Generative Unbinding of Names

Andrew Pitts and Mark Shinwell
University of Cambridge and 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

→ 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

$\text{FreshML} = \text{ML} + \text{features for manipulating syntax mod } \alpha\text{-equiv.}$

**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

```
type atm

val fresh : unit → atm

val bind : atm * α → α bnd

val unbind : α bnd → atm * α

val (=) : atm → atm → bool
```

Closed values $a : atm$ 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:

\[ \langle \bar{a}, \text{fresh()} \rangle \longrightarrow \langle a' :: \bar{a}, a' \rangle \]

where \( a' \notin \bar{a} \)
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
```

`fresh()` creates a fresh atom:

\[ \langle \bar{a}, \text{fresh()} \rangle \rightarrow \langle a' :: \bar{a}, a' \rangle \]

where \( a' \notin \bar{a} \)

**current state = finite list of distinct atoms created so far**
FreshML sig for name-binding

```ocaml
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
```

Closed values «a»v of type τ bnd are represented by pairs consisting of an atom a and a closed value v : τ, created by evaluating 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:

```
<\vec{a}, unbind(«a»v)> → <a' :: \vec{a}, (a', v{a'/a})>
```

where \(a' \notin \vec{a}\).
FreshML sig for name-binding

```ocaml
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*:

\[
\langle \bar{a}, \text{unbind}(\langle a \rangle v) \rangle \longrightarrow \langle a' :: \bar{a}, (a', v\{a'/a\}) \rangle
\]

where \( a' \notin \bar{a} \).

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 \( \lambda \)-calculus (all terms, open or closed, with variables given by atoms)

can be represented in FreshML by

\[ \tau = V \text{ of } \text{atm} \]
\[ \mid L \text{ of } \tau \text{ bnd} \]
\[ \mid A \text{ of } \tau \ast \tau \]

(More generally, the representation works the same way for terms over any nominal signature [Urban-Gabbay-AMP].)
\[ \tau = V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \star \tau \]

\(\lambda\)-terms \(t\) map onto closed values \(\llbracket t \rrbracket : \tau\)

\[
\begin{align*}
\llbracket a \rrbracket & \triangleq V \ a \\
\llbracket \lambda a. t \rrbracket & \triangleq L(\llbracket a \rrbracket, \llbracket t \rrbracket) \\
\llbracket t_1 \ t_2 \rrbracket & \triangleq A(\llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket)
\end{align*}
\]
\( \tau \equiv V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \ast \tau \)

\( \lambda \)-terms \( t \) map onto closed values \( \llbracket t \rrbracket : \tau \)

and syntax-manipulating functions can be coded nicely. E.g. capture-avoiding substitution, \( t'[t/a] \), is given by \( \text{sub } a \llbracket t \rrbracket \llbracket t' \rrbracket \), where

\[
\text{sub } x \ y \ y' = \text{match } y' \text{ with}
\]

\[
\begin{align*}
&V \ x' \rightarrow \text{if } x = x' \text{ then } y \text{ else } y' \\
&| L(\langle x' \rangle z) \rightarrow L(\text{bind } x'(\text{sub } x \ y \ z)) \\
&| A(z, z') \rightarrow A(\text{sub } x \ y \ z, \text{sub } x \ y \ z')
\end{align*}
\]
\( \tau = V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \times \tau \)

\(\lambda\)-terms \(t\) map onto closed values \(\llbracket t \rrbracket : \tau\) and syntax-manipulating functions can be coded nicely. E.g. capture-avoiding substitution, \(t'[t/a]\), is given by \(\text{sub } a \llbracket t \rrbracket \llbracket t' \rrbracket\), where

\[
\text{sub } x \ y \ y' = \text{match } y' \text{ with }
\begin{align*}
V \ x' & \rightarrow \text{if } x = x' \text{ then } y \text{ else } y' \\
| L(\langle x' \rangle z) & \rightarrow L(\text{bind } x'(\text{sub } x \ y \ z)) \\
| A(z, z') & \rightarrow A(\text{sub } x \ y \ z, \text{sub } x \ y \ z')
\end{align*}
\]

This kind of pattern-match desugars to a use of \texttt{unbind}.
\[ \tau = V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \ast \tau \]

\( \lambda \)-terms \( t \) map onto closed values \( \llbracket t \rrbracket : \tau \)

and syntax-manipulating functions can be coded nicely. E.g. capture-avoiding substitution, \( t'[t/a] \), is given by \( \text{sub } a \llbracket t \rrbracket \llbracket t' \rrbracket \), where

\[
\text{sub } x y y' = \text{match } y' \text{ with } \\
V x' \rightarrow \text{if } x = x' \text{ then } y \text{ else } y' \\
| L z \rightarrow \text{let}(x', z') = \text{unbind } z \text{ in } \\
\quad L(\text{bind } x'(\text{sub } x y z')) \\
| A(z, z') \rightarrow A(\text{sub } x y z, \text{sub } x y z')
\]
\[ \tau = V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \ast \tau \]

\(\lambda\)-terms \(t\) map onto closed values \(\llbracket t \rrbracket : \tau\)

and syntax-manipulating functions can be coded nicely.
E.g. capture-avoiding substitution, \(t'[t/a]\), is given by \(\text{sub} a \llbracket t \rrbracket \llbracket t' \rrbracket\), where

\[
\text{sub } x y y' = \text{match } y' \text{ with } \\
V x' \rightarrow \text{if } x = x' \text{ then } y \text{ else } y' \\
| L z \rightarrow \text{let}(x', z') = \text{unbind } z \text{ in } \\
\quad \text{L(bind } x'(\text{sub } x y z')) \\
| A(z, z') \rightarrow A(\text{sub } x y z, \text{sub } x y z')
\]

E.g. corresponding to \((\lambda b. a)[b/a] = \lambda c. b\), have:

\[ \langle [a, b], \text{sub } a (V b) (L(\langle b \rangle(V a))) \rangle \rightarrow^* \langle [a, b, c], L(\langle c \rangle(V b)) \rangle \]
\[
\tau = V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \ast \tau
\]

\(\lambda\)-terms \(t\) map onto closed values \(\llceil t \rrceil : \tau\)

Want:

**Correctness of Representation**: two \(\lambda\)-terms are \(\alpha\)-equivalent, \(t_1 =_\alpha t_2\), iff \(\llceil t_1 \rrceil\) and \(\llceil t_2 \rrceil\) are contextually equivalent closed values of type \(\tau\).
\[ \tau = V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \ast \tau \]

\( \lambda \)-terms \( t \) map onto closed values \( \llbracket t \rrbracket : \tau \)

Want:

**Correctness of Representation**: two \( \lambda \)-terms are \( \alpha \)-equivalent, \( t_1 \equiv_\alpha t_2 \), iff \( \llbracket t_1 \rrbracket \) and \( \llbracket t_2 \rrbracket \) are contextually equivalent closed values of type \( \tau \).

i.e. can be used interchangeably in any well-typed FreshML program without affecting the observable results of program execution
\[ \tau = V \text{ of atm} \mid L \text{ of } \tau \text{ bnd} \mid A \text{ of } \tau \ast \tau \]

\( \lambda \)-terms \( t \) map onto closed values \( \llbracket t \rrbracket : \tau \)

Want:

**Correctness of Representation**: two \( \lambda \)-terms are \( \alpha \)-equivalent, \( t_1 \equiv_\alpha t_2 \), iff \( \llbracket t_1 \rrbracket \) and \( \llbracket t_2 \rrbracket \) are contextually equivalent closed values of type \( \tau \).

Proved for FreshML in Shinwell’s thesis (with a denotational semantics based on Gabbay-AMP “FM-sets”).
Algorithms involving atoms

FreshML only has \((\equiv): \text{atm} \rightarrow \text{atm} \rightarrow \text{bool}\).

Do other relations on atoms mess up the Correctness Property?
E.g. is it possible to have linearly ordered atoms, \((<): \text{atm} \rightarrow \text{atm} \rightarrow \text{bool}\)?

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

FreshML only has $\ (==) : \text{atm} \rightarrow \text{atm} \rightarrow \text{bool}$.  

Do other relations on atoms mess up the Correctness Property?

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

**Apparent problem**: proof of Correctness relies on equivariance $= \text{invariance under atom-permutations}$.

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

FreshML only has \((=)\): \(\text{atm} \rightarrow \text{atm} \rightarrow \text{bool}\).

Do other relations on atoms mess up the Correctness Property?

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

**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

\[ \text{obs} : \text{atm} \ast \cdots \ast \text{atm} \rightarrow \text{int} \]

with \textit{state-dependent} dynamics

\[ \langle \vec{a}, \text{obs}(a_1, \ldots, a_k) \rangle \rightarrow \langle \vec{a}, [\text{obs}]_{\vec{a}}(a_1, \ldots, a_k) \rangle \]
Observations on atoms

Extend FreshML with primitive functions

\[\text{obs} : \text{atm} \times \cdots \times \text{atm} \rightarrow \text{int} \]

with state-dependent dynamics

\[\langle \tilde{a}, \text{obs}(a_1, \ldots, a_k) \rangle \rightarrow \langle \tilde{a}, \llbracket \text{obs} \rrbracket_{\tilde{a}}(a_1, \ldots, a_k) \rangle\]

integer-valued function of \(\tilde{a}\) and \(a_1, \ldots, a_k \in \tilde{a}\)
that is equivariant.
E.g. \(\llbracket \text{obs} \rrbracket_{[a,b,c]}(a, c) = \llbracket \text{obs} \rrbracket_{[b,c,a]}(b, a)\)

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

Equality $\llbracket \text{eq} \rrbracket \vec{a}(a, a') \triangleq \begin{cases} 1 & \text{if } a = a', \\ 0 & \text{otherwise}. \end{cases}$

Linear order $\llbracket \text{lt} \rrbracket \vec{a}(a, a') \triangleq \begin{cases} 1 & \text{if } a \text{ occurs to the left of } a' \text{ in the list } \vec{a}, \\ 0 & \text{otherwise}. \end{cases}$

Ordinal $\llbracket \text{ord} \rrbracket \vec{a}(a) \triangleq n$, if $a$ is the $n$th element of the list $\vec{a}$.

Non-example: $\llbracket \text{bad} \rrbracket \vec{a}(a) = \alpha^{-1}(a)$, where $\alpha : \mathbb{N} \cong A$ is some fixed enumeration of the set of atoms.
Main result of the paper

**Theorem.** The Correctness of Representation property

for all \( \lambda \)-terms \( t_1, t_2 \), it is the case that

\( t_1 \equiv_\alpha t_2 \) iff \( \llbracket t_1 \rrbracket \) and \( \llbracket t_2 \rrbracket \) are

contextually equivalent

holds no matter what (equivariant) observations on

atoms we add to FreshML.

[Stated for \( \lambda \)-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 $\tau \text{bnd}$, mirroring key property of $=_\alpha$:

$$
\begin{align*}
&t\{a''/a\} =_\alpha t'\{a''/a'\} \\
&\lambda a. t =_\alpha \lambda a'. t' \\
&\text{if } a'' \in \text{fv}(a, t, a', t')
\end{align*}
$$

Bottom-up direction fails for higher types $\tau$ unless observations on atoms are insensitive to adding extra atoms at start. (It OK, ord not OK.)
Conclusions, further directions

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

E.g. want

\[
\text{sub } x \ y \ y' = \text{match } y'\text{ with}
\]

\[
\text{V } x' \rightarrow \text{if } x = x' \text{ then } y \text{ else } y'
\]

\[
\mid \text{L}((«x'»)z) \rightarrow \text{L}(\text{bind } x'((\text{sub } x \ y \ z)))
\]

\[
\mid \text{A} (z, z') \rightarrow \text{A} ((\text{sub } x \ y \ z), (\text{sub } x \ y \ z'))
\]

to satisfy that \(\text{sub } a {\overline{t'}}{\overline{t}}\) and \(\overline{t'}[\overline{t}/a]\) 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)?