# A Fresh Calculus for Name Management
# (joint work with D.Ancona)

Eugenio Moggi

moggi@disi.unige.it

DISI, Univ. of Genova

# Overall Aims

Core calculus $\mathsf{MML}^N_\nu$ supporting the use of symbolic names for

- programming in-the-large, like Ancona and Zucca's CMS [AZ02]
  - but mixins types are explained in terms of more elementary types

$$\text{mixin}[\Sigma_1; \Sigma_2] = \text{extensible record}[\Sigma_1] \to \text{fixed record}[\Sigma_2]$$

RISC versus CISC approach to design calculi

# Overall Aims

Core calculus $\mathsf{MML}^{N}_{\nu}$ supporting the use of symbolic names for

- programming in-the-large, like Ancona and Zucca's CMS [AZ02]
- meta-programming, like Nanevski and Pfenning's $\nu^{\Box}$ [Nan02,NP03]
  - but connection to S4 modal logic unnecessary/misleading, the key point is to make name resolvers explicit
  - $\nu^{\Box}$ and MetaML (MMML) are different approaches to the same problem
    - combine (safely) execution of closed code, and
    - manipulation of open code (as in partial evaluation)

  we would like to understand the trade-offs!

# Overall Aims

Core calculus $\mathsf{MML}_\nu^N$ supporting the use of symbolic names for

- programming in-the-large, like Ancona and Zucca's CMS [AZ02]
- meta-programming, like Nanevski and Pfenning's $\nu^\square$ [Nan02,NP03]
- capturing some aspects of Java multiple loaders [LY99]
  - loaders modeled by name resolvers

# Syntax of $\mathsf{MML}_\nu^N$

- $$\boxed{\begin{array}{rl} e \in \mathsf{E} ::= & x \mid \lambda x.e \mid e_1\ e_2 \mid \theta.X \mid \mathsf{b}(r)e \mid e\langle\theta\rangle \mid \\ & \mathsf{ret}\ e \mid \mathsf{do}\ x \leftarrow e_1; e_2 \mid \nu X.e \mid \ldots \end{array}}$$ terms
- $\theta \in \mathsf{ER} ::= r \mid ? \mid \theta\{X : e\}$ name resolvers

$X \in \mathsf{N}$ symbolic name, $x \in \mathsf{X}$ term variable, $r \in \mathsf{R}$ resolver variable

## Commentary to BNF

**name resolver)** $\theta$ denotes partial function $\mathsf{N} \stackrel{fin}{\to} \mathsf{E}$ from names to terms

**name resolution)** $\theta.X$ term obtained when $\theta$ resolves $X$

**fragment)** $\mathsf{b}(r)e$ denotes function $(\mathsf{N} \stackrel{fin}{\to} \mathsf{E}) \to \mathsf{E}$ from resolvers to terms

**linking)** $e\langle\theta\rangle$ term obtained when fragment $e$ is linked to $r$

## Syntax of $\mathsf{MML}_{\nu}^{N}$

- $$\boxed{\begin{array}{ll} e \in \mathsf{E}{::=} & x \mid \lambda x.e \mid e_1\ e_2 \mid \theta.X \mid \mathsf{b}(r)e \mid e\langle\theta\rangle \mid \\ & \mathsf{ret}\ e \mid \mathsf{do}\ x \leftarrow e_1; e_2 \mid \nu X.e \mid \ldots \end{array}}$$ terms
- $\theta \in \mathsf{ER}{::=} r \mid ? \mid \theta\{X : e\}$ name resolvers
- monadic metalanguage with operational Semantics a la [MF03] (CHAM like)
  - simplification $e \longrightarrow e'$ confluent relation defined as compatible closure
  - computation $Id \longmapsto Id' \mid \mathsf{done}$ describing how *configurations* may evolve

  it enforces simple equivalences, unlike operational semantics that bundle computation with a deterministic simplification strategy.

## Syntax of $\mathsf{MML}_{\nu}^{N}$

- $$\boxed{\begin{array}{rl} e \in \mathsf{E}::= & x \mid \lambda x.e \mid e_1\ e_2 \mid \theta.X \mid \textsf{b}(r)e \mid e\langle\theta\rangle \mid \\ & \textsf{ret}\ e \mid \textsf{do}\ x \leftarrow e_1; e_2 \mid \nu X.e \mid \ldots \end{array}}$$ terms
- $\theta \in \mathsf{ER}::= r \mid ? \mid \theta\{X\colon e\}$ name resolvers
- monadic metalanguage with operational Semantics a la [MF03] (CHAM like)
- name resolvers $\theta$ as extensible records (this is what's missing in $\nu^{\Box}$!)
  - resolvers are handled by simplification
  - calculus is *expressive* even with *second-class* resolvers

## Syntax of MML$_\nu^N$

- $$\boxed{\begin{array}{ll} e \in \mathsf{E} ::= & x \mid \lambda x.e \mid e_1\ e_2 \mid \theta.X \mid \mathsf{b}(r)e \mid e\langle\theta\rangle \mid \\ & \mathit{ret}\ e \mid \mathit{do}\ x \leftarrow e_1; e_2 \mid \nu X.e \mid \ldots \end{array}}$$ terms
- $\theta \in \mathsf{ER} ::= r \mid ? \mid \theta\{X : e\}$ name resolvers
- monadic metalanguage with operational Semantics a la [MF03] (CHAM like)
- name resolvers $\theta$ as extensible records (this is what's missing in $\nu^{\Box}$!)
- name generation $\nu X.e$ is a computational effect, as in FreshML [SGP03]
  - mathematical underpinning for freshness provided by FM-sets [GP99]
  - name generation *essential* to prevent accidental overriding of resolver
  - object language syntax modulo $\alpha$-conversion (as in FreshML) not our aim!
  - but in MML$_\nu^N$ names are pervasive: they occur in terms and in types

# Equivariance (and finite support)!

# Operational Semantics of $\mathsf{MML}_{\nu}^{N}$: Simplification

- $\boxed{e \in \mathsf{E}::= \quad x \mid \lambda x.e \mid e_1\ e_2 \mid \theta.X \mid \boldsymbol{b}(r)e \mid e\langle\theta\rangle \mid \textit{\textbf{ret}}\ e \mid \textit{\textbf{do}}\ x \leftarrow e_1; e_2 \mid \nu X.e}$
- $\theta \in \mathsf{ER}::= r \mid ? \mid \theta\{X{:}\,e\}$

**beta)** $(\lambda x.e_2)\ e_1 \longrightarrow e_2[x{:}\,e_1]$

**resolve)** $(\theta\{X{:}\,e\}).X \longrightarrow e$

**delegate)** $(\theta\{X{:}\,e\}).X' \longrightarrow \theta.X'$ if $X' \neq X$

**link)** $(\boldsymbol{b}(r)e)\langle\theta\rangle \longrightarrow e[r{:}\,\theta]$

## Operational Semantics of $\mathsf{MML}_\nu^N$: Computation

- $\boxed{e \in \mathsf{E} ::= \quad x \mid \lambda x.e \mid e_1\ e_2 \mid \theta.X \mid \mathsf{b}(r)e \mid e\langle\theta\rangle \mid \textsf{ret}\ e \mid \textsf{do}\ x \leftarrow e_1; e_2 \mid \nu X.e}$
- $\theta \in \mathsf{ER} ::= r \mid ? \mid \theta\{X : e\}$

$(\mathcal{X}|e, E)$ configurations: current name space $\mathcal{X} \subseteq_{fin} \mathsf{N}$, program fragment $e$ under consideration, and its evaluation context $\boxed{E \in \mathsf{EC} ::= \square \mid E[\textsf{do}\ x \leftarrow \square; e]}$

- Administrative steps

(A.0) $(\mathcal{X}|\textsf{ret}\ e, \square) \longmapsto$ done

(A.1) $(\mathcal{X}|\textsf{do}\ x \leftarrow e_1; e_2, E) \longmapsto (\mathcal{X}|e_1, E[\textsf{do}\ x \leftarrow \square; e_2])$

(A.2) $(\mathcal{X}|\textsf{ret}\ e_1, E[\textsf{do}\ x \leftarrow \square; e_2]) \longmapsto (\mathcal{X}|e_2[x : e_1], E)$

- Name generation step

($\nu$) $(\mathcal{X}|\nu X.e, E) \longmapsto (\mathcal{X}, X|e, E)$ with $X$ **renamed to avoid clashes**, i.e. $X \notin \mathcal{X}$

# Addition of other computational effects *straightforward*!

# Type System $\mathcal{X};\Pi;\Gamma \vdash e\colon \tau$ and $\mathcal{X};\Pi;\Gamma \vdash \theta\colon \Sigma$

- $\mathcal{X} \subseteq_{fin}$ N current name space (finite set of names)
- $\boxed{\tau \in \mathsf{T}_{\mathcal{X}} ::= \tau_1 \to \tau_2 \mid [\Sigma|\tau] \mid M\tau \mid \ldots}$ $\mathcal{X}$-type
- $\Sigma \in \Sigma_{\mathcal{X}} \stackrel{\Delta}{=} \mathcal{X} \stackrel{fin}{\to} \mathsf{T}_{\mathcal{X}}$ $\mathcal{X}$-signature $\{X_i\colon \tau_i | i \in m\}$
- $\Pi\colon \mathsf{R} \stackrel{fin}{\to} \Sigma_{\mathcal{X}}$ $\mathcal{X}$-signature assignment for resolver variables
- $\Gamma\colon \mathsf{X} \stackrel{fin}{\to} \mathsf{T}_{\mathcal{X}}$ $\mathcal{X}$-type assignment for term variables

Contrary to record calculi $\mathcal{X}$ is finite (but may grow as computation progresses!)

# Type System $\mathcal{X};\Pi;\Gamma \vdash e\colon\tau$ and $\mathcal{X};\Pi;\Gamma \vdash \theta\colon\Sigma$

- $\mathcal{X} \subseteq_{fin}$ N current name space (finite set of names)
- $\boxed{\tau \in \mathsf{T}_{\mathcal{X}} ::= \tau_1 \to \tau_2 \mid [\Sigma|\tau] \mid M\tau \mid \ldots}$ $\mathcal{X}$-type
- $\Sigma \in \Sigma_{\mathcal{X}} \stackrel{\Delta}{=} \mathcal{X} \stackrel{fin}{\to} \mathsf{T}_{\mathcal{X}}$ $\mathcal{X}$-signature $\{X_i\colon\tau_i | i \in m\}$
- $\Pi\colon \mathsf{R} \stackrel{fin}{\to} \Sigma_{\mathcal{X}}$ $\mathcal{X}$-signature assignment for resolver variables
- $\Gamma\colon \mathsf{X} \stackrel{fin}{\to} \mathsf{T}_{\mathcal{X}}$ $\mathcal{X}$-type assignment for term variables

## Sample of Typing Rules

$$\text{link}\ \frac{\begin{array}{c}\mathcal{X};\Pi;\Gamma \vdash e\colon[\Sigma|\tau] \\ \mathcal{X};\Pi;\Gamma \vdash \theta\colon\Sigma'\end{array}}{\mathcal{X};\Pi;\Gamma \vdash e\langle\theta\rangle\colon\tau}\ \Sigma \subseteq \Sigma' \qquad \nu\ \frac{\mathcal{X},X;\Pi;\Gamma \vdash e\colon M\tau}{\mathcal{X};\Pi;\Gamma \vdash \nu X.e\colon M\tau}\ X \notin \mathrm{FV}(\Pi,\Gamma,\tau)$$

- (link) allows limited form of *width* subtyping
- $\vdash (\mathcal{X}|e,E)\colon\tau'$ well-formed configuration

## Generative Programming in $\mathsf{MML}_{\nu}^{N}$

Require name generation, and type- and *signature*-polymorphism

- component as fragment of type $[\Sigma|\tau]$
  - $\Sigma$ specifies the parameters needed for deployment

## Generative Programming in $\mathsf{MML}_{\nu}^{N}$

Require name generation, and type- and *signature*-polymorphism

- component as fragment of type $[\Sigma|\tau]$
- Generative programming support the dynamic manufacturing of customized components from elementary (highly reusable) components
- building block for generative programming are polymorphic functions of type

$$G\colon \forall p.[p,\Sigma_i|\tau_i] \to M[p,\Sigma|\tau]$$

  - result type is computational (generation may require computation)
  - *signature variable* $p$ classifies information passed to arguments of $G$, but not directly used/supplied by $G$.

# Comparison with MetaML (MMML)

$\text{MML}_\nu^N$ appears more expressive (also more fine-grained/verbose), and avoids the problems due to *scope extrusion* (more precise types).

- *open code* type $\langle\tau\rangle$ to correspond to $[\Sigma|\tau]$
  $\Sigma$ specifies what names need to be resolved
- $\lambda_M x.e$ computation (in MMML) to generate code for a $\lambda$-abstraction, becomes

$$\nu X.\quad \textit{do}\ u \leftarrow e[x\colon(\textbf{\textit{b}}(r')r'.X)];\quad \textit{ret}\ (\textbf{\textit{b}}(r)\lambda x.u\langle r\{X\colon x\}\rangle)$$

1. generate a fresh name $X$
2. compute fragment $u$ by evaluating $e$ with $x$ replaced by $\textbf{\textit{b}}(r')r'.X$
   resolver $r'$ for fresh name $X$ (and possibly other names)
3. return fragment for a $\lambda$-abstraction
   $r$ does not have to resolve $X$, since $u$ is linked to $r' = r\{X\colon x\}$

## Comparison with MetaML (MMML)

MML$_\nu^N$ appears more expressive (also more fine-grained/verbose), and avoids the problems due to *scope extrusion* (more precise types).

- *open code* type $\langle\tau\rangle$ to correspond to $[\Sigma|\tau]$
  $\Sigma$ specifies what names need to be resolved
- $\lambda_M x.e$ computation (in MMML) to generate code for a $\lambda$-abstraction, becomes

$$\nu X. \quad \textit{do}\ u \leftarrow e[x\colon(\textbf{b}(r')r'.X)]; \quad \textit{ret}\ (\textbf{b}(r)\lambda x.u\langle r\{X\colon x\}\rangle)$$

1. generate a fresh name $X$
2. compute fragment $u$ by evaluating $e$ with $x$ replaced by $\textbf{b}(r')r'.X$
   resolver $r'$ for fresh name $X$ (and possibly other names)
3. return fragment for a $\lambda$-abstraction
   $r$ does not have to resolve $X$, since $u$ is linked to $r' = r\{X\colon x\}$

# The End!