A Calculus for Symbolic Names Management

D.Ancona, E.Moggi[*]
DISI, Univ. of Genova, v. Dodecaneso 35, 16146 Genova, Italy
`email: {davide,moggi}@disi.unige.it`

January 22, 2004

Abstract

We define a basic calculus ML^N for manipulating symbolic names inspired by λ-calculi with extensible records. The resulting calculus supports the use of symbolic names for meta-programming and programming in-the-large, it subsumes Ancona and Zucca's CMS, and partly Nanevski and Pfenning's $\nu^\Box$, and seems able to model some aspects of the mechanism of Java class loaders. We present two different extensions of the basic calculus, the first consider the interaction with computational effects (in the form of imperative computations), the second shows how CMS can be naturally encoded into ML^N.

1 Introduction

We introduce a basic λ-calculus ML^N for manipulating (symbolic) names. The calculus involves three distinct concepts (besides names $X \in \mathsf{N}$):

- terms $e \in \mathsf{E}$, a closed term corresponds to an *executable* program
- name resolvers, mapping names to terms, more specifically they denote partial functions $\mathsf{N} \overset{fin}{\to} \mathsf{E}$, we write $r.X$ for the term obtained by applying r to *resolve* name X
- fragments *box* $r.e$ are terms abstracted w.r.t. a resolver r, i.e. they denote functions $(\mathsf{N} \overset{fin}{\to} \mathsf{E}) \to \mathsf{E}$, we write $e\langle r\rangle$ for the term obtained by *linking* fragment e using resolver r.

The aim of ML^N is to provide a uniform framework for modeling several uses of symbolic names in programming. We were motivated by the use of symbolic names for programming in-the-large, like that supported by module systems [Car97, AZ02], and for meta-programming, more specifically partial evaluation [JGS93, Dav96], staging [Tah99, She01], and run-time code generation [DP01, Nan02]. ML^N turns out to be a simple core calculus with a limited form of extensible records [CM94], indeed a record amounts to a partial function mapping names (of components) to their values.
ML^N captures smoothly most of the peculiar aspects of CMS and $\nu^\Box$ (except freshness), while overcoming some deficiencies and (unnecessary) complexities of these calculi.

- In CMS recursion is bundled in mixin, and removing it results in a very in-expressive calculus. On the contrary, ML^N is an interesting calculus even without recursion. The addition of recursion to ML^N follows mainstream approaches (either a fix-point combinator *fix* $x.e$ or mutual recursive declarations *let* ρ *in* e) and is orthogonal to the features for name management.

  Moreover, in ML^N one can express some form of dynamic linking, whereas in CMS operations on modules corresponds to traditional static linking. In particular, it seems able to model some aspects of the mechanism of Java class loaders [LB98].
- In $\nu^\Box$ the typing rules for $\Box$-types are inspired by S4 modal logic and are quite restrictive, such restrictions have no reason to exist in ML^N. To describe interesting examples of meta-programming in $\nu^\Box$ one has to exploit features such as creation of fresh names and support polymorphism, while in ML^N the same examples can be described in a simpler way, by exploiting the name management facilities borrowed from CMS.

[*]Supported by EU project DART IST-2001-33477 and thematic network APPSEM II IST-2001-38957

2 A basic calculus with names: ML^N

This section describes syntax and type system of ML^N, by focusing on the novel features. The syntax is abstracted over basic types b, term variables $x \in \mathsf{X}$, resolver variables $r \in \mathsf{R}$ and symbolic names $X \in \mathsf{N}$:

- $\boxed{\tau \in \mathsf{T} ::= b \mid \tau_1 \to \tau_2 \mid [\Sigma|\tau]}$ types, where $\Sigma \in \mathbf{\Sigma} \overset{\Delta}{=} \mathsf{N} \overset{fin}{\to} \mathsf{T}$ is a signature $\{X_i : \tau_i | i \in m\}$
- $\boxed{e \in \mathsf{E} ::= x \mid \lambda x.e \mid e_1\ e_2 \mid er.X \mid e\langle er\rangle \mid box\, r.e}$ terms
- $\boxed{er \in \mathsf{ER} ::= r \mid ? \mid er\{X : e\}}$ name resolvers.

We give an informal semantics of ML^N. The type $[\Sigma|\tau]$ classifies fragments which produce a term of type τ when linked with a resolver for Σ. The terms $er.X$ and $e\langle er\rangle$ use er to *resolve* name X and to *link* fragment e. The term $box\, r.e$ *represents* the fragment obtained by abstracting e w.r.t. r. The resolver ? cannot resolve any name, while $er\{X : e\}$ resolves X with e and *delegates* the resolution of other names to er.

References

[AZ99] Davide Ancona and Elena Zucca. A primitive calculus for module systems. In *Proc. Int'l Conf. Principles & Practice Declarative Programming*, volume 1702 of *LNCS*, pages 62–79. Springer-Verlag, 1999.

[AZ02] D. Ancona and E. Zucca. A calculus of module systems. *J. Funct. Programming*, 12(2):91–132, March 2002. Extended version of [AZ99].

[Car97] Luca Cardelli. Program fragments, linking, and modularization. In *Conf. Rec. POPL '97: 24th ACM Symp. Princ. of Prog. Langs.*, pages 266–277, 1997.

[CM94] L. Cardelli and J. C. Mitchell. Operations on records. In C. A. Gunter and J. C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*, pages 295–350. The MIT Press, Cambridge, MA, 1994.

[Dav96] R. Davies. A temporal-logic approach to binding-time analysis. In *the Symposium on Logic in Computer Science (LICS '96)*, pages 184–195, New Brunswick, 1996. IEEE Computer Society Press.

[DP01] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[JGS93] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice Hall, 1993.

[LB98] S. Liang and G. Bracha. Dynamic class loading in the Java Virtual Machine. In *ACM Symp. on Object-Oriented Programming: Systems, Languages and Applications 1998*, volume 33(10) of *Sigplan Notices*, pages 36–44. ACM Press, October 1998.

[Nan02] Aleksandar Nanevski. Meta-programming with names and necessity. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP-02)*, ACM SIGPLAN notices, New York, October 2002. ACM Press.

[She01] T. Sheard. Accomplishments and research challenges in meta-programming. In W. Taha, editor, *Proc. of the Int. Work. on Semantics, Applications, and Implementations of Program Generation (SAIG)*, volume 2196 of *LNCS*, pages 2–46. Springer-Verlag, 2001.

[Tah99] W. Taha. *Multi-Stage Programming: Its Theory and Applications.* PhD thesis, Oregon Graduate Institute of Science and Technology, 1999.