FreshML: A Fresh Approach to Name Binding in Metaprogramming Languages

Final Report[*]

Andrew M. Pitts

February 2005

Abstract

The FreshML project made a fundamental and rigorously-founded extension to functional programming languages. The extension has to do with metaprogramming, *the activity of creating software systems that manipulate syntactical structures (interpreters, compilers, proof checkers, proof assistants, etc). In all but the most trivial cases, these syntactical structures involve name binding, with associated notions of free and bound names, renaming of bound names, substitution of terms for free variables, and so on. It is generally agreed that this important aspect of representing and computing with syntax is not catered for satisfactorily in existing functional programming languages commonly used for metaprogramming activities. The project addressed this issue by developing and applying a new mathematical model of names and binding called "nominal sets", based on simple, but subtle ideas to do with permutations of names that first arose in mathematical logic in the 1930s. This model was the basis for the design of a "Fresh" patch of the Objective Caml functional language that relieves the user from having to deal with many tedious and error-prone details when programming with names and binders, while still remaining close to informal practice. The project also explored applications of nominal sets for programming language semantics and for logic programming.*

1 Background and Context

Typed functional programming languages in the ML tradition, such as Objective Caml (`www.ocaml.org`) and Haskell (`www.haskell.org`), provide their users with high-level facilities that greatly simplify one of the main tasks for which these languages were intended, namely *metaprogramming*—the construction and manipulation of syntactical structures. In these languages the implementation automatically takes care of the low-level details of constructing and deconstructing data values. Thus users of these functional metaprogramming languages can specify datatypes for representing "object-level" syntactical structures in quite an abstract fashion: they just need to declare the types of the various functions for constructing the data. Then they can use patterns to indicate how values should be decomposed by algorithms that operate on such data. In particular, the declaration of recursive, meta-level functions for manipulating the parse trees of object-level languages is made much simpler (and hence less error prone) through the use of patterns to match against parts of parse trees.

Unfortunately a pervasive problem spoils this rosy picture: object-level languages often involve *binding operations*. In this case meta-level programs only make sense, or at least only have good properties, when we operate not on parse trees themselves, but on α-equivalence classes of parse trees, identifying trees differing only in the names of bound entities. At the moment metaprogrammers deal with this case-by-case according to the nature of the object-level language being implemented, using a self-imposed discipline. For example, they might work out (not so hard) and then correctly use (much harder) some "nameless" representation of α-equivalence classes of parse trees in the style of de Bruijn [7]. The tedious and error-prone nature of *ad hoc* solutions to this semantically trivial, but pragmatically non-trivial issue of α-equivalence is widely acknowledged [21, Sect.13]. There is a clear need for better automatic support for object-level α-equivalence in metaprogramming languages. The FreshML project has provided a way of meeting this need by developing some new theory that seems useful in practice.

2 Achievements of the Research

The FreshML project more than fully met its original objectives, which were:

1. To develop the FM-sets model [14] of abstract syntax with binders (see Section 2.2).

2. To design and implement a functional programming language based on that model (see Section 2.1).

[*]The FreshML project ran from July 2001 to December 2004 and was funded by EPSRC grant GR/R07615/1 and a donation from Microsoft Research Ltd.

3. To discover, through experimentation, whether the model and programming language provide a useful idiom for metaprogramming tasks involving syntax with binders (see Sections 2.3 and 2.4).

In fact the research also developed in ways that were unforeseen at the time the original proposal was formulated; these developments are outlined in Section 2.5.

2.1 Fresh Objective Caml

The main piece of software produced by this project is Shinwell's "Fresh" patch of the Objective Caml language. The latter is a product of INRIA Rocquencourt, who provide open access to source code, documentation and tools at `www.ocaml.org`. Shinwell's Fresh O'Caml [22] makes the distinctive features of FreshML, as described in [25], available in the context of what is a well-regarded and increasingly widely used general-purpose functional programming language in the ML family. Specifically, the Fresh patch extends Objective Caml with a general-purpose type construction, written `<<bty>>ty`, for binding names of user-declared type `bty` in expressions of arbitrary type `ty`. This new type construction is used in declarations of types representing object-level syntax to specify information about name binding operations. For example, suppose we want to represent expressions of a small ML-like language with the following syntactic forms:

x	value identifier
$\mathit{fun}\ x \rightarrow e$	function abstraction
$e_1\, e_2$	function application
$\mathit{let}\ x = e_1\ \mathit{in}\ e_2$	local value
$\mathit{let\ rec}\ f\, x = e_1\ \mathit{in}\ e_2$	local recursive function

In Fresh O'Caml we can declare a new type `vid` of bindable names for object-level value identifiers and then declare a datatype `expr` for the above expressions:

```
type expr = Vid of vid
          | Fun of <<vid>>expr
          | App of expr * expr
          | Let of expr *(<<vid>>expr)
          | Ltr of
              <<vid>>((<<vid>>expr) * expr)
```

In this declaration, types (such as tell the system which data constructors are binders and how their arguments are bound. For example in $\mathit{let\ rec}\ f\, x\ =\ e_1\ \mathit{in}\ e_2$ there is a binding occurrence of f whose scope is both e_1 and e_2; and a binding occurrence of x whose scope is just e_1. These binding scopes are reflected by the argument type `<<vid>>((<<vid>>expr) * expr)` of the constructor `Ltr` in the declaration of `expr`. This declarative specification of binding structure is particularly useful because, as proved in [24], the design of Fresh O'Caml endows datatypes like `expr` with two crucial properties:

Abstractness: *object-level expressions represented as values of the datatype are operationally equivalent in* Fresh O'Caml *if and only if they are α-equivalent in the object language.*

Concreteness: *values of such datatypes can be deconstructed by matching against patterns that explicitly name bound entities.*

Indeed, FreshML provides automatic language-level support for the common informal idiom which refers to α-equivalence classes via representative parse trees, with bound names changed "on the fly" to make them distinct among themselves and distinct from any other names in the current context of use. For example, the following Fresh O'Caml declaration of a function `subst` of type `expr -> vid -> expr -> expr` suffices for `subst e x e'` to compute (a representation of) the object-language expression obtained by capture-avoiding substitution of the expression represented by `e` for all free occurrences of the value identifier named `x` in the expression represented by `e'`.

```
let subst e x =
 let rec s e' =
   match e' with
      Vid x1 -> if x1 = x then e else e'
    | Fun(<<x1>>e1) -> Fun(<<x1>>(s e1))
    | App(e1, e2) -> App(s e1, s e2)
    | Let(e1, <<x1>>e2) ->
        Let(s e1, <<x1>>(s e2))
    | Ltr(<<f>>(<<x1>>e1, e2)) ->
        Ltr(<<f>>(<<x1>>(s e1), s e2))
 in s
```

This declaration not only endows `subst` with the expected properties, but is remarkably simple compared with how capture-avoiding substitution must be defined in pre-existing languages that are commonly used for metaprogramming tasks. In particular, in the above declaration the clauses dealing with substitution under a binder only have to specify the result when the bound name is sufficiently fresh.

A distinguishing feature of the FreshML-style representation of object-level binders is that it uses a meta-level binary operation `<<->>(-)` on names and values (combined with the ability to declare local fresh names), rather than a meta-level binding operation. This is in contrast to previous, "functional" approaches to representing object-level syntax modulo α-conversion, either based on meta-level typed λ-calculus [16], or on de Bruijn indexes/levels and presheaves [9] and seems to account for the expressiveness of the FreshML approach: the user of Fresh O'Caml gets expressive linguistic tools for manipulating binders that not only respect α-conversion, but also seem rather close to informal, "pencil-and-paper" practice.

2.2 Nominal sets and domains

Fresh O'Caml's novel features arise from a mathematical model of fresh names and name binding that was introduced in [14] and which makes use of the classical Fraenkel-Mostowski permutation model of set theory. That paper expresses its results in terms of a non-standard axiomatic set theory and our experience is that this formalism impedes the take up of the new ideas within computer science. Therefore we have developed an essentially equivalent, but more concrete description of the model as standard sets equipped with some simple extra structure (a permutation action for which every element of the set has "finite support"). These so-called *nominal sets* are introduced in [17]. The paper [24] and Shinwell's thesis [23] develop the rudiments of Plotkin-Scott style domain theory in nominal sets. This combines the ability to solve recursive domain equations with interesting new domain constructors for name abstraction and dynamic allocation of names. The new domain theory is used to prove that FreshML correctly represents syntactical data modulo α-equivalence. One of the exciting features of this domain theory is that it provides a notion of "support" that generalises the notion of "set of free variables" from syntactical data to abstract mathematical objects, such as extensional functions of higher type. This opens up new possibilities for the denotational semantics of computations on *open* expressions, such as occurs in *normalisation by evaluation* and *type-directed partial evaluation* [8], and for programming languages that mix syntax and semantics, such as *multi-staged metaprogramming* [21]. Much of the theory of "nominal domains" remains to be developed and applied.

2.3 Case studies

Fresh O'Caml sources, documentation and examples are available at `www.fresh-ocaml.org`. We have used the language for a number of case studies: type inference algorithms; normalisation-by-evaluation for untyped λ-calculus and for Girard-Reynolds polymorphic λ-calculus; and the calculation of labelled transitions in the Milner-Parrow-Walker π-Calculus (see [13]). We have also used it in as yet unpublished joint work with Sheard (PSU, Oregon) to explore the denotational semantics of MetaML [26, 21]. This work showed up one of the strengths of Fresh O'Caml: it is extremely useful for rapidly prototyping algorithms that have to manipulate expressions with free names (semantics-based interpreters for MetaML, in this case). Since it allows one to remain quite close to familiar informal practices concerning bound names, new implementation ideas can be tried out quickly; but at the same time one avoids getting bogged down in the error-prone book-keeping usually associated with respecting α-conversion of explicitly given bound names. This also make Fresh O'Caml useful as a teaching aid; we have used it as an adjunct to a third-year undergraduate course on type systems (`www.cl.cam.ac.uk/Teaching/2004/Types/`).

The biggest test of Fresh O'Caml so far has been conducted in collaboration with the Acute project (`www.cl.cam.ac.uk/users/pes20/acute/`) at Cambridge and INRIA Rocquencourt. Acute is a language with features for typed distributed communication [20]. Its prototype implementation is of the order of 20,000 lines of Fresh O'Caml code. This represents a substantial experiment in using the FreshML constructs for programming with explicit names and binders. The collaboration has led to a more robust Fresh O'Caml implementation and has brought out several issues that are important for its large-scale use: the importance of delayed permutations for performance, name-indexed hash tables, and name-variety-indexed binders. Solutions for all of these have been proposed and will be investigated as part of the EPSRC project beginning in 2005 on *Naming, Distribution, and Versioning: Programming Language Design and Implementation* (GR/T11715), led by Sewell.

2.4 Nominal logics

Beyond its application to the design and properties of Fresh O'Caml, the theory of nominal sets has the potential to improve both the state of the art and the state of mechanised support for the ubiquitous issues of *naming*, *name binding*, *locality* and *freshness of names* in programming language semantics (both operational and denotational semantics). In order to realise that potential it is first necessary to develop formal logics that express the way that these things are dealt with using nominal sets. So we investigated such logics as part of the FreshML project. In [12], Gabbay describes a higher-order logic, FM-HOL, that bears the same relationship to (a generalised version of) Fraenkel-Mostowski set theory as does Church's higher-order logic to classical set theory. In [11], he reports on his development of machine-assistance for proving theorems in FM-HOL, using Paulson's generic theorem proving environment Isabelle (`www.cl.cam.ac.uk/Research/HVG/Isabelle/`). An alternative approach (as yet unpublished) has recently been taken by Urban. Rather than developing a new Isabelle object-logic, he uses the standard and extensive Isabelle/HOL logic to develop the properties of nominal sets relevant to programming semantics, such as its principle of induction for structure modulo α-equivalence [14, Theorem 6.8]. Both of these efforts have highlighted how useful it would be to have the nominal logic notions of "name-permutation" and "finite support" built into the *metalogic*. There are theoretical issues to be solved to achieve this; but the main obstacle is practical: re-engineering a major theorem-proving system such as Isabelle is a huge undertaking.

2.5 Nominal computational logic

Our experience using nominal sets shows that the "freshness" of a name with respect to an element of a nominal set emerges as a central concept, even though it is defined in terms of simpler things (name-permutation and quantification over finite sets of names). By taking name-freshness as well as name-permutation as primitive, Pitts was able to develop a first-order theory, *nominal logic* [17], which is incomplete, but nevertheless seems to capture the practically important aspects of modelling names and binders using nominal sets. This first-order nominal logic also turned out to have interesting applications for fully automatic (rather than machine-assisted) computational logic, as we now describe.

By examining the equational part of nominal logic, Urban, Pitts and Gabbay developed a generalisation of first-order unification to the practically important case of equations between terms involving binding operations [28]. A substitution of terms for variables solves such an equation if it makes the equated terms α-equivalent, i.e. equal up to renaming bound names. For the applications they have in mind (computing with rule schemas in structural operational semantics), one must consider the simple, textual form of substitution in which names occurring in terms may be captured within the scope of binders upon substitution. They are able to use the approach to binding characteristic of this project, in which bound entities are explicitly named (rather than using nameless, de Bruijn-style representations), and yet get a version of this form of substitution that respects α-equivalence and possesses good algorithmic properties. They achieve this by adapting two existing ideas. The first one is terms involving explicit substitutions of names for names, except that they only need to use explicit *permutations* (bijective substitutions). The second one is that the unification algorithm should solve not only equational problems, but also problems about the freshness of names for terms. They show that there is a simple generalisation of classical first-order unification problems to this setting which retains most of the latter's pleasant properties: unification problems involving α-equivalence and freshness are decidable; and solvable problems possess most general solutions. (However, the most efficient algorithm for nominal unification known so far is quadratic rather than linear in the size of the problem.)

Following on from this, Cheney and Urban have developed a prototype logic programming language, AlphaProlog, for declaring and querying relations between terms modulo α-equivalence [6]. AlphaProlog not only relies on the unification algorithm from [28], but also on the proof theory of nominal logic developed by Cheney and Gabbay [10, 5]. Some initial difficulties reported in [27] seem to have been overcome [4]; but much remains to be understood about the expressive power and practical utility of this extension of logic programming.

3 Project Plan Review

Comparison with original plan When this project began we planned just to implement a prototype version of FreshML containing some, but not all the features of a fully-fledged ML-like language. One of the novel features envisaged in the original design of FreshML [18] was to enforce the condition "freshly chosen names do not occur in the support of the final result". This was to be achieved through an extension of ML's usual type inference at compile-time, by deducing information about a relation of *freshness* of names for expressions that is a decidable approximation to the (in general undecidable) nominal sets notion of "not-in-the-support-of". The result is a *pure* functional programming language: static freshness inference ensures the dynamics of replacing insufficiently fresh names on the fly is referentially transparent. Purity makes reasoning about program properties simpler and is desirable, but our static freshness inference which achieved it had a drawback: since freshness is only an approximation of "not-in-the-support-of", the type-checker inevitably rejects some algorithms that do in fact respect α-conversion of bound names. In trying to "program round" this difficulty in specific examples, we made a key discovery: perhaps surprisingly, static freshness inference is not necessary for the crucial Abstractness property mentioned in Section 2.1 to hold. We can have Abstractness in the presence of the Concreteness property also mentioned there, so long as the operational semantics of [18] is modified to make the declaration of fresh bindable names *generative* like some other sorts of names (references, exceptions and type names) are in ML. (This fact is not at all obvious and requires proof, which we finally gave, after some false starts, in [24].) Ditching freshness inference at the expense of a moving to a less "pure" functional programming language was a huge win. It enabled us to integrate the simplified FreshML design into the fully-fledged Objective Caml language, making experimentation much easier and user-adoption much more likely. (For example, the Acute work mentioned in Section 2.3 would have been impossible without this development.)

Staff The original Project Plan called for an experienced Research Associate to carry out FreshML design and implementation work. Despite considerable efforts, we failed to recruit at the post-doctoral level and fell back on employing Mark Shinwell as a post-graduate Research Assistant. The money saved by this lower-level appointment enabled us to employ Murdoch Gabbay as a Research Associate to work on the theoretical and theorem-proving aspects of the project for two years rather than the one year originally envisaged (at the expense of also not employing 5% of a Computer Officer and Administrator). On leaving the project, Gabbay moved to the INRIA Futurs Laboratory in Paris as a research fellow. As it turned out, appointing Shinwell as a Research Assistant was very effective. Not only did he implement Fresh O'Caml, but he also completed a PhD thesis [23] containing a very interesting blend of theory and practice, which was successfully defended in February 2005. Shinwell is being employed as a Research Associate on a new project beginning in 2005 (see Section 7). Mike Gordon was named as a co-investigator in the original Project Plan, but played less of a role than planned: this was due to the fact that our case studies concentrated on functional metaprogramming and not, as originally envisaged, also on building prototype theorem-proving systems.

Collaboration The project interacted with the Programming Principles and Tools group at Microsoft Research Cambridge—particularly Simon Peyton Jones, Mark Shields (now at Galois Connections Inc, USA) and Nick Benton. (This group employed Mark Shinwell as an intern prior to his employment on this project.) With Simon Peyton Jones and Mark Shields, and also Koen Claessen (Chalmers), we explored issues to do with the implementation of FreshML and in particular the possibility of a call-by-need version. The desirability of such a "Fresh-Haskell" seems clear enough, but more work is needed to understand whether there are efficient lazy implementations of FreshML's characteristic name-permutation operations; furthermore Haskell puts great store on encapsulating impure effects, whereas we have found the way Fresh O'Caml mixes up dynamic creation of fresh names with expression evaluation to be very convenient. As a result of Pitts' visit to Portland, Sheard has produced a useful Haskell library that provides a "poor man's" version of FreshML's characteristic features without having to re-implement Haskell itself. We also had discussions with Benton about the domain theory that we developed in nominal sets (Section 2.2); this has been used by Benton and Leperchey as the basis of a new analysis of program equivalences in higher-order functional programming with local state [2].

Pitts and Gabbay also collaborated on nominal logic with Christian Urban, Michael Norrish (both Research Fellows funded by Cambridge Colleges) and James Cheney. Cheney was a graduate student from Cornell University who accompanied his advisor Morrisett on a visit to Cambridge in 2003; he subsequently went on to complete a PhD thesis on nominal logic [5], containing important results for the work described in Section 2.5.

4 Research Impact and Benefit to Society

The most important impact of this project has been to promote within the programming languages research community the fundamental ideas behind the nominal sets model—the use of name-permutations and the notion of "finite support"—as best-practice when dealing with syntactical structure involving names and binders. There is already evidence of their take-up among researchers: they have been used by Caires and Cardelli in their spatial logic for concurrency [3]; by Abramsky *et al* [1] in the context of their games semantics to solve a long-standing full abstraction problem to do with dynamic allocation of names; by Schöpp and Stark [19] in a dependent type theory with names and binding; and (indirectly) by Nanevski and Pfenning in their work on modal types in functional metaprogramming [15]. However, it is very difficult to change the existing orthodoxy, which relies on (total) functions to represent binders, because of the legacy of mature systems based upon that method of representation (whose drawbacks we discussed at the end of Section 2.1). Nevertheless, we believe the ideas behind the nominal sets model are sufficiently simple to stand a good chance of being widely adopted as standard tools by people who create and use formal descriptions of programming languages and logics. We plan to promote them through expository material and teaching (see Section 6); and Harper (CMU) is using them in a forthcoming graduate text on *Programming Languages: Theory and Practice* (`www.cs.cmu.edu/~rwh/plbook/`, Chapter 5).

For our main software product, Fresh O'Caml, to itself have a lasting impact outside the research community would require it to be folded in to a future release of the Objective Caml language. The changes made by the current Fresh patch are too great and too inefficient for there to be much prospect of that happening at the moment; the situation may change after further development as part of the *Naming, Distribution, and Versioning* project (see Section 7). In fact we believe that, rather than a general-purpose programming language, the development of domain-specific tools for metaprogramming based on the ideas of FreshML are probably the way ahead.

5 Explanation of Expenditure

There was no significant variation from the original spending plans other than described in the paragraph on Staff in Section 3. The donation from Microsoft Research Ltd was used to fund: the visit of Koen Claessen from Chalmers University during November 2001; a graduate internship for Peter White during the summer of 2002 to work on the FreshML implementation; the visit of Pitts to OGI (Portland, Oregon) for October 2003; and for various project-related conference travel.

6 Dissemination Activities

Papers arising from the research carried out by the FreshML project have so far been published in the journals *Information and Computation* [17], *Theoretical Computer Science* [28, 24] and *Formal Aspects of Computing* [14]. Conference papers and talks were given at the following international conferences and workshops: TACS 20001, Sendai, Japan (invited); Workshop on 35 years of Automath, Heriot-Watt, Edinburgh (refereed) [12]; ICALP 2002, Malaga, Spain (invited); MERLIN 2003, Uppsala, Sweden (refereed) [22]; ICFP 2003, Uppsala, Sweden (refereed) [25]; CSL'03 & KGC, Vienna, Austria (refereed) [28]; APPSEM 2004, Tallinn, Estonia (refereed) [24]; CTCS 2004, Copenhagen (invited); SOS 2004, London (invited).

Urban gave a course on "Nominal Reasoning" at the 16th European Summer School in Logic, Language and Information (ESSLLI 2004), Nancy, France, August 2004. Pitts will give a course on "Nominal Syntax and Semantics" at the Summer School of the EU Thematic Network on Applied Semantics II (IST-2001-38957) to be held at Frauenchiemsee, Germany, in September 2005. In connection with the latter, Pitts is working on an expository publication setting out the main ideas behind the nominal sets model of names and binders and their applications in semantics.

7 Further Research Activities

Further development of Fresh O'Caml will take place as part of the EPSRC-funded project led by Sewell on *Naming, Distribution, and Versioning: Programming Language Design and Implementation* (GR/T11715) that started in 2005 and that employs Shinwell as a Research Associate in the University of Cambridge Computer Laboratory. Gabbay was awarded an EPSRC Visiting Fellowship (*Rewriting Frameworks*, EP/C517148) to collaborate with Fernandez and Mackie at King's College London on nominal rewriting during 2004/5. Pitts plans to develop further computational aspects of nominal sets (see Section 2.5) in collaboration with Fernandez and Mackie, for which EPSRC funding has been requested (linked proposals EP/D000459/1 and EP/D501016/1).

References

[1] S. Abramsky, D. R. Ghica, A. S. Murowski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings LICS'04* . IEEE Computer Society Press, 2004.

[2] P. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proceedings TLCA'05, Nara, Japan*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[3] L. Caires and L. Cardelli. A spatial logic for concurrency (part II). In *Proceedings CONCUR'02, Brno, Czech Republic*, volume 2421 of *Lecture Notes in Computer Science*, pages 209–225. Springer-Verlag, 2002.

[4] J. Cheney. The complexity of equivariant unification. In *Proceedings ICALP'04, Turku, Finland*, volume 3142 of *Lecture Notes in Computer Science*, pages 332–344. Springer-Verlag, 2004.

[5] J. Cheney. *Nominal Logic Programming*. PhD thesis, Cornell University, August 2004.

[6] J. Cheney and C. Urban. αProlog, a fresh approach to logic programming modulo α-equivalence. In *Proceedings UNIF'03, Valencia, Spain*, number DSIC-II/12/03 in Departamento de Sistemas Informáticos y Computación Technical Report Series. Universidad Politécnica de Valencia, 2003.

[7] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Math.*, 34:381–392, 1972.

[8] P. Dybjer and A. Filinski. Normalization and partial evaluation. In G. Barthe, P. Dybjer, and J. Saraiva, editors, *Applied Semantics, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science, Tutorial*, pages 137–192. Springer-Verlag, 2002.

[9] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings LICS'99*, pages 193–202. IEEE Computer Society Press, 1999.

[10] M. Gabbay and J. Cheney. A proof theory for nominal logic. In *Proceedings LICS'04*. IEEE Computer Society Press, 2004.

[11] M. J. Gabbay. Automating Fraenkel-Mostowski syntax. In V. A. Carreno, C. A. Munoz, and S. Tahar, editors, *Track B Proceedings of TPHOLs 2002, Hampton, Virginia, USA.*, NASA CP-2002-211736, pages 60–70. NASA Langley Research Center, 2002.

[12] M. J. Gabbay. FM-HOL, a higher-order theory of names. In F. Kamareddine, editor, *Workshop on Thirty Five years of Automath, Informal Proceedings*. Heriot-Watt University, Edinburgh, Scotland, April 2002.

[13] M. J. Gabbay. The π-calculus in FM. In Fairouz Kamareddine, editor, *Thirty-Five Years of Automating Mathematics*, volume 28 of *Applied Logic*, pages 71–123. Kluwer, 2003.

[14] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[15] A. Nanevski. Meta-programming with names and necessity. In *Proceedings ICFP'02, Pittsburgh, Pennsylvania*, pages 206–217. ACM Press, 2002.

[16] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proceedings PLDI'88*, pages 199–208. ACM Press, 1988.

[17] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.

[18] A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In R. Backhouse and J. N. Oliveira, editors, *Proceedings MPC 2000, Ponte de Lima, Portugal*, volume 1837 of *Lecture Notes in Computer Science*, pages 230–255. Springer-Verlag, 2000.

[19] U. Schöpp and I. D. B. Stark. A dependent type theory with names and binding. In *Proceedings CSL'04, Karpacz, Poland*, volume 3210 of *Lecture notes in Computer Science*, pages 235–249. Springer-Verlag, 2004.

[20] P. Sewell, J. J. Leifer, K. Wansbrough, M. Allen-Williams, F. Zappa Nardelli, P. Habouzit, and V. Vafeiadis. Acute: high-level programming language design for distributed computation. Design rationale and language definition. Technical Report UCAM-CL-TR-605, University of Cambridge Computer Laboratory, October 2004.

[21] T. Sheard. Accomplishments and research challenges in meta-programming. In W. Taha, editor, *Proceedings SAIG 2001, Florence, Italy*, volume 2196 of *Lecture Notes in Computer Science*, pages 2–44. Springer, 2001.

[22] M. R. Shinwell. Swapping the atom: programming with binders in Fresh O'Caml. In *Proceedings MERLIN'03, Uppsala, Sweden*. ACM Press, 2003.

[23] M. R. Shinwell. *The Fresh Approach: Functional Programming with Names and Binders*. PhD thesis, University of Cambridge Computer Laboratory, 2005. Available as University of Cambridge Computer Laboratory Technical Report UCAM-CL-TR-618.

[24] M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 2005. To appear. A preliminary version appears in the proceedings of the Second workshop of the EU FP5 IST thematic network IST-2001-38957 APPSEM II, Tallinn, Estonia, April 2004.

[25] M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. FreshML: Programming with binders made simple. In *Proceedings ICFP'03, Uppsala, Sweden*, pages 263–274. ACM Press, 2003.

[26] W. Taha. *Multi-Stage Programming: Its Theory and Applications*. PhD thesis, Oregon Graduate Institute of Science and Technology, Portland, Oregon, USA, 1999.

[27] C. Urban and J. Cheney. Avoiding equivariance in alpha-prolog. In *Proceedings TLCA'05, Nara, Japan*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[28] C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473–497, 2004.