also in the collection [GM94].

[CM91] Luca Cardelli and John Mitchell. Operations on records. *Mathematical Structures in Computer Science*, 1:3–48, 1991. Also in the collection [GM94]; available as DEC Systems Research Center Research Report #48, August, 1989, and in the proceedings of MFPS '89, Springer LNCS volume 442.

[CP96] Adriana B. Compagnoni and Benjamin C. Pierce. Higher-order intersection types and multiple inheritance. *Mathematical*

Structures in Computer Science, 6(5):469–501, October 1996. Preliminary version available as University of Edinburgh technical report ECS-LFCS-93-275 and Catholic University Nijmegen computer science technical report 93-18, Aug. 1993, under the title *Multiple Inheritance via Intersection Types*.

[CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

[dB91] Frank S. de Boer. A proof system for the language POOL. In Jaco W. de Bakker, Willem-Paul de Roever, and Grzegorz Rozenberg, editors, *Foundations of Object-Oriented Languages (REX Workshop)*, volume 489 of *Lecture Notes in Computer Science*, pages 124–150. Springer-Verlag, 1991.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[DT88] Scott Danforth and Chris Tomlinson. Type theories and object-oriented programming. *Computing Surveys*, 20(1):29–72, 1988.

[FHM94] K. Fisher, F. Honsell, and J. C. Mitchell. A lambda calculus of objects and method specialization. *Nordic Journal of Computing*, 1:3–37, 1994. Preliminary version appeared in *Proc. IEEE Symp. on Logic in Computer Science,* 1993, 26–38.

[FM94] Kathleen Fisher and John Mitchell. Notes on typed object-oriented programming. In Hagiya and Mitchell [HM94], pages 844–885.

[Gir72] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupure dans l'arithmetique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[GM94] Carl A. Gunter and John C. Mitchell. *Theoretical Aspects of Object-Oriented Programming, Types, Semantics, and Language Design*. Foundations of Computing Series. MIT Press, 1994.

[HM94] M. Hagiya and John C. Mitchell, editors. *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[How80] W. A. Howard. The formulae-as-types-notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[HP92] Martin Hofmann and Benjamin Pierce. An abstract view of objects and subtyping. Technical Report ECS-LFCS-92-225, Laboratory for Foundations of Computer Science, University of Edinburgh, August 1992.

[HP95a] Martin Hofmann and Benjamin Pierce. Positive subtyping. In *22nd Annual Symposium on Principles of Programming Languages (POPL)*, pages 186–197. ACM,

January 1995. Full version in *Information and Computation*, volume 126, number 1, April 1996. Also available as University of Edinburgh technical report ECS-LFCS-94-303, September 1994.

[HP95b] Martin Hofmann and Benjamin Pierce. A unifying type-theoretic framework for objects. *Journal of Functional Programming*, 5(4):593–635, October 1995. Previous versions appeared in the Symposium on Theoretical Aspects of Computer Science, 1994, (pages 251–262) and, under the title "An Abstract View of Objects and Subtyping (Preliminary Report)," as University of Edinburgh, LFCS technical report ECS-LFCS-92-226, 1992.

[Jon94] Cliff Jones. Process algebra arguments about an object-based design notation. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 1–15. Prentice Hall, 1994.

[KG89] Sonya Keene and Dan Gerson. *Object-Oriented Programming in Common Lisp.* Addison-Wesley, Reading, Massachusetts, 1989.

[KMP84] Gilles Kahn, David MacQueen, and Gordon Plotkin, editors. *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

[Lea88] Gary T. Leavens. *Verifying Object-Oriented Programs that use Subtypes.* PhD thesis, Massachusetts Institute of Technology, 1988.

[LW95] Xinxin Liu and David Walker. Confluence of processes and systems of objects. In Peter D. Mosses, Mogens Nielsen, and Michael I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 217–231. Springer-Verlag, 1995. Presented in the CAAP-section. Available as University of Warwick Research Report CS-RR-272, October 1994.

[Mit90] John C. Mitchell. Toward a typed foundation for method specialization and inheritance. In POPL'90 [POP90], pages 109–124. Also in the collection [GM94].

[Mor88] Carroll C. Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3):403–419, July 1988.

[MP88] John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, July 1988.

[Pie97] Benjamin C. Pierce. Intersection types and bounded polymorphism. *Mathematical Structures in Computer Science*, 7(2):129–193, April 1997. Summary version appeared in Conference on Typed Lambda Calculi and Applications, March 1993, pp. 346–360. Preliminary version available as University of Edinburgh technical report ECS-LFCS-92-200.

[POP90] ACM. *Seventeenth Annual Symposium on Principles of Programming Languages (POPL)*, January 1990.

[POP92] ACM. *Nineteenth Annual Symposium on Principles of Program-*

ming Languages (POPL), January 1992.

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

[PT94] Benjamin Pierce and David Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, April 1994. A preliminary version appeared in Principles of Programming Languages, 1993, and as University of Edinburgh technical report ECS-LFCS-92-225, under the title "Object-Oriented Programming Without Recursive Types".

[Rém92] Didier Rémy. Typing record concatenation for free. In POPL'92 [POP92], pages 166–176. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

[Rey74] John Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Colloque sur la programmation (Paris, France)*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, 1974.

[RT88] Edmund Robinson and Robert Tennent. Bounded quantification and record-update problems. Message to `Types` electronic mail list, October 1988.

[Tsu94] Hideki Tsuiki. A normalizing calculus with overloading and subtyping. In Hagiya and Mitchell [HM94], pages 273–295.

[UR93] Mark Utting and Ken Robinson. Modular reasoning in an object-oriented refinement calculus. In R. S. Bird, C. C. Morgan, and J. P. C. Woodcock, editors, *Mathematics of Program Construction 1992*, volume 669 of *Lecture Notes in Computer Science*, pages 344–367. Springer-Verlag, 1993.

[Utt92] Mark Utting. *An Object-oriented Refinement Calculus with Modular Reasoning.* PhD thesis, University of New South Wales, Australia, 1992.

[Wal94] David Walker. Algebraic proofs of properties of objects. In Don Sannella, editor, *Proceedings of ESOP '94*, volume 788 of *Lecture Notes in Computer Science*, pages 501–516. Springer-Verlag, 1994.

[Wan91] Mitchell Wand. Type inference for record concatenation and multiple inheritance. *Information and Computation*, 92:1–15, 1991. A preliminary version appeared in the Proceedings of LICS'89, pages 92–97.

[Wan94] Mitchell Wand. Type inference for objects with instance variables and inheritance. In *Theoretical Aspects of Object-Oriented Programming, Types, Semantics, and Language Design* [GM94], pages 97–120. Also appeared as Northeastern University Technical Report, NU-CCS-89-2, 1989.

[Weg87] Peter Wegner. Dimensions of object-based language design. In *Object Oriented Programing: Systems, Languages, and Applications (OOPSLA) '87 (Orlando, FL)*, pages 168–182. ACM, 1987. In *SIGPLAN Notices* 22(12).