Generative Unbinding of Names

Andrew Pitts and Mark Shinwell
University of Cambridge CodeSourcery, Ltd

The FreshML project

2001–2005: Jamie Gabbay + AMP + Mark Shinwell + Christian Urban

"nominal sets" model of names, binding and freshness — based on properties that are invariant under permuting names

The FreshML project

2001–2005: Jamie Gabbay + AMP + Mark Shinwell + Christian Urban

"nominal sets" model of names, binding and freshness — based on properties that are invariant under permuting names

inspired by Fraenkel & Mostowski's 1930s permutation model of set theory

The FreshML project

2001–2005: Jamie Gabbay + AMP + Mark Shinwell + Christian Urban

“nominal sets” model of names, binding and freshness — based on properties that are invariant under permuting names

$\Longrightarrow$ FreshML = ML + features for manipulating syntax mod α-equiv.

Implementations: Shinwell’s Fresh patch for Ocaml, Pottier’s Cαml tool, Cheney’s FreshLib library for ghc.

The FreshML project

2001–2005: Jamie Gabbay + AMP + Mark Shinwell + Christian Urban

“nominal sets” model of names, binding and freshness — based on properties that are invariant under permuting names

$\Longrightarrow$ FreshML = ML + features for manipulating syntax mod α-equiv.

Implementations: Shinwell's Fresh patch for Ocaml, Pottier's Cαml tool, Cheney's FreshLib library for ghc.

all use generative unbinding of names

FreshML sig for name-binding

```
type  atm
type  α bnd
val   fresh : unit ⟶ atm
val   bind : atm * α ⟶ α bnd
val   unbind : α bnd ⟶ atm * α
val   (=) : atm ⟶ atm ⟶ bool
```

FreshML sig for name-binding

$$
\begin{array}{ll}
\texttt{type} & \texttt{atm} \\
\texttt{type} & \alpha\ \texttt{bnd} \\
\texttt{val} & \texttt{fresh} : \texttt{unit} \longrightarrow \texttt{atm} \\
\texttt{val} & \texttt{bind} : \texttt{atm} * \alpha \longrightarrow \alpha\ \texttt{bnd} \\
\texttt{val} & \texttt{unbind} : \alpha\ \texttt{bnd} \longrightarrow \texttt{atm} * \alpha \\
\texttt{val} & (=) : \texttt{atm} \longrightarrow \texttt{atm} \longrightarrow \texttt{bool}
\end{array}
$$

Closed values $a : \texttt{atm}$ come from a fixed, infinite set of "atoms".

FreshML sig for name-binding

```
type  atm
type  α bnd
val   fresh : unit ⟶ atm
val   bind : atm * α ⟶ α bnd
val   unbind : α bnd ⟶ atm * α
val   (=) : atm ⟶ atm ⟶ bool
```

fresh() creates a fresh atom:

$\boxed{\langle \vec{a}, \mathtt{fresh}() \rangle \longrightarrow \langle a' :: \vec{a}, a' \rangle}$ where $a' \notin \vec{a}$

FreshML sig for name-binding

```
type  atm
type  α bnd
val   fresh : unit ⟶ atm
val   bind : atm * α ⟶ α bnd
val   unbind : α bnd ⟶ atm * α
val   (=) : atm ⟶ atm ⟶ bool
```

$\mathtt{fresh}()$ creates a fresh atom:

$$\langle \vec{a}, \mathtt{fresh}() \rangle \longrightarrow \langle a' :: \vec{a}, a' \rangle$$ where $a' \notin \vec{a}$

current state = finite list of distinct atoms created so far

FreshML sig for name-binding

$$
\begin{array}{ll}
\texttt{type} & \texttt{atm} \\
\texttt{type} & \alpha\,\texttt{bnd} \\
\texttt{val} & \texttt{fresh} : \texttt{unit} \longrightarrow \texttt{atm} \\
\texttt{val} & \texttt{bind} : \texttt{atm} * \alpha \longrightarrow \alpha\,\texttt{bnd} \\
\texttt{val} & \texttt{unbind} : \alpha\,\texttt{bnd} \longrightarrow \texttt{atm} * \alpha \\
\texttt{val} & (=) : \texttt{atm} \longrightarrow \texttt{atm} \longrightarrow \texttt{bool}
\end{array}
$$

Closed values «a»v of type $\tau\,\texttt{bnd}$ are represented by pairs consisting of an atom a and a closed value $v : \tau$, created by evaluating $\texttt{bind}(a, v)$.

FreshML sig for name-binding

```
type  atm
type  α bnd
val   fresh : unit ⟶ atm
val   bind : atm * α ⟶ α bnd
val   unbind : α bnd ⟶ atm * α
val   (=) : atm ⟶ atm ⟶ bool
```

`unbind` carries out *generative unbinding*:

$$\langle \vec{a}, \texttt{unbind}(«a»v) \rangle \longrightarrow \langle a' :: \vec{a}, (a', v\{a'/a\}) \rangle$$

where $a' \notin \vec{a}$.

FreshML sig for name-binding

```
type  atm
type  α bnd
val   fresh : unit ⟶ atm
val   bind : atm * α ⟶ α bnd
val   unbind : α bnd ⟶ atm * α
val   (=) : atm ⟶ atm ⟶ bool
```

`unbind` carries out *generative unbinding*:

$$\langle \vec{a}, \mathtt{unbind}(\langle\!\langle a \rangle\!\rangle v) \rangle \longrightarrow \langle a' :: \vec{a}, (a', v\{a'/a\}) \rangle$$

where $a' \notin \vec{a}$.

rename all occurrences of a in v to be a'

Representing object-level languages

For example

$t ::= a \mid \lambda a.t \mid t\,t$ terms of the λ-calculus (all terms, open or closed, with variables given by atoms)

can be represented in FreshML by

closed values of the recursive data type

$$\begin{array}{lll} \tau & = & \texttt{V of atm} \\ & \mid & \texttt{L of } \tau \texttt{ bnd} \\ & \mid & \texttt{A of } \tau * \tau \end{array}$$

(More generally, the representation works the same way for terms over any nominal signature [Urban-Gabbay-AMP].)

$$\tau = \texttt{V of atm} \mid \texttt{L of } \tau \texttt{ bnd} \mid \texttt{A of } \tau * \tau$$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

$$\begin{aligned}
\ulcorner a \urcorner &\triangleq \texttt{V}\, a \\
\ulcorner \lambda a.t \urcorner &\triangleq \texttt{L}(\langle\!\langle a \rangle\!\rangle \ulcorner t \urcorner) \\
\ulcorner t_1\, t_2 \urcorner &\triangleq \texttt{A}(\ulcorner t_1 \urcorner, \ulcorner t_2 \urcorner)
\end{aligned}$$

$$\boxed{\tau = \mathtt{V}\ \mathtt{of}\ \mathtt{atm} \mid \mathtt{L}\ \mathtt{of}\ \tau\ \mathtt{bnd} \mid \mathtt{A}\ \mathtt{of}\ \tau * \tau}$$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

and syntax-manipulating functions can be coded nicely. E.g. capture-avoiding substitution, $t'[t/a]$, is given by $\mathtt{sub}\, a \ulcorner t \urcorner \ulcorner t' \urcorner$, where

$$\begin{aligned}
\mathtt{sub}\, x\, y\, y' = \mathtt{match}\ y'\ \mathtt{with}\ & \\
& \mathtt{V}\, x' \longrightarrow \mathtt{if}\ x = x'\ \mathtt{then}\, y\ \mathtt{else}\, y' \\
& \mid \mathtt{L}(\text{«}x'\text{»}z) \longrightarrow \mathtt{L}(\mathtt{bind}\, x' (\mathtt{sub}\, x\, y\, z)) \\
& \mid \mathtt{A}(z, z') \longrightarrow \mathtt{A}(\mathtt{sub}\, x\, y\, z, \mathtt{sub}\, x\, y\, z')
\end{aligned}$$

$$\tau = \mathtt{V\ of\ atm} \mid \mathtt{L\ of}\ \tau\ \mathtt{bnd} \mid \mathtt{A\ of}\ \tau * \tau$$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

and syntax-manipulating functions can be coded nicely. E.g. capture-avoiding substitution, $t'[t/a]$, is given by $\mathtt{sub}\, a \ulcorner t \urcorner \ulcorner t' \urcorner$, where

$$\begin{aligned}
\mathtt{sub}\, x\, y\, y' = \mathtt{match}\ y'\ \mathtt{with}\ & \mathtt{V}\, x' \longrightarrow \mathtt{if}\ x = x'\ \mathtt{then}\ y\ \mathtt{else}\ y' \\
& \mid \mathtt{L}(\text{«}x'\text{»}z) \longrightarrow \mathtt{L}(\mathtt{bind}\, x'(\mathtt{sub}\, x\, y\, z)) \\
& \mid \mathtt{A}(z, z') \longrightarrow \mathtt{A}(\mathtt{sub}\, x\, y\, z, \mathtt{sub}\, x\, y\, z')
\end{aligned}$$

This kind of pattern-match desugars to a use of unbind

$$\boxed{\tau = \mathtt{V\ of\ atm} \mid \mathtt{L\ of}\ \tau\ \mathtt{bnd} \mid \mathtt{A\ of}\ \tau * \tau}$$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

and syntax-manipulating functions can be coded nicely. E.g. capture-avoiding substitution, $t'[t/a]$, is given by $\mathtt{sub}\, a \ulcorner t \urcorner \ulcorner t' \urcorner$, where

$$\begin{aligned}
\mathtt{sub}\, x\, y\, y' = \mathtt{match}\ y'\ \mathtt{with}\quad & \\
& \mathtt{V}\, x' \longrightarrow \mathtt{if}\ x = x'\ \mathtt{then}\ y\ \mathtt{else}\ y' \\
& \mid \mathtt{L}\, z \longrightarrow \mathtt{let}(x', z') = \mathtt{unbind}\, z\ \mathtt{in} \\
& \qquad\qquad \mathtt{L}(\mathtt{bind}\, x'(\mathtt{sub}\, x\, y\, z')) \\
& \mid \mathtt{A}(z, z') \longrightarrow \mathtt{A}(\mathtt{sub}\, x\, y\, z, \mathtt{sub}\, x\, y\, z')
\end{aligned}$$

$$\boxed{\tau = \mathtt{V}\ \mathtt{of}\ \mathtt{atm} \mid \mathtt{L}\ \mathtt{of}\ \tau\ \mathtt{bnd} \mid \mathtt{A}\ \mathtt{of}\ \tau * \tau}$$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

and syntax-manipulating functions can be coded nicely. E.g. capture-avoiding substitution, $t'[t/a]$, is given by $\mathtt{sub}\, a \ulcorner t \urcorner \ulcorner t' \urcorner$, where

$$\begin{aligned}
\mathtt{sub}\, x\, y\, y' = \mathtt{match}\ y'\ \mathtt{with}\ & \\
& \mathtt{V}\, x' \longrightarrow \mathtt{if}\ x = x'\ \mathtt{then}\ y\ \mathtt{else}\ y' \\
& \mid \mathtt{L}\, z \longrightarrow \mathtt{let}(x', z') = \mathtt{unbind}\, z\ \mathtt{in} \\
& \qquad\qquad \mathtt{L}(\mathtt{bind}\, x'(\mathtt{sub}\, x\, y\, z')) \\
& \mid \mathtt{A}(z, z') \longrightarrow \mathtt{A}(\mathtt{sub}\, x\, y\, z, \mathtt{sub}\, x\, y\, z')
\end{aligned}$$

E.g. corresponding to $(\lambda b.\, a)[b/a] = \lambda c.\, b$, have:
$\langle [a, b], \mathtt{sub}\, a\, (\mathtt{V}\, b)\, (\mathtt{L}(«b»(\mathtt{V}\, a))) \rangle \longrightarrow^* \langle [a, b, c], \mathtt{L}(«c»(\mathtt{V}\, b)) \rangle$

$\tau = \texttt{V of atm} \mid \texttt{L of } \tau \texttt{ bnd} \mid \texttt{A of } \tau * \tau$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

Want:

Correctness of Representation: two λ-terms are α-equivalent, $t_1 =_\alpha t_2$, iff $\ulcorner t_1 \urcorner$ and $\ulcorner t_2 \urcorner$ are contextually equivalent closed values of type τ.

$\tau = \mathtt{V}\ \mathtt{of}\ \mathtt{atm} \mid \mathtt{L}\ \mathtt{of}\ \tau\ \mathtt{bnd} \mid \mathtt{A}\ \mathtt{of}\ \tau * \tau$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

Want:

Correctness of Representation: two λ-terms are α-equivalent, $t_1 =_\alpha t_2$, iff $\ulcorner t_1 \urcorner$ and $\ulcorner t_2 \urcorner$ are contextually equivalent closed values of type τ.

i.e. can be used interchangeably in any well-typed FreshML program without affecting the observable results of program execution

$\tau = \mathtt{V}\ \mathtt{of}\ \mathtt{atm} \mid \mathtt{L}\ \mathtt{of}\ \tau\ \mathtt{bnd} \mid \mathtt{A}\ \mathtt{of}\ \tau * \tau$

λ-terms t map onto closed values $\ulcorner t \urcorner : \tau$

Want:

Correctness of Representation: two λ-terms are α-equivalent, $t_1 =_\alpha t_2$, iff $\ulcorner t_1 \urcorner$ and $\ulcorner t_2 \urcorner$ are contextually equivalent closed values of type τ.

Proved for FreshML in Shinwell's thesis
(with a denotational semantics based on Gabbay-AMP "FM-sets").

Algorithms involving atoms

FreshML only has $(=) : \mathtt{atm} \longrightarrow \mathtt{atm} \longrightarrow \mathtt{bool}$.

Do other relations on atoms mess up the Correctness Property?

E.g. is it possible to have linearly ordered atoms, $(<) : \mathtt{atm} \longrightarrow \mathtt{atm} \longrightarrow \mathtt{bool}$?

Apparent problem: proof of Correctness relies on equivariance = invariance under atom-permutations.

Algorithms involving atoms

FreshML only has $(=) : \mathtt{atm} \longrightarrow \mathtt{atm} \longrightarrow \mathtt{bool}$.

Do other relations on atoms mess up the Correctness Property?

E.g. is it possible to have linearly ordered atoms, $(<) : \mathtt{atm} \longrightarrow \mathtt{atm} \longrightarrow \mathtt{bool}$?

Apparent problem: proof of Correctness relies on equivariance = invariance under atom-permutations.

$a = a'$ is equivariant,
but $a < a'$ appears not to be

Algorithms involving atoms

FreshML only has $(=) : \mathtt{atm} \longrightarrow \mathtt{atm} \longrightarrow \mathtt{bool}$.

Do other relations on atoms mess up the Correctness Property?

E.g. is it possible to have linearly ordered atoms, $(<) : \mathtt{atm} \longrightarrow \mathtt{atm} \longrightarrow \mathtt{bool}$?

Apparent problem: proof of Correctness relies on equivariance = invariance under atom-permutations.

Solution: take into account the current state of dynamically created atoms.

Observations on atoms

Extend FreshML with primitive functions

$$\texttt{obs} : \texttt{atm} * \cdots * \texttt{atm} \longrightarrow \texttt{int}$$

with *state-dependent* dynamics

$$\langle \vec{a}, \texttt{obs}(a_1, \ldots, a_k) \rangle \longrightarrow \langle \vec{a}, [\![\texttt{obs}]\!]_{\vec{a}}(a_1, \ldots, a_k) \rangle$$

Observations on atoms

Extend FreshML with primitive functions

$$\texttt{obs} : \texttt{atm} * \cdots * \texttt{atm} \longrightarrow \texttt{int}$$

with *state-dependent* dynamics

$$\langle \vec{a}, \texttt{obs}(a_1, \ldots, a_k) \rangle \longrightarrow \langle \vec{a}, [\![\texttt{obs}]\!]_{\vec{a}}(a_1, \ldots, a_k) \rangle$$

integer-valued function of $\vec{a}$ and $a_1, \ldots, a_k \in \vec{a}$ that is equivariant.
E.g. $[\![\texttt{obs}]\!]_{[a,b,c]}(a, c) = [\![\texttt{obs}]\!]_{[b,c,a]}(b, a)$

(We insist on equivariant functions in order to abstract away from concrete implementations of generativity.)

Examples of observations on atoms

Equality $[\![\mathtt{eq}]\!]_{\vec{a}}(a, a') \triangleq \begin{cases} 1 & \text{if } a = a', \\ 0 & \text{otherwise.} \end{cases}$

Linear order $[\![\mathtt{lt}]\!]_{\vec{a}}(a, a') \triangleq$

$$\begin{cases} 1 & \text{if } a \text{ occurs to the left of } a' \text{ in the list } \vec{a}, \\ 0 & \text{otherwise.} \end{cases}$$

Ordinal $[\![\mathtt{ord}]\!]_{\vec{a}}(a) \triangleq n$, if a is the nth element of the list $\vec{a}$.

Non-example: $[\![\mathtt{bad}]\!]_{\vec{a}}(a) = \alpha^{-1}(a)$, where $\alpha : \mathbb{N} \cong \mathbb{A}$ is some fixed enumeration of the set of atoms.

Main result of the paper

Theorem. The Correctness of Representation property

> for all λ-terms t_1, t_2, it is the case that $t_1 =_\alpha t_2$ iff $\ulcorner t_1 \urcorner$ and $\ulcorner t_2 \urcorner$ are contextually equivalent

holds no matter what (equivariant) observations on atoms we add to FreshML.

[Stated for λ-terms, but true for terms over any nominal signature.]

Ingredients of the proof

- Direct from operational semantics, rather than via denotational model.
- Uses equivariant versions of standard techniques (such as Howe's method for proving congruence of Mason-Talcott style ciu-equivalence).

Ingredients of the proof

- "Extensionality" for contextual equivalence at atom-binding types $\tau\,\mathtt{bnd}$, mirroring key property of $=_\alpha$:

$$\frac{t\{a''/a\} =_\alpha t'\{a''/a'\}}{\lambda a.t =_\alpha \lambda a'.t'}\ a'' \notin fv(a, t, a', t')$$

  Bottom-up direction fails for higher types τ unless observations on atoms are insensitive to adding extra atoms at start. ($\mathtt{lt}$ OK, $\mathtt{ord}$ not OK.)

Conclusions, further directions

- The main result is only about data correctness. What about program correctness?

  E.g. want

$$\begin{aligned}\mathtt{sub}\, x\, y\, y' = \mathtt{match}\ y'\ \mathtt{with}\ & \mathtt{V}\, x' \longrightarrow \mathtt{if}\ x = x'\ \mathtt{then}\ y\ \mathtt{else}\ y' \\ \mid\ & \mathtt{L}(\text{«}x'\text{»}z) \longrightarrow \mathtt{L}(\mathtt{bind}\, x'(\mathtt{sub}\, x\, y\, z)) \\ \mid\ & \mathtt{A}(z, z') \longrightarrow \mathtt{A}(\mathtt{sub}\, x\, y\, z, \mathtt{sub}\, x\, y\, z')\end{aligned}$$

  to satisfy that $\mathtt{sub}\, a \ulcorner t \urcorner \ulcorner t' \urcorner$ and $\ulcorner t'[t/a] \urcorner$ are always contextually equivalent. Some obs on atoms will break this.

Conclusions, further directions

- The main result is only about data correctness. What about program correctness?
- Instead of extra observations on atoms, add abstract types of finite maps on atoms.
- What about pure FreshML (Pottier, 2006)?