

Six Month Report

Mark Andrew Price
University of Bath

2nd December 2005

Contents

1	Introduction	2
2	Courses	2
2.1	Formal Methods and Programming	3
2.2	Programming I	4
2.3	Learning in Laboratories, Tutorials and Problem Classes	4
2.4	L ^A T _E X Course	5
3	Tutoring	5
4	Seminars	6
5	Conferences	6
6	Publications	6
7	Background Work and Literature Review	7
7.1	Logical Frameworks	7
7.2	Categorical Logic	8
7.3	λ -Calculus	9
7.4	Natural Deduction	10
7.5	Kripke Semantics	10
7.6	Philosophical Basis	10

8	Area of Research	11
8.1	Logical Framework	12
8.2	Representation	14
8.3	Semantics	15
9	Research Questions	15
10	Research Timetable	17

1 Introduction

This report is intended to detail what I have been doing over the last six months as well as mapping out the research I intend to carry out. There is also a timetable for detailing when the various stages of the research will be completed.

Section 2 looks at the university courses which I have attended during my time at Bath. These are two courses in programming, a \LaTeX course and a course designed to teach me how to run tutorials.

The next section, 3, simply details the lecture courses I have run tutorials for.

Section 4 mentions which seminar series I attend on a regular basis.

In section 5 the conferences and summers schools that I have attended are listed.

Section 6 contains details on a yet unpublished note I have written with my MSc supervisor.

A presentation of the background material I have studied over the past six months along with a brief literature review is contained in section 7.

Section 8 explains in more detail the area of my research along with some motivation for the research to be carried out.

Penultimately, section 9 poses some of the questions I would like to answer through my research with a timetable for this research being put forward in section 10.

2 Courses

As well as studying for my PhD, it is recommended that I attend some courses to improve my transferable skills. Below is the list of courses that I

have attended since arriving at Bath:

- Formal Methods and Programming
- Programming I
- Learning in Laboratories, Tutorials and Problem Classes
- L^AT_EX Course.

The first two were intended to improve my programming skills. The third course was mandatory and the final course is designed to improve my L^AT_EX skills.

2.1 Formal Methods and Programming

Formal methods and programming is an MSc level introduction to programming and the ideas behind the major ways of constructing programs. The main language used was Oberon-2 but since the course was mainly concerned with how to produce different styles of programs, the skills learned were transferable.

The main ways to design a program are modular and object-oriented. A modular approach involves dividing the problem into smaller sub-problems and then individually solving each sub-problem. The solutions are then put together in the final program with a module being used to solve each problem when it is encountered.

The following diagram shows an example of a modular program.

Here a maths module will solve the required mathematical problems and a database module will carry out all the database manipulation.

The other design approach is an object-oriented approach. This involves doing a data abstraction and a procedural abstraction.

Below is the information required to obtain a data abstraction:

- a set of data objects, which all have the same properties;
- a set of operations that can be applied to these objects.

We could have the set of integers which is just the type integer and here the set of operations would be the usual arithmetic operations, e.g. $+$, $-$, \times , etc.

A procedural abstraction is an algorithm which embodies some task or action we wish to be applied to our data abstraction.

The object-oriented approach means that the complexity of the problem is often reduced and the solution is more general so it can be re-used in a variety of contexts.

As well as learning the main ways to design and implement a program, I learned how to test and document simple programs, how to use simple formal methods for specification and verification and how to specify, implement and use standard data structures.

The result of which is that I gained a working knowledge of some of the theory behind the methodology of programming.

2.2 Programming I

The Programming I course is simply an introduction to Java for first year undergraduates. Java is an example of an object-oriented programming language. So this course linked well with formal methods and programming since it allowed me to become engaged in a formal language and solve some real world problems.

As well as learning the language and syntax of Java, problems were implemented allowing the abstract understanding of object-oriented problem solving to be cemented in my own mind.

2.3 Learning in Laboratories, Tutorials and Problem Classes

This was a two hour course designed to teach postgraduates how to successfully run tutorials and problem classes. Since I am running various tutorials,

this course was invaluable in explaining how to interact successfully with the undergraduates, keep them attentive and aid their understanding.

Various styles of undergraduate work were explored; for example, group work and individual work. These was taught alongside the different ways in which people learn.

The most and least effective ways of asking questions and drawing information out of the students were explained to us so that we could be good tutors to the students. We were also made aware of how we could often explain things in a way which was not useful to the students and how best to correct this.

The disability and discrimination act was explained to us, so that we would show tolerance, respect and understanding towards all the students who attended any session we were running.

2.4 \LaTeX Course

This course runs during the first semester of the 2005/2006 academic year, so it is still ongoing during the writing of this report.

It is being run by the mathematics department and is intended to cover all the \LaTeX students will require to produce their theses.

\LaTeX is a type setting language which is invaluable in producing any document which involves a large quantity of mathematics, such as this report and my thesis.

I already have a working knowledge of \LaTeX since I produced my masters dissertation in it but there are a few areas which are very sketchy and the course covered these. The most useful areas were inserting graphics and how to produce presentations in \LaTeX .

3 Tutoring

This semester I have run the following tutorials:

- CM10139 Computation I: Numbers and Structures
- CM10017 Systems I: Architecture and Operating Systems.

The first of these courses is the introductory mathematics course for first year undergraduates. It teaches them about sets, Boolean logic, quantifiers, proof by induction, maps, sequences, etc.

Systems I teaches the undergraduates about the different types of computer architectures, data representation within a computer and Boolean algebra. The tutoring is split into two parts; the first being tutorials which last five weeks and the second being supervising students and marking their work while they build logic circuits.

4 Seminars

There are various seminars on offer within the computer science department. The main ones of interest are the logic seminar, neural networks seminar and the real algebraic geometry seminar. I attend these whenever they are held.

On occasion the Artificial Intelligence group and the departmental seminar have topics of interest. Here the topic of the seminar dictates the attendance.

5 Conferences

During my time as a PhD student at Bath I have been to a meeting of the British Logic Colloquium, the Computer Science Logic conference and the international summer school on applied semantics (APPSEM).

6 Publications

Since arriving at Bath, a short note was produced jointly with Harold Simmons entitled ‘A ‘new’ Abstraction Algorithm’ [PS]. This note has come out of the my dissertation. It is currently in the process of being reviewed for the Journal of Logic and Computing.

The note describes and illustrates an algorithm for simulating lambda abstraction that was first published by Grzegorzcyk [Grz64] in the 1960s. The note points out that the original version was incorrect and shows how the algorithm can be corrected. There is also a comparison of the algorithm to similar algorithms available in the literature.

On the 11th November 2004, I gave a seminar as part of the logic seminar series presenting the results in this note.

7 Background Work and Literature Review

To be able to begin research into a theory of representing logics within a logical framework a lot of material needed to be read and understood. The following is an overview of this material.

7.1 Logical Frameworks

The idea of a logical framework is key to my research. From an early stage it was suggested by my supervisor that I should learn about the Edinburgh Logical Framework (LF) and consider whether or not I wished to find a general theory of representing logics in LF.

In [HHP91] the type theory of LF is introduced along with showing how to encode both first order and second order logic into LF.

In [AHM91] the following logics are encoded into LF:

- Kleene's Three-Valued Logic
- First Order Logic with a Choice Operator
- Hilbert Style Modal Logics
- Natural Deduction Style S4
- The Classical Lambda Calculus
- Call-by-Value Lambda Calculus
- Linear Lambda Calculus
- Hoare Logic
- Two-Register Machine Hoare Logic.

These encodings start to show the power of LF, since any sensible logic can be encoded within it.

LF has been implemented as a proof checker in a variety of incarnations. The first implementation was as LEGO with the details in [Pol94]. This is unfortunately no longer maintained, but it is still available online with all the documentation.

Frank Pfenning has implemented LF in both ELF and TWELF, with TWELF being based upon ELF. He also has an unpublished survey paper [Pfe] which looks at the ideas behind LF from a programming perspective.

Amy Felty has developed a system called Canonical LF which can be embedded in LF, which she calls Full LF, and has the same expressive power as LF restricted to Canonical LF. Details of this system can be found in the appendix of [Fel91]. She then uses this system for the proofs since it is a ‘simpler’ system.

Simpson’s unpublished paper [Sim] is the first to consider seriously the semantics of the meta-logic. However he only considers the semantics for a logical framework which is a minimal implicational predicate logic with quantification over higher types. He does not look at the more general LF, that is the $\{\forall, \subset\}$ -fragment of minimal first order logic with proof objects.

Simpson considers representation within a logical framework by looking at how logical consequence is encoded into his logical framework. He notes that faithfulness, i.e. that something true in the meta-logic is true in the logic, established semantically is more intuitive than obtaining the same result proof theoretically. The reasoning behind this is that logical consequence is encoded via a meta-axiomization of its properties. Establishing the ‘truth’ of these meta-axioms should give us faithfulness.

Gardner has studied and analysed the semantics of the logic of LF in her PhD thesis which was turned into the following technical report [Gar92]. Here a notion of a categorical model is given, but it is very syntactical and is not about meaning/truth nor does it give a satisfaction, \models , relation.

7.2 Categorical Logic

Categorical Logic looks at the relationship between category theory and logic. Category theory is way of analysing mathematical structures through their morphisms while mathematical logic is the language used for formalising structures in terms of their constituent parts.

Girard’s book, Proofs and Types [Gir89], looks at λ -calculi and how they can be used to understand logic. It covers important ideas like the Curry-Howard isomorphism which leads to the propositions-as-types correspondence. It also talks about natural deduction and its relationship to the sequent calculus. It also provides an introduction to linear logic and coherent spaces.

MacLane’s book [Mac97] covers most of the required category theory

that is used in day to day research. It is a general introduction and is at the required level.

Two books which cover the relationship between type theories and categorical semantics are [Cro93] and [LS86]. Both books provide a useful introduction to some of the key areas within categorical logic and allowed me to build up a solid foundation in this area.

Cartmell [Car86] and Pitts [Pit00] both produce a categorical structure to interpret the basic framework of dependent types, with Cartmell's paper using the idea of indexed categories for the semantics. Cartmell calls these indexed categories contextual categories. Pitts suppresses this and 'hides' the indexing. A path between these two extremes is taken in my own research.

Pitts's handbook article covers more than just a categorical semantics for dependent types. It is a survey of the core of categorical logic. It explains the categorical semantics of many-sorted equational logic in a category with finite products, as well as the categorical semantics of some simply typed constructors. A term-model construction for equational logic is developed and the resulting theory-category correspondence is drawn out.

Both Hofmann [Hof96] and Dybjer [Dyb95] look at the internal type theory of dependent types. They use the idea of a setoid, and Dybjer represents Hofmann's setoid model of type theory within type theory.

In [See83], Seely makes the relationship between hyperdoctrines and logic precise by showing that hyperdoctrines correspond to first order intuitionistic theories with equality.

7.3 λ -Calculus

λ -calculus is the main tool for presenting the syntax of the logic being studied.

Two books provide the introduction to the more basic types of λ -calculi. They are [HS86] and [Bar81]. However, neither of them go as far as covering dependent types, ie the $\lambda\Pi$ -calculus. They do however show the main ideas behind λ -calculi and how to study them mathematically.

The main λ -calculus that I am interested in is the $\lambda\Pi$ -calculus. This will capture the behaviour of dependently typed systems. A good presentation of the $\lambda\Pi$ -calculus, or polymorphic λ -calculus is in Seely's paper, [See87], where as well as introducing the calculus, he also provides a categorical semantics for it.

In Barendregt's handbook article, [Bar92], he presents a cube of λ -calculi and looks at the relationship between them.

7.4 Natural Deduction

Natural deduction is a way of showing how proofs are derived within a system. The admissible laws of a system are often expressed in a natural deduction style.

Within my research, natural deduction will be treated in as general a way as possible. The system of natural deduction used will be Prawitz's [Pra65], who looks at natural deduction in full generality.

Vigano's work, [Vig00], shows us how to present general logics in a natural deduction style.

7.5 Kripke Semantics

Kripke semantics have their roots in the philosophical ideas of Saul Kripke. The idea was originally formulated to deal with modal and intuitionistic logic. In his original papers he does not put an ordering relation on the set of all possible worlds. His main papers are [Kri63b], [Kri63c] and [Kri63a]. The ideas are also presented in his book [Kri72].

Most of the mathematical content has already been taken from his work and the above are cited mainly for reference.

Mitchell and Moggi, [MM91], produce a Kripke semantics more in line with the semantics I hope to produce. It becomes clear in their paper and my own research that the logical structure is inseparable from the term model.

7.6 Philosophical Basis

The main logical language I will use is the intuitionistic logic of Martin-Löf. The main papers of Martin-Löf of interest to me are [ML96], [ML87], [ML98], [ML75b], [ML75a] and [ML71]. Martin-Löf has also written the following book [ML84].

The ideas of Martin-Löf and others have led to a school of philosophy within mathematics called constructivism. In this school, only constructive proofs are considered to have meaning. This idea can be taken as a philosophical basis for mathematics. However, there are areas of mathematics where constructive proofs have yet to be found, and so there is no mechanism for talking about these results within constructivism.

However these philosophical ideas, while interesting, do not concern me since the main idea in my research is to treat constructive logic as a mathe-

mathematical object to be studied, rather than as a philosophical basis for mathematics. It is, however, important to be aware of where these ideas come from and their implications.

In view of this, we want to be able to work with tools and techniques within this field. Martin-Löf's ideas have been put to use within programming in [SNP90]. The idea of a Frege structure also exists within this universe, and Aczel's paper [Acz80], covers most of the information required.

8 Area of Research

In the late 1980s and early 1990s the computer science department at Edinburgh formulated the Logical Framework (LF) which provides a means to define (or present) logics [HHP91]. It is based on a general treatment of syntax, rules and proofs, by means of a dependently typed λ -calculus.

Here the syntax is treated in a similar way to Martin-Löf's system of arities [ML75b]. However, the approach taken is actually more general than Martin-Löf's system. The treatment of rules and proofs focuses on his notion of a judgement and introduces a new idea, that of judgements-as-types. The judgement-as-types principle means that each judgement is identified with the type of its proof.

So far no suitable categorical semantics has been found for LF. Gardner [Gar92] has developed a categorical model, but it is very syntactical and does not have a notion of truth/meaning. Simpson [Sim] has looked at developing a semantics for the meta-logic, but has only developed one for a logical framework, that is a minimal implicational predicate logic with quantification over higher types.

LF is a presentation of the $\{\forall, \subset\}$ -fragment of minimal first-order predicate logic with proof objects, so we need to find a similar semantics for LF based on the method employed in his paper.

The work of Mitchell and Moggi [MM91] defines Kripke-style models for the typed λ -calculus. This style of understanding Kripke models will be used as the main motivation for the Kripke-Beth-Joyal style models developed within my research.

8.1 Logical Framework

LF is a formal meta-logic which allows one to describe a logic in a manner which is suitable for mechanical implementation. To describe a logic within LF one must have the following:

1. A characterisation of the class of object-logics to be presented;
2. A meta-logic or language, together with its meta-logical status *vis-à-vis* the class of object-logics;
3. A characterisation of the representation mechanism for object logics.

These can be summarised by the following slogan:

$$\textit{Framework} = \textit{Language} + \textit{Representation}$$

These components are not entirely independent of each other.

A lot of study has been done on the effect of the language on LF, for example [HHP91] and [AHM91]. However, so far no one has taken seriously the issue of representation. I intend to study the effects that seriously considering the representation has on the framework, and the implications of altering the representation.

Since we need both a language and representation, a fixed language will be chosen while the effects of changing the representation are considered. For the purposes of this research, the language chosen will be that of the dependently typed λ -calculus, i.e. the $\lambda\Pi$ -calculus.

The choice of language should be viewed in light of Barendregt's λ -cube, [Bar81], where we see that LF with the $\lambda\Pi$ -calculus is equivalent to his system λP , that is the weakest of the available languages which still allows us to talk about dependent types. We also note that we have a signature within our λ -calculus, since we are taking the representation seriously and need to distinguish those 'objects' which are assigned to constants and are involved in encoding the logic's consequence relation.

With the λ -cube in mind, we note that we could have considered one of the other systems presented there; for example the calculus of constructions [CH88] and the system F of Girard [Gir72]. Since the choice of language and representation are not entirely independent of each other, we need to know what effect our choice of language has on our understanding of the various representations.

The judgements-as-types principle has its origins in Martin-Löf's [ML96] development of Kant's notion of a judgement [Kan00]. The judgement-as-types principle is that judgements are represented by the types of their proofs. This principle can be regarded as the meta-theoretic analogue of the well known propositions-as-types correspondence.

Essentially we assign a type to each judgement and proofs are represented as terms whose type is the representation of the judgement that they prove. LF has a structure which provides for the uniform extension of the basic judgement forms to two higher-order forms introduced by Martin-Löf, the hypothetical representing consequence, and the schematic representing generality. Using these forms, it is possible to see inference rules as primitive proofs of higher-order judgements. Thus, the notions of rule and proof can be collapsed into one and the distinction between primitive and derived rules of inference is eliminated.

We can make the judgement-as-types principle more precise by considering the indexed epimorphism between suitable Kripke-style models induced by the judgements-as-types correspondence. This indexed epimorphism will be referred to as the judgement-as-types epimorphism.

Within LF, a logical system is represented by a signature which assigns kinds and types to a finite set of constants that represent its syntax, judgements and rule schemes. An object-logic's rule and proofs can be seen as proofs of hypothetico-general judgements. There are representation theorems which relate consequence within the object-logic, $\vdash_{\mathcal{L}}$ to consequence within the encoded logic, $\vdash_{\Sigma_{\mathcal{L}}}$. We show the general idea behind an encoding:

$$\begin{array}{ccc} (X) & \Delta \vdash_{\mathcal{L}}^{\Phi} \psi & \textit{object-consequence} \\ & \Downarrow & \textit{encoding} \\ & \Gamma_X, \Gamma_{\Delta} \vdash_{\Sigma_{\mathcal{L}}} M_{\Phi} : j(\psi) & \textit{meta-consequence} \end{array}$$

where Δ is a set of judgements, $J_1(\psi_1), \dots, J_m(\psi_m)$; X is the set of variables that occur in ψ_i, ψ ; Φ is a proof-object (e.g. a λ -term); Γ_X corresponds to X ; $\Gamma_{\Delta} = x_1 : J_1(\psi_1), \dots, x_m : J_m(\psi_m)$ where each x_i corresponds to a place-holder for the encoding of J_i and M_{Φ} is a meta-logic term corresponding to the encoding of Φ .

We note that Γ_{Δ} and Γ_X both contain variables with types assigned to them. So it is possible to combine these sets into one set Γ . However we wish to make the distinction between the typed variables which arise from the judgements and those which arise from the variables occurring in the

judgements. The reason for doing this is that we are taking the idea of representation in LF seriously so we need to know where the typed variables have come from.

A more technical reason for keeping the variables separate is that we wish to consider applications of this research to logic programming. Here we will need to have a distinguished set of variables which correspond to the Γ_X and know that we cannot substitute them. This is needed to enable us to set up the required lattice and fixed point operator so as to talk sensibly about a basis for logic programming.

8.2 Representation

Representation within LF involves representing the logical consequence relation of the logic and representing the syntax of the language. The logical consequence relation is encoded into (meta-)axioms of the meta-logic, while the syntax is represented by terms of the chosen language, in our case the $\lambda\Pi$ -calculus.

When we encode a logic into LF, we want to be able to say whether or not it is adequate. For an encoding to be adequate, we need it to be both full and faithful. An encoding is full if it does not introduce any additional entities, i.e. terms, formulas and proofs, and faithful if it encodes all entities uniquely.

Fullness often comes from very little work; the proof system for the encoded logic is examined and one shows that each proof in the logic can be mimicked in the meta-logic. Faithfulness usually requires a lot more work, at least using the proof theoretic method. Here one needs to show that any derivation within the framework is essentially the representation of a derivation within some proof system of the encoded logic.

To see the difficulties with obtaining a proof-theoretic account of faithfulness see [HHP91], where they obtain proofs for the faithfulness of the encoding of first order logic and higher order logic by analysis of normal forms for derivations in the framework. This often requires a lot of tedious, fiddly work.

Considering faithfulness semantically, we see that encoding logical consequence amounts to a meta-axiomatization of its properties. Faithfulness should then follow from the ‘truth’ of these meta-axioms.

As noted previously, there is currently no complete semantics for LF, so to make a mathematical analysis of representation we first need to establish

a semantics for LF. With a suitable semantics for LF, we can then focus on the faithfulness of representations of logics in LF.

8.3 Semantics

When considering the semantics of LF, it is natural to consider a Kripke-Beth-Joyal style semantics, since it is the most general framework we have for describing semantics of a logical system. LF is a general presentation of logics, so it will need a general semantics.

There are also two technical reasons for choosing a Kripke-style semantics. The first of these is that LF is the $\{\forall, \subset\}$ -fragment of minimal first-order logic with proof objects. The $\{\forall, \subset\}$ -fragment of minimal first-order logic already has a well-understood Kripke semantics, for example [Dal94], so since LF is a generalization of this fragment, a generalization of Kripke semantics should provide a suitable semantics. Secondly, we are considering the $\lambda\Pi$ -calculus as our language and we know from the work of Mitchell and Moggi [MM91], that models of λ -calculi are linked to Kripke-style models of their semantics.

9 Research Questions

The main idea behind this research is to take representation within LF seriously. In this context, seriously means to study the properties of encodings from the logic into the meta-logic in a general and systematic way. Currently all the known encodings have been done in an ad-hoc manner, and there is no systematic way of achieving an encoding.

The first thing which has to be looked at is what the general logics we wish to encode in LF are and how they are presented. A good starting point is the labelled natural deduction systems of Viganò, [Vig00], since he has developed a framework for systematically studying non-classical logics in a uniform and modular way. We then ask how we can encode a system of this style into LF in such a way that we obtain faithfulness and fullness results. It seems likely that some conditions will need to be added to Viganò's systems to obtain a sensible encoding.

Simpson's discussion of how best to achieve faithfulness results for an encoding into a logical framework, [Sim], tells us that the easiest way to obtain the result is from a semantical analysis of the meta-logic. Unfortunately there is currently no suitable semantics for LF available. This is something

which will also need to be developed, and will be an important (technical) contribution of the research.

From the discussion in Section 8.3, we see that it will be necessary to develop a Kripke-style semantics for LF. To this end, a Kripke-style semantics of the $\lambda\Pi$ -calculus will be presented. The semantics of $\lambda\Pi$ are already available in the literature. An approach somewhere between that of Pitts [Pit00] and Cartmell [Car86] will be taken. So the indexed nature of the interpreting categories will be brought more to the fore than in [Pit00], but will not be as detailed as in [Car86].

Mirroring this style of presentation, a similar semantics for LF will be developed. This will then allow us to obtain the necessary faithfulness results for our encodings. We note that since we have chosen $\lambda\Pi$ as our language for representation in LF, the semantics developed for LF should be closely related to the semantics for $\lambda\Pi$.

Since there are already many encodings of logics into LF, the next step will be to see how these fit into the general framework. Firstly, can we represent the logics in a labelled natural deduction style, and what conditions arise in doing this? The next question is, what is the relationship between the general encoding and the encodings already obtained on an ad-hoc basis?

The effect of the propositions-as-types correspondence needs to be studied, since in general we obtain a judgements-as-types epimorphism between models, rather than an isomorphism between models. The conditions which give rise to an isomorphism from the epimorphism also need to be studied.

The result of the above research should be a theory for the representation of logics which can be presented in a labelled natural deduction form within LF.

Following on from the above research, it is possible to consider the applications to logic programming. It is possible to develop a systematic approach to logic programming in the style of PROLOG using LF. Essentially, we define (Horne) clauses for the meta-logic and develop the necessary fixed point operator for the underlying lattice.

Since we have shown how to encode a logic into LF in a general way, we can encode a logic into the above framework for logic programming and so be able to write logic programs for any logic which can be encoded into LF.

10 Research Timetable

The first six months of my PhD have been spent doing some of the ground-work required to start researching into representation within LF. Most of the material covered has been detailed in section 7.

By the end of February 2006, a suitable presentation of a Kripke-style semantics of the $\lambda\Pi$ -calculus should be completed and fully written up.

Alongside this, an understanding of labelled natural deduction systems and how to encode them into LF will be developed. This material should be well understood by June/July 2006, i.e. about the time of the twelve month report.

Most of the work on a Kripke-style semantics for LF should also be completed around the time of the twelve month report.

The second and third year will be spent developing a theory of representation within LF. This will involve looking in detail at the general encodings we will develop, seeing how they relate to existing encodings and asking questions about what it means to have an encoding for a logic and why it might not be possible to obtain one in certain circumstances.

During this period, the judgements-as-types epimorphism between Kripke models needs to be considered, as well as conditions within the logic which affect this epimorphism.

A considerable period of time during the third year will be spent turning the research into a thesis suitable for submission the university. This should be done by the end of the third year.

References

- [Acz80] Peter Aczel. Frege structures and the notions of proposition, truth and set. In H. J. Keisler J. Barwise and K. Kunen, editors, *The Kleene Symposium*, pages 31–50. North-Holland, 1980.
- [AHM91] Arnon Avron, Furio A. Honsell, and Ian Mason. Used typed lambda calculus to implement formal systems on a machine. Technical report, ECS, 91.
- [Bar81] Henk P. Barendregt. *The Lambda calculus its syntax and semantics*. North-Holland, 1981.

- [Bar92] Henk P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 117–309. Oxford University Press, 1992.
- [Car86] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.
- [CH88] Thierry Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.
- [Cro93] Roy L. Crole. *Categories for Types*. Cambridge University Press, 1993.
- [Dal94] Dirk Van Dalen. *Logic and Structure*. Springer, 1994.
- [Dyb95] Peter Dybjer. Internal type theory. In *BRA TYPES workshop*. Springer LNCS, 1995.
- [Fel91] Amy Felty. Encoding dependent types in an intuitionistic logic. In Grard Huet and Gordon D. Plotkin, editors, *Logical Frameworks*. Cambridge University Press, 1991.
- [Gar92] Philippa Gardner. Representing logics in type theory. Technical report, ECS, 92.
- [Gir72] Jean Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.
- [Gir89] Jean-Yves Girard. *Proofs and Types*. Cambridge University Press, 1989. Translated and with Appendixes by Yves Lafont and Paul Taylor.
- [Grz64] A. Grzegorzcyk. Recursive objects in all finite types. *Fund. maths.*, 54:73–93, 1964.
- [HHP91] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. Technical report, ECS, 91.

- [Hof96] Martin Hofmann. Syntax and semantics of dependent types. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*. Cambridge University Press, 1996.
- [HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ -Calculus*. Cambridge University Press, 1986.
- [Kan00] Immanuel Kant. *Immanuel Kants Logik: Ein Handbuch zu Vorlesungen*. Friedrich Nicolovius, Königsberg, 1800. In translation: R. S. Hartmann and W. Schwarz, Dover Publications, Inc., 1974.
- [Kri63a] Saul A. Kripke. Semantical analysis of intuitionistic logic I. In J. N. Crossley and M. A. E. Dummett, editors, *Formal Systems and Recursive Functions*. North-Holland, 1963.
- [Kri63b] Saul A. Kripke. Semantical analysis of modal logic I. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 16:83–94, 1963.
- [Kri63c] Saul A. Kripke. Semantical considerations on modal logics. *Acta Philosophica Fennica*, 9:83–94, 1963.
- [Kri72] Saul A. Kripke. *Naming and Necessity*. Blackwell, 1972.
- [LS86] J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge University Press, 1986.
- [Mac97] Saunders MacLane. *Categories for the Working Mathematician*. Springer, 1997.
- [ML71] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. E. Fenstad, editor, *Second Scandinavian Logic Symposium*. North-Holland, 1971.
- [ML75a] Per Martin-Löf. About models for intuitionistic type theories and the notion of definitional equality. In S. Kanger, editor, *Proceedings of the Third Scandinavian Logic Symposium*. North-Holland, 1975.
- [ML75b] Per Martin-Löf. An intuitionistic theory of types. In H. E. Rose and J. C. Shepherdson, editors, *Studies in Logic and Foundations of Mathematics*, volume 80, pages 73–118. North-Holland, 1975.

- [ML84] Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
- [ML87] Per Martin-Löf. Truth of a proposition, evidence of a judgement, validity of a proof. *Synthese*, 73:407–420, 1987.
- [ML96] Per Martin-Löf. On the meaning of the logical constants and the justification of logical laws. *Nordic Journal of Philosophical Logic*, 1, 1996.
- [ML98] Per Martin-Löf. An intuitionistic theory of types. In Giovanni Sambin and Jan Smith, editors, *Twenty-Five Years of Constructive Type Theory*. Oxford University Press, 1998.
- [MM91] John C. Mitchell and Eugenio Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51:99–124, 1991.
- [Pfe] Frank Pfenning. The practice of logical frameworks. ftp.cs.cmu.edu/user/fp/papers/caap96.ps.gz.
- [Pit00] Andrew M. Pitts. Categorical logic. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, chapter 2, pages 39–128. Oxford University Press, 2000.
- [Pol94] Randy Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994.
- [Pra65] Dag Prawitz. *Natural Deduction*. Almqvist and Wiksells, 1965.
- [PS] Mark A. Price and Harold Simmons. A ‘new’ abstraction algorithm. <http://www.cs.bath.ac.uk/map22/downloads/Grzalg.ps>.
- [See83] Robert A. G. Seely. Hyperdoctrines, natural deduction and the beck condition. *Z. Math. Logik Grundlagen Math*, 29:505–542, 1983.
- [See87] Roger A. G. Seely. Categorical semantics for higher-order polymorphic lambda calculus. *Symbolic Logic*, 52:969–989, 1987.

- [Sim] Alex Simpson. Kripke semantics for a logical framework. homepages.inf.edu.ac.uk/als/Research/kripke.ps.gz.
- [SNP90] Jan Smith, Bengt Nordström, and Kent Petersson. *Programming in Martin-Löf's Type Theory*. Oxford University Press, 1990.
- [Vig00] Luca Vigano. *Labelled Non-Classical Logics*. Kluwer Academic Press, 2000.