# Reasoning with Names and Identity in Programming Languages

Previous research track record

## Personnel

Ian Stark has an MA and Certificate of Advanced Study in Mathematics, a Diploma and PhD in Computer Science; all from the University of Cambridge. Following his doctorate, during 1995 he held a Marie Curie Fellowship at the University of Pisa, Italy; then worked for two years at the University of Aarhus, Denmark, as a member of BRICS, the institute for Basic Research in Computer Science. Since December 1997 he has been a lecturer in the Division of Informatics at the University of Edinburgh, a member of the Laboratory for Foundations of Computer Science (LFCS). He is currently Deputy Director of LFCS.

His research is in the mathematical semantics of programming languages: working with category theory, domain theory, models of concurrency, functional and object-oriented languages. His investigation of local names, originally in collaboration with Pitts, forms the basis for this proposal [33, 34, 38, 39, 40]. He has also published work on categorical models for concurrency, both independently and in collaboration with Winskel and Cattani; and is involved in the development of the MLj compiler of Benton, Kennedy and Russell [5].

He presently supervises one PhD student, Chothia, who is working on a calculus of names in distributed systems.

## Host organisation

The research proposed is to be carried out at the Laboratory for Foundations of Computer Science in the Division of Informatics at the University of Edinburgh.

The Division of Informatics draws together the previous departments of Artificial Intelligence, Computer Science, and Cognitive Science. Its remit is to study the structure, behaviour, and interactions of natural and artificial computational systems. As such it hosts a number of research institutes, of which the Laboratory is one.

Within the Division, LFCS provides an excellent environment for research in this area. It is a substantial internationally-known research establishment, with a broad community of researchers from across theoretical computer science. The Laboratory has many links to other groups in academia and industry, both national and international, and attracts numerous visitors. It currently has around 60 members: 30 teaching and research staff, 20 PhD students, plus assorted visitors and associate members.

This proposal draws on areas where LFCS has a strong past record and current presence. Logical frameworks were pioneered at Edinburgh, and there is ongoing work on the usability of mechanised proof (Burstall, Jackson, Plotkin, Aspinall). Work on names and binding at LFCS includes the EPSRC project ‘Structure of Programming Languages: Syntax and Semantics’, GR/M56333, and a number of local researchers are active in this field (Hofmann, Plotkin, Power, Turi), as well as in semantics generally (Abramsky, Fourman, Longley, Simpson, Wehr).

The Laboratory has always looked to the practical application of theory as an important part of its work, and this proposal follows that approach. Moreover, LFCS presently hosts the object-based MLj compiler, and is supporting the development of Poly/ML: access to the mechanics of these compilers will provide a useful resource for the work proposed.

LFCS provides a extremely rich and supportive environment for postgraduate training and research. The community of around 20 research students is closely involved in the life of the Laboratory, organising research clubs and social events. There is a strong international element, with many students from across Europe and overseas. The unique LFCS advanced course in theoretical computer science has been established for several years: all students attend during their first year. The Informatics Graduate school and the University offer further training in general research skills. Computing and support facilities are maintained to a high standard.

# Reasoning with Names and Identity in Programming Languages

Proposed research and its context

## 1 Background

There is a substantial body of work surrounding formal reasoning about programs written in functional languages — indeed, this is commonly stated as one of their advantages [22]. Such work encompasses a variety of different approaches: for example, *axiomatic* reasoning takes certain equalities as given, and derives further properties of programs; *operational* reasoning works with an explicit semantics of program evaluation; and *denotational* methods build an abstract mathematical model of program meaning. All are successful in deriving or verifying properties of purely functional programs. Some of this success comes from the absence of side-effects in such programs; part is due to the formal and exact specification of systems like the lambda-calculus.

This style of work need not however be limited to such chaste languages. Standard ML, for example, mixes higher-order functions with a range of non-functional features while retaining a precise and principled operational semantics [29]. The Java language, though imperative and object-oriented, has a strong type system and the beginnings of a firm semantics [17, 26, 23, 12]. It is appropriate then to look at transferring successful techniques from the purely functional setting into these broader areas.

The benefits from this flow in both directions. The powerful methods developed for pure languages can give practical results when applied to richer, multifaceted languages that lie closer to the real world. Conversely, the interaction between language features is interesting in itself, and sheds new light on existing models of computation.

One particular line of active research concerns the behaviour of functions with *local state*: privately generated storage cells whose identity persists over time. More broadly, this is an instance of *generativity*: the idea that an entity may be freshly created, distinct from all others. This is captured by the study of *names*, abstract individuals with *identity* but nothing else: all one can do is generate them, pass them around, and compare them with each other.

The combination of names and functions has been explored by the proposer in work on the *nu-calculus*, developed with Pitts [38, 33]. This has further developed into an analysis of functions with local state, where the central issue is whether variables held by functions remain private [34]. A parallel strand of research, originating with Reynolds, considers local state in Idealized Algol, and the question of interference between blocks of code [36, 37].

Names and generativity also arise in object-oriented programs, with the dynamic creation of new objects as instances of a class, and the issue of object identity. The question of functions that interfere (or collaborate) through private state is matched by objects and their instance variables. Moreover, object-oriented languages are starting to yield to formal semantics and analysis. One example is the comprehensive theory of objects proposed by Abadi and Cardelli [1]; there is also an active workshop community in this area [7, 9].

A useful test for reasoning about programming languages is *contextual equivalence*, whereby two expressions are equivalent if they can be freely exchanged in any program; there is no way in the language itself to distinguish them. A denotational model of a programming language is said to be *adequate* if it can be used to prove contextual equivalences; it is *fully abstract* if it can prove them all. As features are added to a language — names, recursion, objects — it becomes progressively harder to find denotational models that can capture the interaction between them. One alternative is to study the language more directly, through its operational semantics. Here too there is a contrast between correct methods, that imply contextual equivalence; and complete ones that describe it exactly.

There is a long history of operational methods for reasoning about state in LISP, notably those of Talcott and associates [27, 20]. For typed languages, the proposer and Pitts have developed a novel technique that completely characterises contextual equivalence for a calculus with higher-order functions, recursion and dynamically-generated integer storage cells [34]. The key innovation is a logical relation on continuations, defined in concert with a relation on terms.

These methods described above are correct, and sometimes complete, for reasoning about languages and programs. However, being usable is another matter. Anything beyond the barest calculus typically gives rise to a complex reasoning system, with awkward proof obligations and a need for rather detailed mathematical knowledge. Commonly, it is necessary to absorb the entire semantics of a programming language before anything can be proved.

From this aspect, the axiomatic approach wins out, with its straightforward algebraic manipulation. Unfortunately simple axioms are insufficient to capture the subtleties of generativity and higher-order behaviour; for these we must step from axioms to rules, appropriately chosen for the system at hand. Such rules can encapsulate delicate reasoning systems in a usable format.

In this vein, I have developed schemes for reasoning about the interaction between names and functions. These take two existing operational techniques, and extract from each a rule-based logic that allows simple and direct proofs of contextual equivalence [39]. The more powerful of the two embodies the method of logical relations, and can distinguish between private and public uses of names in higher-order functions.

Not only do such rule systems make operational techniques more palatable, they also open up a short-cut to automated reasoning about programs. Instead of encoding the entire semantics of a language in a theorem prover, it is enough to capture a rule system already proved to be correct. This is akin to the difference between "shallow" and "deep" embeddings of one logic in another.

## 2 Overall aims

The aims of the proposed research are to refine and extend some existing methods for reasoning about programming languages that combine first-class functions and local state. These extensions are in two complementary directions: towards applicability, through implementation in an automatic theorem prover; and towards other language paradigms, looking specifically at first-class objects with private variables.

## 3 Objectives

This research has three principal objectives.

1. To implement rule-based reasoning about names [39] in the Coq theorem prover [3].
2. To construct a scheme of rules for reasoning about state in functional languages, based on the operational methods of Pitts and Stark [34].
3. To develop operational methods for reasoning about local state in a simple object-oriented language, drawing on analogous work with functional languages [38, 34, 40].

## 4 Methodology

Each of the project objectives supports the others, with theoretical models leading to implemented reasoning methods. Individually, though, they require different methods and techniques.

### 4.1 Mechanised reasoning on names

The groundwork for this objective is the scheme of rules for reasoning about the nu-calculus previously described by the proposer [39]. That paper presents two approaches, a basic equational scheme and a more powerful relational one. Assertions in these take the following form:

$$s, \Gamma \vdash M_1 =_\sigma M_2 \qquad \text{and} \qquad s \vdash M_1\ R_\sigma\ M_2\ .$$

These state that terms $M_1$ and $M_2$ are equivalent, or related according to $R$, respectively. Annotations $s$, $\Gamma$ and $\sigma$ describe the names, variables, and types involved; $R$ is a relation matching two sets of names. Such assertions are derived by applying a family of rules. Most are fairly simple, like the following congruence which shows that a branch in a conditional can always be replaced with an equivalent one.

$$\frac{s, \Gamma \vdash M_1 =_o M_2}{s, \Gamma \vdash \mathit{if}\ B\ \mathit{then}\ M_1\ \mathit{else}\ M' =_\sigma \mathit{if}\ B\ \mathit{then}\ M_2\ \mathit{else}\ M'}$$

One or two are more elaborate:

$$\frac{\begin{array}{rl} s, \Gamma \vdash M_1[n/x] =_\sigma M_2[n/x] & \text{each } n \in s \\ s \oplus \{n'\}, \Gamma \vdash M_1[n'/x] =_\sigma M_2[n'/x] & \text{some fresh } n' \end{array}}{s, \Gamma \oplus \{x : \nu\} \vdash M_1 =_\sigma M_2}\ .$$

This states that to show the equivalence of open terms $M_1$ and $M_2$, featuring a free name variable $x$, it is enough to show that they are equivalent when $x$ is replaced by any concrete name $n \in s$ and by one fresh name $n' \notin s$.

The logic is useful because all derived assertions correspond to true statements about contextual equivalence between terms. It is easier than reasoning directly with operational semantics, because the rules do all: there is no need to explicitly reduce terms, or test functions on arguments.

To mechanise this the proposal is to use a *logical framework*: these are flexible reasoning tools that can represent a variety of object logics [18, 32]. A number are freely available, with varying degrees of support: such as Coq [3], Isabelle [31], and Lego [24]. In this case the framework must be sufficient to represent the terms of the nu-calculus, together with the assertions and rules of the logic. All of these components are strongly typed, which a framework based on a sufficiently powerful type theory can use to advantage.

The initial choice for this part of the project is the Coq system developed at INRIA [3]. This is based around just such a type theory, and, as described below, it has already been successfully used for reasoning about names in the $\pi$-calculus. Coq has an accompanying graphical reasoning interface, Pcoq [6](previously CtCoq); both are actively maintained, and are expected to remain so during the life of the project. Coq is also supported by the Proof General tool for proof management developed at Edinburgh [35].

The trickiest part of placing the nu-calculus in a logical framework is the treatment of names and name binding. Variable binding has traditionally been handled with higher-order abstract syntax [11]; however, the fact that names have distinct identities makes their behaviour subtly different. To manage this requires a reexamination of binding, substitution and $\alpha$-conversion.

A number of recent and current research projects attempt exactly this. Work of Gordon and Melham [16], and Pollack and McKinna [28] has looked at an explicit treatment of binding in logical frameworks, using concrete representation of terms. Honsell, Miculan and Scagnetto have recently adapted higher-order abstract syntax to handle names in a Coq encoding of the $\pi$-calculus [21]. Purely mathematical models of binding are also under development: by Gabbay and Pitts [15], Fiore, Plotkin and Turi [14] and Hofmann [19]. Notably, Hofmann proves the soundness of the work done with Coq in [21].

To implement nu-calculus reasoning, this project proposes to draw especially from these recent mathematical models; and also to feed back new insights from applying them to a concrete problem.

### 4.2 Rules for reasoning on state

This objective starts with the operational reasoning developed by Pitts and the proposer in [34]. That describes a language *ReFS* featuring higher-order functions with recursion and locally-declared integer reference cells; a simple form of ML, in fact. In particular, ReFS functions can store information from one call to the next — as happens with memoisation and other techniques for dynamic programming. Two such functions can then present identical argument-result behaviour while using persistent private store in different ways.

To capture this, that paper defines a logical relation based on an explicit matching between memory states. In a significant novelty, this relation is defined on both expressions and contexts by mutual induction over types. The central result shows that this completely characterises contextual equivalence for this language. It also validates a particular concrete idiom for reasoning about functions with private store: the *Principle of Local Invariants* states that two expressions are equivalent if there exists some relation between their private states that is preserved on execution.

As it stands, the operational technique described requires a full knowledge of the reduction semantics for the language at hand. The proposal here is to develop a relational logic, similar to that existing for names in the nu-calculus, that will allow straightforward rule-based deduction. For expressions of ground type, assertions and relations would approximate those familiar from Hoare logic — "executing this code preserves that property". For higher types, the machinery of logical relations comes into play; for example, an assertion might be that two functions behave equivalently whenever supplied with arguments that respect some relation. Crucially, proving such assertions in the logic would not require any actual reduction of terms, or explicit consideration of all possible function arguments. Instead such information is embedded in the rule schemes.

The examples of [34, §5] provide several case studies for a relational logic, of varying sophistication. The simplest address representation independence: first-order functions that use local store in varying ways but with the same observed effect. More challenging examples involve higher-order memoisation and profiling operators, which modify other functions by attaching local state.

The operational work cited motivates two different styles of reasoning rules. A basic scheme would capture relations on terms, with the Principle of Local Invariants as the key reasoning tool. A more advanced scheme would also express relations between continuations, with the Principle of Local Invariants as a derived rule. This could in principle prove more equivalences between higher-order terms; a risk is that the logic may become unreasonably complex.

The proposed project is to develop the basic scheme, trialling it through the case studies mentioned, and investigate the practicality of the more advanced reasoning scheme.

### 4.3 Object identity and local state

The operational methods for ReFS mentioned above use techniques developed for names to reason about memory cells holding integers: intuitively, the names are the cell addresses. The third objective of the proposed work is to use these same techniques to examine the notion of identity in object-oriented languages.

In most such languages, objects not only have fields and methods, but also a concrete identity, usually represented by the heap address at which the object resides. Any two objects may be compared: if equal, then their fields and methods are certainly the same, but not conversely as they may also have equal parts and yet be distinct.

There are a number of formal models of object-oriented languages, often concentrating on type systems and inheritance; see the comparative survey of Bruce *et al.* [8]. Some of these models give an operational semantics with an attendant reasoning system. One approach is to take objects as simple terms, with the only storage being instance variables within them [1, 23, 30]. Unfortunately this precludes reasoning about object identity, which depends on the reverse model: objects stored within a heap, linked by pointers [2, 10].

The proposal is to describe a small object-oriented calculus based on this latter model, using name technology to formalise heap pointers and hence identity. This description would include evaluation rules, suitable notions of object equivalence, and examples of how to represent some object-oriented programming idioms.

The relational reasoning methods of ReFS can then be brought to bear on this language. The aim is to derive intuitive principles for reasoning about objects that use private instance variables. In particular, such principles must remain valid in the presence of methods that can pass around other objects, and may be re-entrant.

For case studies we can draw on the work of Abadi and Leino [2], who give a range of examples with their first-order Hoare logic for objects. These include some using local variables that their system cannot handle because of its global store. In discussing such problems they cite the proposer’s earlier work [33] as a possible remedy: this project takes up that lead.

One relevant technique is the style from [34] of defining a logical relation in tandem on both terms and continuations. Originally introduced to handle the interaction between state and recursion, this seems particularly stable under changes in language features; it takes account not just of what terms may do, but what others may do with them.

It is worth noting that in the setting of ML-like languages, the obvious next development for the ReFS work would be to allow storage of functions in cells, not just ground data. In its full generality, this is known to complicate things considerably; by taking a side-step into object-oriented languages, we are in fact making a smaller increment. This is because we can take advantage of the distinction between fields, which can be freely updated, and method bodies, which are typically more constrained and might even be fixed at object creation.

Existing logical systems for object calculi are mostly directed to proving Hoare-style assertions about program code. The intention in this project is to handle Morris-style equivalence: proving that two different program fragments can replace one another. This is the notion that lies at the heart of any program optimisation. Moreover, it can handle higher-order functions, and has a uniform applicability across different languages.

A further refinement is to investigate relational reasoning, as done in ReFS, to demonstrate equivalences between objects that use private variables in different ways.

## 5 Timeliness

The objectives listed are all set at a level well matched to other current research. Reasoning methods for functions and state are now sufficiently well-understood to allow distillation into workable rule-based schemes, and to support mechanical implementation. Theoretical work on names and binding is extremely active right now, and this project both uses that and aims to participate. Foundational approaches to object-oriented languages are also a busy topic; in the past much of this has concentrated on static semantics and sound type systems, but dynamic semantics is now coming to the fore. By addressing the issue of identity and pointers this project will add to present understanding of first-class object behaviour.

## 6 Relevance to beneficiaries

The work proposed can aid the design of systems for reasoning about full-scale programming languages, by providing a sound basis for annotation and assertion languages. The beneficiaries here are those who design and build tools for such reasoning, in both academic and industrial research groups. We do not intend to deliver such tools ourselves, but rather to investigate and demonstrate feasible reasoning systems to underly them. Regarding this we are in contact with Leino at Compaq SRC, developer of the Extended Static Checker for Java [25].

A second relevant area is the development of optimising compilers for high-level languages, that analyse effects and interference in order to write better code. This relies on consistent logical systems to describe effects, with suitable reasoning schemes to match. The work proposed here is again appropriate to underpin this, and so is of benefit to implementors of such optimising compilers. We have a specific current collaboration with Benton and Kennedy, the developers of the MLj compiler, now at Microsoft Research Cambridge [5, 4]. This is a compiler for Standard ML targeting the object-based Java Virtual Machine.

## 7 Dissemination of results

The output of this work will take the form of technical reports, articles, and, for the first objective, a specific implemented system. We shall present these at relevant conferences and workshops, as well as maintaining regular contact with a range of European and overseas research colleagues. We shall place all papers in electronic archives, and also make the software produced available on the world-wide web.

## 8 Justification of resources

**People** We request one EPSRC research studentship, to begin in October 2000 at the earliest. The student will work full time on the project, concentrating on objective 1, mechanised reasoning with names, as detailed in the plan of work. The principal investigator will contribute $1/2$ of his personal research time to this project, an average of eight hours per week, including supervision of the student.

**Equipment** We ask for portable computers for the principal investigator and the student. These are to a standard Linux configuration set up by our computing staff. Basic computing infrastructure will be provided by the Division of Informatics, for which it makes a fixed charge. There is also a research support charge associated with the studentship, paid to the Informatics Graduate School.

**Travel** An important part of the research and its dissemination is our continuing contact and collaboration with the beneficiaries named above and groups working in related areas, both in Europe and overseas. This includes the programming languages group at Microsoft Cambridge, together with researchers at Cambridge University, QMW, Birmingham, Sussex, Aarhus, Genoa, Pisa, Udine, INRIA sites in France and Compaq Systems Research Centre (formerly DEC SRC) in the US.

In support of this we are asking for funding for a number of trips within the UK, to Europe and overseas, for both the principal investigator and the research student. In addition, we seek costs for participating in relevant workshops and conferences, such as ETAPS, ICALP, POPL, LICS, HOOTS, ICFP, TLCA, FOOL, ECOOP and OOPSLA, as well as appropriate one-off events.

## References

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag Monographs in Computer Science, 1996.

[2] M. Abadi and K. R. M. Leino. A logic of object-oriented programs. Research Report 161, Digital Systems Research Centre, Palo Alto, California, September 1998.

[3] B. Barras *et al*. *The Coq Proof Assistant: Reference Manual*. Version 6.3.1, INRIA, December 1999.

[4] N. Benton and A. Kennedy. Monads, effects and transformations. In *HOOTS 99: Higher-Order Operational Techniques in Semantics*, ENTCS 26. Elsevier, 1999.

[5] P. N. Benton, A. Kennedy, and G. Russell. Compiling Standard ML to Java bytecodes. In *International Conference on Functional Programming: Proceedings of ICFP '98*. ACM Press, 1998.

[6] Y. Bertot *et al*. Pcoq: A graphical user-interface for Coq. `http://www-sop.inria.fr/lemme/pcoq/`.

[7] K. B. Bruce *et al*. FOOL: International workshops on foundations of object-oriented languages. `http://www.cs.williams.edu/~kim/FOOL/`.

[8] K. B. Bruce, L. Cardelli, and B. C. Pierce. Comparing object encodings. In *Theoretical Aspects of Computer Software: Proceedings of TACS '97*, pages 415–438, 1997.

[9] L. Cardelli, A. Jung, P. O'Hearn, and J. Palsberg. The semantic challenge of object-oriented programming. Seminar Report 216, Schloss Dagstuhl, June 28–July 3, 1998.

[10] F. S. de Boer. A WP-calculus for OO. In *Foundations of Software Science and Computation Structure: Proceedings of FoSSaCS '99*, LNCS 1578, pages 135–149, March 1999.

[11] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In *Typed Lambda Calculi and Applications: Proceedings of TLCA '95*, LNCS 902, pages 124–138. Springer-Verlag, 1995.

[12] S. Drossopoulou, S. Eisenbach, and S. Khurshid. Is the Java type system sound? *Theory and Practice of Object Systems*, 5(1):3–24, 1999.

[13] *Proceedings of the Fourteenth Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2–5, 1999.* IEEE Computer Society Press, 1999.

[14] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *LICS '99* [13], pages 193–202.

[15] M. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *LICS '99* [13], pages 214–224.

[16] A. D. Gordon and T. Melham. Five axioms of alpha-conversion. In *Theorem Proving in Higher Order Logics: Proceedings of TPHOLs '96*, LNCS 1125, pages 173–191. Springer-Verlag, 1996.

[17] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison Wesley, 1996.

[18] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

[19] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *LICS '99* [13], pages 204–213.

[20] F. Honsell, I. A. Mason, S. Smith, and C. Talcott. A variable typed logic of effects. *Information and Computation*, 119(1):55–90, May 1995.

[21] F. Honsell, M. Miculan, and I. Scagnetto. Pi-calculus in (co)inductive type theories. Technical report, Dipartimento di Matematica e Informatica, Università di Udine, September 1998.

[22] J. Hughes. Why functional programming matters. *The Computer Journal*, 32(2):98–107, April 1989.

[23] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. Reasoning about Java classes (preliminary report). In *Proceedings of OOPSLA '98*. ACM Press, 1998.

[24] The LEGO proof assistant. `http://www.lfcs.informatics.ed.ac.uk/lego/`.

[25] K. R. M. Leino *et al*. ESC/Java: the Extended Static Checker for Java. `http://research.compaq.com/SRC/esc/EscJava.html`.

[26] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison Wesley, second edition, April 1999. Online edition at `http://java.sun.com/docs/books/vmspec/`.

[27] I. A. Mason and C. Talcott. Inferring the equivalence of functional programs that mutate data. *Theoretical Computer Science*, 105:167–215, 1992.

[28] J. McKinna and R. Pollack. Some type theory and lambda calculus formalized. *Journal of Automated Reasoning*, 23(3–4):373–409, November 1999.

[29] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, May 1997.

[30] W. Naraschewski and M. Wenzel. Object-oriented verification based on record subtyping in higher-order logic. In *Theorem Proving in Higher Order Logics: Proceedings of TPHOLs '98*, LNCS 1479, pages 349–366. Springer-Verlag, September 1998.

[31] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. Springer-Verlag, 1994.

[32] F. Pfenning. The practice of logical frameworks. In *Trees in Algebra and Programming — CAAP '96*, LNCS 1059, pages 119–134. Springer-Verlag, 1996.

[33] A. M. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *Mathematical Foundations of Computer Science: Proceedings MFCS '93*, LNCS 711, pages 122–141. Springer-Verlag, 1993.

[34] A. M. Pitts and I. Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*, pages 227–273. Cambridge University Press, 1997.

[35] Proof General. `http://www.lfcs.informatics.ed.ac.uk/proofgen/`.

[36] J. C. Reynolds. The essence of Algol. In *Proceedings of the 1981 International Symposium on Algorithmic Languages*, pages 345–372. North Holland, 1981.

[37] J. C. Reynolds. Syntactic control of interference, part II. In *Automata, Languages and Programming: Proceedings of the 16th International Colloquium*, LNCS 372, pages 704–722. Springer-Verlag, 1989.

[38] I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, December 1994.

[39] I. Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33(4):369–396, April 1998.

[40] I. Stark. When is a local variable not a local variable? In *The Semantic Challenge of Object-Oriented Programming* [9]. Abstract of talk.

# Reasoning with Names and Identity in Programming Languages

Diagrammatic work plan

## Individual schedules

The proposed work covers the principal investigator and a research student.

| | Year 1 | Year 2 | Year 3 |
|---|---|---|---|
| Principal investigator | Representation of nu-calculus in Coq | Rule-based reasoning for state | [Logic for state and continuations] |
| | | Object calculus and test cases | Reasoning with object identity |
| Research student | Training and background research. Taught postgraduate course. | Equational and relational reasoning for nu-calculus in Coq. | Demonstration of mechanised nu-calculus reasoning. Dissertation |

In order to provide the necessary focus for a PhD dissertation, the plan for the research student concentrates on mechanised reasoning for names, following initial work by the principal investigator on a framework for representing nu-calculus name binding. The timetable for this includes a period of initial training and background research, supported by the LFCS taught postgraduate course.

The tasks for the principal investigator, across all the objectives, are expected to overlap and reinforce each other. There is still a certain ordering though, which reflects the act that some objectives are more immediately ready, and later ones would benefit from understanding from earlier ones.

## Milestones

| | | | | |
|---|---|---|---|---|
| 1. | Mechanised reasoning on names | Scheme for representing nu-calculus | Year 1 | PI |
| | | Implementation in Coq | Year 2 | RS |
| | | Demonstration and dissertation write-up | Year 3 | RS |
| 2. | Reasoning with state | Rule-based reasoning for state | Year 2 | PI |
| | | [Logic for state and continuations] | Year 3 | PI |
| 3. | Object identity and local state | Design of object calculus and test cases | Year 2 | PI |
| | | Reasoning principles for object identity | Year 3 | PI |