

**Case for Support – Anton Setzer**

**Extensions of Dependent Type Theory –  
Induction, Interaction, Universes**

**October 5, 2002**

Department of Computer Science  
University of Wales Swansea  
Singleton Park  
Swansea SA2 8PP

## Part 1 Previous Research Track Record

Anton Setzer got his first degree (Dipl.math, with distinction) and his PhD (magna cum laude) from the University of Munich. He has been a lecturer at the university of Swansea since November 1, 2000. Before he had a position as teaching assistant in Munich and as “forskar assistant” (research associate) in Uppsala, Sweden. He had invitations for longer research visits at the Mittag-Leffler Institute in Stockholm (Royal Academy of Sciences, Sweden), twice at the university of Hiroshima in Japan (funded by the Royal Academy of Sciences and the Japanese research council JSPS), at the university of Gothenburg, Sweden, the university of Stockholm (invitation by P. Martin-Löf), the university of Berne, Switzerland and the university of Leeds. Further, he had a grant for a Research in Pairs Project at the research institute in Oberwolfach. He has been invited to the conference “Mathematical Logic” in Oberwolfach 1995, 1998 and 2002, the Dagstuhl meetings “Dependent type theory meets practical programming” and “Proof Theory in Computer Science” 2001. He was plenary speaker at the Logic Colloquium 1998 in Prague, special session speaker at the Logic Colloquium 1997 in Leeds. He was invited speaker at JIG4 in Kobe, Japan 2000, “Proof and Computation”, Munich, 1999, “Proof and computation”, Leeds, 1999, “Operations, sets and types”, Italy, 1998, “Symposium on types and ordinal notation systems”, Uppsala, Sweden, 1996, “Twenty-five years of constructive type theory”, Venice, 1995, “International Gauß Symposium”, Munich, 1993. Besides this, he has contributed talks to 1-2 computer science and logic conferences per year. He has been a frequent referee for the Annals of Pure and Applied Logic, the Journal of Symbolic Logic, the Archive for Mathematical Logic, the conference Computer Science Logic (CSL), typed lambda calculi and applications (TLCA) and other computer science conferences and workshops. He has been a reviewer for the Bulletin of Symbolic Logic, and is a regular reviewer for Zentralblatt MATH and Mathematical Reviews.

He has a grant from the Nuffield Foundation, 31.v.2001 - 31.v. 2003 with title “Proof-theoretically strong extensions of Martin-Löf type theory and applications”.

### Research Summary.

The main focus of the research of AS is on **dependent type theory (DTT)**, especially Martin-Löf Type Theory, with an emphasis on proof theoretic methods.

DTT extends the concept of a simple type, as it occurs in most programming language, by allowing types to depend on elements (programs) of other types. Therefore, one can specify the type of programs, the output of which has a property depending on its input. This allows to specify and guarantee the correctness of programs. Martin-Löf Type Theory is one version of type theory, in which types are built predicatively (from below).

*Proof theory* is a discipline of logic in which the limit of what can be proved in theories is explored and measured by the proof theoretic ordinal.

In his thesis [8, 11], AS used proof theory in order to measure the strength of Martin-Löf type theory. Later he has worked on proof theoretic strong extensions of it, and developed especially the Mahlo universe [14], the strength of which exceeds any standard extensions of Martin-Löf type theory. Together with P. Dybjer, he has developed a closed formalisation of inductive-recursive (IR) definitions. IR definitions subsume all standard extensions of Martin-Löf type theory and include the usual inductive data types. The closed formulation contains a data type the elements of which are codes for all inductive-recursive definitions ([2, 3, 4, 5]). He has in collaboration with P. Hancock ([6]) developed concepts for formalizing interactive programs in DTT. They added new constructs like while- and repeat-loops and a refinement construct to it. The latter allows to translate high-level programs into low-level implementations.

He has developed a novel concept for developing ordinal notation systems in proof theory, called ordinal systems ([13, 15]). Further, he has investigated the strength of fixed point theories [7], worked on the proof theory of inductive definitions ([9, 10] and on the pigeon hole principle and upper bounds for hierarchies of propositional proof systems ([1]).

### Selected publications

- [1] P. Clote, A. Setzer: On PHP st-connectivity, and odd charged graphs. In: P. Beame, S. Buss (Eds.): *Proof complexity and feasible arithmetics, DIMACS workshop April 21 - 24, 1996*. American Mathematical Society, 1998, pp. 73 - 92.
- [2] P. Dybjer, A. Setzer: *Finite axiomatizations of inductive and inductive-recursive definitions*. In: Workshop on Generic Programming, Marstrand, Sweden, 1998.
- [3] P. Dybjer, A. Setzer: *A finite axiomatization of inductive-recursive definitions*. In: Girard, J.-Y. (Ed.): *Typed Lambda Calculi and Applications*. Proceedings of TLCA'99. LNCS 1581, Vol. 1581, 1999, pp. 129 - 146.
- [4] P. Dybjer, A. Setzer: *Indexed Induction-Recursion*. In: R. Kahle, P. Schroeder-Heister, R. Stärk (Eds.): *Proof Theory in Computer Science*. LNCS 2183, pp. 93 - 113, 2001.
- [5] P. Dybjer, A. Setzer: *Induction-Recursion and initial algebras*. To appear in *Annals of Pure and Applied Logic*, 2002 or later.
- [6] P. Hancock, A. Setzer: Interactive programs in dependent type theory. In: P. Clote, H. Schwichtenberg: *Computer Science Logic*. LNCS 1862, pp. 317 - 331, 2000.
- [7] G. Jäger, R. Kahle, A. Setzer, T. Strahm: *The proof-theoretic analysis of transfinitely iterated fixed point theories*. *Journal of Symbolic Logic*, 64, 1: 53 - 67, 1999.
- [8] A. Setzer: Proof theoretical analysis of Martin-Löf type theory with W-type and one Universe. PhD Thesis, 140 pp, Munich 1993.
- [9] A. Setzer: Translating set theoretical proofs into type theoretical programs. In: G. Gottlob, A. Leitsch, D. Mundici (Eds.): *Computational logic and proof theory. 5th Kurt Gödel Colloquium, KGC' 97*. LNCS 1289, 1997, pp. 278 - 289.
- [10] A. Setzer: Inductive definitions with decidable atomic formulas. In: D. van Dalen, M. Bezem (Eds.): *Computer science logic. 10th International workshop CSL '96*. LNCS 1258, 1997, pp. 414 - 431.
- [11] A. Setzer: *Well-ordering proofs for Martin-Löf type theory with W-type and one Universe*. *Annals of Pure and applied Logic*, 92:113 - 159, 1998.
- [12] A. Setzer: An introduction to well-ordering proofs in Martin-Löf's type theory. In: G. Sambin and J. Smith (Eds.), *Twenty-five years of constructive type theory*, Clarendon Press, Oxford, 1998, p. 245 - 263.
- [13] A. Setzer: Ordinal systems. In: B. Cooper and J. Truss (Eds.): *Sets and proofs*, Cambridge University Press, 1999, pp. 301 - 338.
- [14] A. Setzer: *Extending Martin-Löf type theory by one Mahlo-universe*. *Arch. Math. Log* 39: 155 - 181, 2000.
- [15] A. Setzer: Ordinal systems. Part 2: One inaccessible. In: S. Buss, P. Hajek, P. Pudlak (Eds.): *Logic Colloquium '98*. ASL Lecture Notes in Logic 13, Peters Ltd, Natick, Massachusetts, 2000, pp. 426 - 448.

### Computer Science at Swansea

The Department of Computer Science (Grade 5 in RAE 2001) at the University of Wales Swansea is internationally well known for its research in logic in computer science and theoretical computer science. The main collaborator in the department will be U. Berger, who has a strong background in type theory, proof theory, computability theory, and synthesis of correct programs from proofs. The project will as well benefit from the expertise of the other members of the department, in particular J.V. Tucker (head of department, logical and algebraic methods for specification, computability theory), P.W. Grant (functional programming, parallel processing), N.A. Harman (term rewriting and hardware verification), O. Kullmann (satisfiability problems) F. Moller (decidability and complexity theory), M. Otto (finite model theory). The department maintains strong links with the logic groups in Munich (Schwichtenberg), Uppsala/Stockholm (P. Martin-Löf, V. Stoltenberg-Hansen, E. Palmgren), Amsterdam (J. Bergstra, honorary visiting professor at Swansea), and McMaster University, Canada (J. Zucker, honorary research fellow at Swansea).

## Part 2 Description of the Proposed Research and its Context

### 2.1 Background and Outline of the Project

It is a central aim of Computer Science to develop programming languages for writing efficient and correct programs. Most programming languages use types in order to obtain this goal. Types restrict the kind of data used by a program for input, output and for intermediate data. It is often possible to predict the size of the data stored in a data type, which allows more efficient memory management. Typed languages prevent the programmer from making mistakes, which result in data, which does not match the type, especially when combining several programs. Further, typed programming languages make it explicit what the programmer had in mind when defining data structures, which results in more transparent programs.

In simple type theory, types do not depend on other types or elements of other types. Many programming languages have added polymorphism, which means that types can depend on other types (eg. templates/generics in object oriented languages, modules, classes and functors in functional programming languages). In dependent type theory (DTT), types might depend on both other types and their elements. In non-dependently typed languages, the type of matrix multiplication can only express that it is a function taking two matrices and returning a third one – that the dimensions are in accordance, has to be checked at runtime. In DTT, we can assign to this operation the type, which takes the three dimensions involved, two matrices of appropriate dimensions and returns a third matrix of appropriate dimension. Another example is the format string operation (printf) in C, which cannot be introduced in Java, since the arguments have mixed types. In DTT, it can be typed [Au98]. Weak forms of dependent types occur in standard languages (eg. templates in C++, generics in Ada or functors in ML), but they are too restrictive to be applicable to the above examples, do not check conformance of parameters (Ada), or only allow instantiation by constants (C++). Recently, a new direction in software engineering called generative programming has been established, with the goal of automatizing the production of software ([CE00]). Apart from working on new language paradigms, research on generative types focuses especially on the template mechanism of C++, which has a static and therefore restrictive type system.

It is due to Augustsson (Chalmers, Sweden, [Au98]), that there exists now an efficient functional programming language, called Cayenne, based on full DTT. Independently, H. Xi has developed two dependently typed programming languages: dependent ML ([Xi01]) and Xanadu ([Xi99, Xi99]). Xi however restricts dependencies to natural numbers. Especially Augustsson's implementation has stimulated a series of workshops on "Dependent Types in Programming"<sup>1</sup>.

Traditionally, the main application of dependent types has not been in the area of programming but in program verification and the development of correct programs. In DTT, a predicate is nothing but a type, which is inhabited, if the predicate is true, and otherwise empty. The property, that a list is sorted, can be expressed as a type depending on that list, and a sorting function can be typed as a function taking two lists and returning a list with this property – the property "sorted" is verified at compile time. Similarly, one can express for instance that a railway interlocking system is safe. Any property, which can be expressed by a formula, can be written as a dependent type. Several type checkers and theorem provers based on DTT have already been developed: the Alf family of theorem provers (Alf, Half, Agda<sup>2</sup>, Alfa)<sup>3</sup> developed in Gothenburg – Agda will be mainly used in this project –, LEGO<sup>4</sup> from Edinburgh, Coq<sup>5</sup> from INRIA, France and Nuprl<sup>6</sup> developed at Cornell.

Type theoretic research is stimulated by TYPES, a project financed as a working group under the ESPRIT program of the European Community. Within this project a variety of topics in type theory are investigated, including meta theory (proof theory, models, etc), implementation issues

---

<sup>1</sup><http://www.dagstuhl.de/01341/>

<sup>2</sup><http://www.cs.chalmers.se/~catarina/agda/>

<sup>3</sup><http://www.cs.chalmers.se/Cs/Research/Logic/implementation.mhtml>

<sup>4</sup><http://www.dcs.ed.ac.uk/home/lego/>

<sup>5</sup><http://pauillac.inria.fr/coq/>

<sup>6</sup><http://www.cs.cornell.edu/Info/Projects/NuPrl/nuprl.html>

(proof assistants, automatic theorem proving), and applications (in education, in formal methods in industry, for formalising mathematics). Dependent types have found applications in linguistics and automated translation of natural languages [Ra95].

In his research, AS has specialized in applying proof theory to DTT and used this knowledge, in order to define various extensions of type theory. In this project, he wants to consider both extensions, which cannot be simulated in the original theory, since the underlying theory is otherwise too weak, and those, which could in principal be simulated, but this simulation requires a lot of coding and can therefore not be used when working with type checkers.

## 2.2 Programme and Methodology

### 2.2.1. Aims.

**1. Introducing Interactive Programs into Dependent Type Theory (DTT).** To introduce a theory of interactive programs and coalgebraic concepts into DTT and to explore the particular contributions of *dependent* types to it.

**2. Object-Oriented Programming in DTT.** To enrich DTT by concepts from object-oriented programming.

**3. Closed Formalization of Inductive-Recursive Definitions and Applications to Generic Programming.** To understand the full power of inductive-recursive definitions and to discover new applications in programming, especially generic programming.

**4. The  $\Pi_3$ -reflecting Universe and Extensions.** To determine the theoretical limits of current formulations of Martin-Löf's type theory and to develop concepts, which go beyond these limits.

### 2.2.2. Measurable Objectives of the Project.

**1. Interactive Programs into Dependent Type Theory (DTT).** To study an existing implementation of interactive programs, expand its interface and enhance its speed. To develop the concept of state-dependent interactive programs in DTT. To introduce rules for final coalgebras in DTT. To determine rules for introducing multi-threaded interactive programs. To carry out case studies of specifications of interactive programs.

**2. Object-oriented Programming in DTT.** To formulate, what an object is in DTT and how to construct programs using this approach. To carry out this formulation using a theorem prover.

**3. Closed Formalization of Inductive-Recursive Definitions.** To study existing applications of inductive-recursive definitions and find at least one new one. To carry out case studies of application of the new data types in the area of generic programming.

**4. The  $\Pi_3$ -reflecting Universe and Extensions.** To determine the extract strength of the existing  $\Pi_3$ -reflecting universe. To study extensions. If possible, to find an extension of the concept of inductive-recursive definitions, which subsumes the concept of a Mahlo universe.

**2.2.3. Methodology.** We will develop rules for various extensions of type theory and introduce models, especially PER-models. Soundness and correctness results will be established. We will type check the rules on a theorem prover, provided there are no principal restrictions imposed by the theorem prover. We will study the source code of theorem provers, especially of Agda, and their type theoretic languages. Further, we will carry out case studies of applications on the machine, using mainly Agda. Minor case studies will be carried out in the form of student projects, and studied in a 4th year module on critical systems. In order to determine the proof theoretic strength of various calculi, we will carry out well-ordering proofs using the technique of distinguished sets and upper bounds by modelling the type theories in suitable extensions of Kripke-Platek set theory.

**2.2.4. Timeliness and Novelty.** Dependent type theory as a programming language is a new research area in dependent type theory, and a new conference series "Dependent Types in Programming"<sup>7</sup> (2 conferences, 1999, 2001) has been established. The research in this proposal aims at solving theoretical problems, which have to be solved in order to create industrially applicable programming languages. There is fast growing interest in the use of generic and generative programming. A lot of research in this area is focused on the static type system of C++, which

<sup>7</sup><http://www.md.chalmers.se/Cs/Research/Semantics/APPSEM/dtp99.html>, <http://www.dagstuhl.de/01341/>

limits its applications. Dependent type theory (DTT) does not have these restrictions, however concepts of interactive programs and of object orientation in DTT are not well enough understood yet, in order to make use of DTT as a basis for an industrially applicable programming languages. This project has the objective of filling this gap. The closed formalization of inductive-recursive definitions developed by AS and P. Dybjer provides a powerful data type, which can be applied to generic programming, but these applications need to be exploited. To develop strong extensions of DTT will increase our theoretical understanding of it and we expect to find new data types, which are substantially different from existing ones, when developing such extensions. The area of theories of strength  $\Pi_3$ -reflection and beyond is currently an intensive area of research in proof theory. Topic IV will both benefit from and contribute to this intensive area using some novel ideas, and we hope that the exploration of this data structure will result in powerful applications.

### 2.2.5. Programme and Milestones.

**I. Introducing Interactive Programs into Dependent Type Theory (DTT).** Programs in ordinary DTT are non-interactive: they have only one input and compute one output. In [6], we have in collaboration with P. Hancock developed concepts for formalizing truly interactive programs in DTT, i.e. programs, which allow possibly infinitely many interactions between the user and the program, like editors, graphical user interfaces. We have discovered, that in the presence of dependent types, new constructs, which are not available when using non-dependent types, can be introduced: while loops and a refinement construct, which allows to translate a program written in a high-level interaction language into a lower level ones in a generic way and therefore allows to build up libraries. We want to explore these extensions further:

**I.a. Implementation and Case Studies.** Pierre Hyvernat has, under supervision of T. Coquand and in collaboration with AS, implemented in Gothenburg, Sweden, interactive programs in DTT [Hy01]. However, the current version of his implementation is rather slow. In this project, we want to expand this interface in order to cover more powerful interactive commands (currently only interaction with standard input/output is possible) and try to improve its speed. Further, some case studies will be carried out using more realistic examples.

**I.b. State-dependent IO.** Because of the richness of DTT, it is possible to introduce the concept of state-dependent IO into type theory. This means that the interactive commands a program can use depend on previous commands and responses. On one hand one can by executing commands increase or decrease directly the number of commands possible: For instance, after switching the printer on, one can print, after opening a window, one can write to this window and after closing a window, this is no longer possible. On the other hand, from responses to commands one obtains additional knowledge: For instance, after a successful check that the printer is switched on one can use the printer. The main task is to work out the precise type theoretic formulation in detail. Further, we want to explore the difference between commands, which directly change the physical world, and those, by which our knowledge about the physical world is increased. We want to explore looping constructs and variants of redirect in the context of state-dependent IO. Some case studies will be carried out.

**I.c. A General Concept of Final Coalgebras in Dependent Type Theory.** State dependent interactive programs are examples of elements of final coalgebras, or largest fixed points of functors, whereas the usual algebraic data types available in type theory formalize least fixed points. The type theoretic concepts considered above formalise restricted forms of such final coalgebras. We want to generalise these concepts in order to capture fixed points of general strictly positive functors. Extending ideas from I.b, we want as well to investigate state dependent final coalgebras such as dependent streams.

**I.d. Specification of Interactive Programs.** Using the concept of state dependent interactive programs, safety properties can be specified. It is more difficult to express purely type theoretically (i.e. without using a lot of coding) liveness properties. In this project, we want to investigate various kinds of properties for interactive programs in such a way, that they can easily be used by proof assistants. This section is particularly suitable for small case studies in the form of student projects.

**I.e. Multi-Threaded Interactive Programs.** Any realistic real-world interactive program makes use of multi-threading. First, we want to develop the concept of multi-threaded interactive programs in the context of type theory. Then we want to look at how to implement this and carry out some case studies for such programs.

**II. Object-Oriented Programming in DTT.** This research is stimulated by a question of Per Martin-Löf in a talk given by AS at TYPES 2002 on how to interpret concepts of functional programming languages involving simple types in Java. He asked: “Can we do the other way round”. Objects can be seen as interactive programs: They have an internal state, receive messages, and depending on these messages send responses and change their internal state. This is of course only a first idea, which covers only a small subset of the concepts of object oriented programming. What needs to be explored is how to combine objects to one program, and how to formulate overriding and polymorphism in this context. We want to explore this, and look at some examples of how to construct programs starting from simple objects. We expect that, when using DTT, new constructs, which cannot be expressed in non-dependent type theory, will arise. All formulations should be implemented in the theorem prover Agda.

**II.a. Component Based Software.** We want to explore applications of the concept of state-dependent interactive programs in order to better express contracts between implementers and clients. We hope that this allows to specify components, whose signature depends dynamically on the users.

**III. A Closed Formalization of Inductive-Recursive Definitions and its Applications in Generic Programming.** This topic will be carried out in collaboration with P. Dybjer, Gothenburg, Sweden. Inductive-recursive definitions generalises the concepts of data types available in ordinary DTT, including universes. The notions of function types and algebraic types are already contained in that notion. With P. Dybjer we have developed a closed formalization of inductive-recursive definitions, and we obtained a data type of codes for all inductive-recursive definitions. This allows to define functions, which take codes of such data types and compute new ones out of them, which is a very general form of generic programming. We want to explore these ideas further in the following directions:

**III.a. Applications of Induction-Recursion.** The goal is to find applications of induction-recursion, which are relevant for running programs (most applications known are in the area of theorem proving). One example was given by A. Bove and V. Capretta [BC01], who have developed a way of formalizing fully recursive programs by using induction-recursion. Another example would be the reflection of the simple typed lambda calculus. We want to work out more examples.

**III.b. Use of Inductive-Recursive Definitions in Generic Programming.** In [2] we have suggested that the data type of inductive recursive definitions can be used in generic programming: one can write program, which analyse a data structure given to them and create from it another data structure. We have already looked at a simple example, namely the extension of a data type by one more constructor and the inclusion of the original data type in the new one. We want to look at more advanced transformations.

**IV. Proof Theoretically Strong Principles, esp. Universes.** Previously we have introduced the Mahlo universe, which was the first example of a universe, the strength of which goes beyond induction recursion. The closed formalisation of inductive-recursive definitions (III) was based on ideas found there.

**IV.a The  $\Pi_3$ -reflecting Universe and Extensions.** We have developed a preliminary formulation of a universe, which we hope reaches the proof theoretic strength of  $\Pi_3$ -reflection. This strength is the current limit of the proof theory as it is published currently. We plan to determine the exact strength of it and look at how to extend it in order to reach the strength, of which proof theoretic methods are available although not published yet. The first steps would be to look at  $\Pi_n$ -reflection and a transfinite extensions of it, which should allow to reach the strength of KP plus the existence of an  $\alpha$ , which is  $\alpha + \rho$ -stable for some fixed  $\rho$ . Then we want to explore  $\Pi_1^1$ -reflecting universes.

**IV.b. Mahlo-Inductive Recursive Definitions.** We hope that, when moving to strong uni-

verses we find ideas of how to generalize inductive-recursive definitions and to include the Mahlo universe construction.

**2.2.6. Relevance to Beneficiaries.** The main beneficiaries will be researchers working in the area of dependent type theory. This area is nationally and internationally investigated in a large scale, to name only some places and names: Durham (Luo, Pollack, McKinna), Edinburgh (Burstall), Manchester (Aczel), Nottingham (Altenkirch), Helsinki (van Plato, Negri), INRIA (France; Dowek), Paris (Paulin-Mohring), Darmstadt (Germany; Streicher), Munich (Nipkow, Hofmann), Gothenburg (Coquand, Dybjer, Nordström, Smith), Uppsala (Palmgren, Stoltenberg-Hansen), Stockholm (Martin-Löf), Oslo (Normann), Padova (Sambin, Italy), Cornell (Constable, Kreitz), Carnegie Mellon (Pfenning), Cincinnati (Xi, USA). Of this large group, researchers working on programming with dependent types as represented by the workshops on programming with dependent types will particularly benefit from this project. Part IV will as well be of importance for researchers in proof theory of impredicative theories (Jäger and Strahm in Berne, Switzerland, Pohlers in Münster, Germany, Buchholz in Munich, Feferman in Stanford, Avigad in Pittsburgh and Rathjen at Ohio State University). We expect that this project will especially intensify the collaboration between the universities of Gothenburg, Uppsala, Munich, Leeds and Swansea.

The project has the objective to carry out fundamental research on new program paradigms, which subsume object-oriented and functional programming and extend it by allowing a much richer type structure. This will allow to develop more reliable programs, the correctness of which is verified at compile time and contribute to the development of more generative programs. Therefore, this research will contribute to a higher productivity and quality in the software industry.

Part of this research, especially case studies, will be of use in a fourth year course on safety critical systems. This will help to increase the appreciation of formal methods in teaching at the University of Swansea.

**2.2.7. Dissemination and Exploitation.** The aspired results of this project are to be presented first at international conferences like *Computer Science Logic (CSL)*, *Typed Lambda Calculi and Applications (TLCA)*, the meetings of the *TYPES* working group, the workshops on programming with dependent types, the *Logic Colloquium*, the workshops on generic programming, the meetings in mathematical logic at Oberwolfach, the workshops on proof theory – in all the above results have been presented previously – and at other conferences like *Logic in Computer Science (LICS)*. It is also anticipated to publish the results through refereed journals, especially *Journal of Theoretical Computer Science*, *Journal of Functional Programming*, *Annals of Pure and Applied Logic*, *Journal of Symbolic Logic*, *Archive of Mathematical Logic* and others. Preprints of the results and all software developed will be made available on the Internet.

**2.2.8. Industrial Collaboration.** We will collaborate on an informal basis with P. Hancock from Nexwave Solutions UK Ltd, Cambridge, who is applying ideas from dependent types in the area of distributed programs. This will provide us with invaluable insights into real world problems.

**2.2.9. Research Risk.** We are very optimistic that the main theoretical ideas will work out well (the outcome of IV, esp. IV.2, is more difficult to predict). The main risk of this project is, whether suitable applications can be found. A problem might be as well the dependency on theorem provers, which, when explored deeply, often reveal bugs, theoretical problems in their type theory or unacceptable runtime behaviour.

**2.2.10. Justification of Resources.** We expect to attract as RA an expert in dependent type theory, who will work mainly on the details of I - III and is able to carry out some implementations and case studies. His/her research will be supplemented by further PhD students (we supervise currently one), who are funded through other sources.

As pointed out already the project will be closely linked to the programming logic group in Gothenburg, the logic group in Uppsala, the LFCS in Edinburgh and the logic group in Munich. The funds requested for travel would enable these vital contacts to be maintained. They would also be used to support the presentation of the achievements of this project by AS and the RA at the

previously mentioned conferences, and enable one longer visit to Japan (T. Arai in Kobe, who is a leading expert in proof theory).

The equipment requested are two Linux PCs for AS and the RA, and additionally one Laptop. This equipment is needed in order to implement and present the system. The funds requested for technicians shall provide support for the equipment and installation, maintenance of software. Consumables include document production, software, and communication.

## References (see as well Part I)

- [Al01] A. Alexandrescu. *Modern C++ Design*. Addison-Wesley, 2001.
- [Au98] L. Augustsson. Cayenne — a language with dependent types. In *Proc. of the International Conference on Functional Programming (ICFP'98)*. ACM Press, September 1998.
- [BC01] A. Bove and V. Capretta. Nested general recursion and partiality in type theory. In: *Theorem proving in higher order logics: 43th international conference, TPHOLs 2001*, pp. 121–135, Springer 2001
- [CP90] T. Coquand and C. Paulin. Inductively defined types, preliminary version. In *LNCS 417, COLOG '88, International Conference on Computer Logic*. Springer, 1990.
- [CE00] K. Czarnecki; U. Eisenecker. *Generative programming*. Addison-Wesley, 2000.
- [Dy91] P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.
- [Dy94] P. Dybjer. Inductive families. *Formal Aspects of Computing*, 6:440–465, 1994.
- [Dy00] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2), June 2000.
- [Hy01] P. Hyvernat. Interactive programs in (pure) Martin-Löf type theory. Examensarbetet, Gothenburg, 20012.
- [Lu94] Z. Luo. *Computation and Reasoning*. Clarendon Press, 1994.
- [ML84] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
- [ML98] P. Martin-Löf. An intuitionistic theory of types. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory*. Oxford University Press, 1998.
- [Me91] P. F. Mendler. Predicative type universes and primitive recursion. In *Proceedings Sixth Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1991.
- [Mo91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
- [NPS90] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: an Introduction*. Oxford University Press, 1990.
- [Pa91] E. Palmgren. *On Fixed Point Operators, Inductive Definitions and Universes in Martin-Löf's Type Theory*. PhD thesis, Uppsala University, 1991.
- [PM93] C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties In *Typed lambda calculi and applications*, volume 664 of *Lecture Notes in Computer Science*, pages 328–245. Springer, 1993.
- [Ra95] A. Ranta. *Type-theoretical Grammar*. Clarendon Press, 1995.
- [Wa92] P. Wadler. The essence of functional programming. In *19'th Symposium on Principles of Programming Languages, Albuquerque*, volume 19. ACM Press, January 1992.
- [Wa95] P. Wadler. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*, volume 925 of *LNCS*. Springer, 1995.
- [Xi99] H. Xi. Facilitating Program Verification with Dependent Types. Preprint, 1999.
- [Xi99] H. Xi. Imperative Programming with Dependent Types. Preprint, 1999.
- [Xi01] H. Xi. Dependent Types for Program Termination Verification. LICS'01, 2001.