On a Monadic Semantics for Freshness

Mark Shinwell and Andrew Pitts
University of Cambridge Computer Laboratory

Introduction

- In this talk we shall see a new variety of domain theory called FM-domain theory.
- FM-domain theory is good for modelling programming languages which involve:
  - α-convertible names and binding operations;
  - dynamic allocation of names at runtime.
- FM-domain theory is similar to classical domain theory, enabling many standard constructions to be utilised without too much difficulty.
- We shall see some details of the theory and examine how some language constructs can be modelled within it.

Traditional approach – 1

- Let's quickly review a traditional, denotational approach to modelling name generation.
- We define domains—sets with some additional structure—and use these to model the types of our language.
- Types τ correspond to domains $[\![\tau]\!]$.
- Values of type τ correspond to non-$\perp$ elements $d \in [\![\tau]\!]$.
- Computations of type τ correspond to elements $d \in \mathsf{T}[\![\tau]\!]$, where T is a dynamic allocation monad.
- The underlying set of such a monad might consist of (value, name-set) pairs, corresponding to the result of a computation and the names which it has created.

Traditional approach – 2

- We define typing judgements $\Gamma \vdash e : \tau$. Denotations of typing contexts are given as follows:

$$[\![\Gamma]\!] \stackrel{\text{def}}{=} \bigotimes_{x \in \text{dom}(\Gamma)} [\![\Gamma(x)]\!].$$

- Then, denotations of canonical forms in context are given by functions

$$\mathcal{V}[\![\Gamma \vdash v : \tau]\!] : [\![\Gamma]\!] \multimap [\![\tau]\!]$$

and denotations of expressions in context are given by functions

$$\mathcal{E}[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \multimap \mathsf{T}[\![\tau]\!].$$

Issues concerning T

- Great care must be taken when defining the monad T.
- For one thing, suppose one computation gives the result $\lambda x \,.\, x$ (having generated the name x) and another gives the result $\lambda y \,.\, y$ (having generated the name $y \neq x$).
  - We likely want such elements to be equal in $\mathsf{T}[\![\tau]\!]$.
- Identities such as the following are also desirable:
  - $\texttt{let } x \texttt{ = fresh in } e \quad = \quad e$ if x not free in e;
  - $\texttt{let } x \texttt{ = fresh in let } y \texttt{ = fresh in } e \quad =$
    $\texttt{let } y \texttt{ = fresh in let } x \texttt{ = fresh in } e$.
- This may all sound obvious, but work has found that quotienting the underlying set of T to arrange for all of this is both hard and troublesome.

Forgetting about name-sets

- We now consider some operational insights which can simplify matters.
- Various work, both on FreshML and the nu-calculus, has lead to the belief that it is easier to prove operational equivalence results by:
  1. discarding the idea of name-sets completely;
  2. concentrating on the sole notion of termination, rather than examining *which particular* names have been created during evaluation.
- This leads to the use of:
  - FM-domains;
  - frame stack semantics.

FM-cppos – 1

- We work with respect to a countably infinite set $\mathbb{A}$ of atoms. These will be the 'semantic objects' representing names.
- An FM-cpo D is a set where each member has a finite support, equipped with:
  - a hereditarial atom-swapping action on its members, written $(a\ a') \cdot d$ for a, $a' \in \mathbb{A}$ and $d \in D$;
  - an order relation $\sqsubseteq$ which is:
    - reflexive;
    - anti-symmetric;
    - transitive;
    - closed under swapping: $d \sqsubseteq d' \Leftrightarrow (a\ a') \cdot d \sqsubseteq (a\ a') \cdot d'$;
  - least upper bounds of all finitely-supported chains.
- An FM-cppo is an FM-cpo with a least element $\perp$.

FM-cppos – 2

- Every element d of an FM-cpo has a finite support, written $\mathsf{supp}(d)$, which is defined in terms of the swapping action.
- Informally, when d represents some piece of syntax e then the support of d corresponds to the free names of e.
- For atoms a, $a' \in \mathbb{A} \setminus \mathsf{supp}(d)$, then $(a\ a') \cdot d = d$.
- The finite support property is both crucial and useful when developing a semantics. In particular,

  > given a member d of an FM-cppo, we can *always* pick an atom a which is fresh for d.

- We may choose *any* name $a \in \mathbb{A} \setminus \mathsf{supp}(d)$ for this purpose.

α-equivalence

- We've seen that swapping has connections with fresh names. It also has connections with α-equivalence.
- For example, using the λ-calculus as an example we can precisely capture α-equivalence with the following set of rules:

$$\frac{x \in \mathsf{VId}}{x \equiv_\alpha x} \qquad \frac{t_1 \equiv_\alpha t_1' \qquad t_2 \equiv_\alpha t_2'}{t_1\ t_2 \equiv_\alpha t_1'\ t_2'}$$

$$\frac{(x\ x'') \cdot t \equiv_\alpha (x'\ x'') \cdot t' \qquad x'' \in \mathsf{VId} \setminus \mathsf{supp}(x, x', t, t')}{\lambda x\ .\ t \equiv_\alpha \lambda x'\ .\ t'}$$

- This hints that swapping can be used to provide a syntax-independent notion of name binding.

Modelling name abstraction

- Given an FM-cpo D then we can form the abstraction FM-cpo, $[\mathbb{A}]D$. The elements are atom-abstractions, written $[a]\,d$ (where $a \in \mathbb{A}$ and $d \in D$).
- When would we expect $[a_1]\,d_1$ and $[a_2]\,d_2$ to be equal?
- This is analogous to asking the question: When would we expect $\lambda x_1\,.\,t_1$ and $\lambda x_2\,.\,t_2$ to be α-equivalent?
- Following our rule for α-equivalence of λ-terms, we have that:

$$[a_1]\,d_1 = [a_2]\,d_2 \Leftrightarrow (a_1\ a)\cdot d_1 = (a_2\ a)\cdot d_2$$

where a is any atom not in the support of a_1, a_2, d_1 or d_2.

Recursive equations on FM-cppos

- A key technique in denotational semantics is the use of recursively-defined domains to denote values of recursive datatypes.
- In FM-domain theory we can construct a domain D to denote terms of the λ-calculus *up to α-equivalence* by solving the following equation:

$$D \cong \mathbb{A} \oplus ([\mathbb{A}]D) \oplus (D \otimes D).$$

- Solutions to such equations (and more complicated ones involving constructors such as $\multimap$ and $\rightarrow$) are constructed using the standard method of chains of e-p pairs.
- The solution of the equation is the domain D and an isomorphism which is canonical in a certain sense.

Frame stack semantics

- A frame stack consists of a list of frames, each of which is a basic evaluation context.
- Frame stacks are typed: $\Gamma \vdash_s S : \tau \multimap ?$.
- The evaluation of expressions is characterised by defining a termination relation. We write $\langle S, e \rangle\!\downarrow$ if the expression e terminates in the frame stack S.
- The termination relation is defined inductively on the structures of S and e.
- A theorem provides the link between the usual big-step semantics and the frame stack semantics.

Example derivation

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle [\,], x[(a_1\texttt{,}\ a_2)/x]\rangle{\downarrow}}}{\langle \texttt{let}\ x\ \texttt{=}\ [-]\ \texttt{in}\ x, (a_1\texttt{,}\ a_2)\rangle{\downarrow}}}{\langle \texttt{let}\ x\ \texttt{=}\ [-]\ \texttt{in}\ x \circ (a_1\texttt{,}\ [-]), a_2\rangle{\downarrow}} \qquad a_2 \in \mathbb{A} \setminus \mathsf{supp}(S_2)}{\langle S_2 \stackrel{\text{def}}{=} \texttt{let}\ x\ \texttt{=}\ [-]\ \texttt{in}\ x \circ (a_1\texttt{,}\ [-]), \texttt{fresh}\rangle{\downarrow}}}{\langle \texttt{let}\ x\ \texttt{=}\ [-]\ \texttt{in}\ x \circ ([-]\texttt{,}\ \texttt{fresh}), a_1\rangle{\downarrow}} \qquad a_1 \in \mathbb{A} \setminus \mathsf{supp}(S_1)}{\langle S_1 \stackrel{\text{def}}{=} \texttt{let}\ x\ \texttt{=}\ [-]\ \texttt{in}\ x \circ ([-]\texttt{,}\ \texttt{fresh}), \texttt{fresh}\rangle{\downarrow}}}{\langle \texttt{let}\ x\ \texttt{=}\ [-]\ \texttt{in}\ x, (\texttt{fresh}\texttt{,}\ \texttt{fresh})\rangle{\downarrow}}}{\langle [\,], \texttt{let}\ x\ \texttt{=}\ (\texttt{fresh}\texttt{,}\ \texttt{fresh})\ \texttt{in}\ x\rangle{\downarrow}}
$$

Denotational semantics with stacks

- The denotation of values is as before:

$$\mathcal{V}[\![\Gamma \vdash v : \tau]\!] : [\![\Gamma]\!] \multimapdotinv\!\!\longrightarrow [\![\tau]\!].$$

- The meaning of a frame stack is determined by the value which is inserted into the hole:

$$\mathcal{S}[\![\Gamma \vdash_s S : \tau \multimap ?]\!] : [\![\Gamma]\!] \circ\!\!\longrightarrow ([\![\tau]\!] \circ\!\!\longrightarrow 1_\perp).$$

- The meaning of an expression is determined by the frame stack which surrounds it:

$$\mathcal{E}[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \circ\!\!\longrightarrow (([\![\tau]\!] \circ\!\!\longrightarrow 1_\perp) \circ\!\!\longrightarrow 1_\perp).$$

- Lo and behold: $([\![\tau]\!] \multimap 1_\perp) \multimap 1_\perp$ is the underlying set of a continuation monad, where the 'results' are $\top$ and $\perp$.

Modelling name generation – 1

- Suppose we have an expression $\mathtt{fresh}$ which generates a fresh name.
- The termination behaviour might be:

$$\frac{\langle S, a\rangle\!\downarrow}{\langle S, \mathtt{fresh}\rangle\!\downarrow} \quad \text{for some/any } a \notin \mathsf{supp}(S).$$

- The stack S must be of type $\mathtt{name} \multimap ?$, where $\mathtt{name}$ is some type of names and $[\![\mathtt{name}]\!] \stackrel{\text{def}}{=} \mathbb{A}_\perp$.
- The denotation of S therefore lies in the FM-cppo $[\![\mathtt{name}]\!]^\perp$, which we recall is $[\![\mathtt{name}]\!] \multimap 1_\perp$.
- The denotation of $\mathtt{fresh}$ lies in $[\![\mathtt{name}]\!]^\perp \multimap 1_\perp$.

Modelling name generation – 2

- We model name generation using a morphism

$$\mathbf{new} : [\![\mathtt{name}]\!]^{\perp} \multimap 1_{\perp}$$

such that, for some/any $a \notin \mathsf{supp}(\sigma)$,

$$\mathbf{new}(\sigma \in [\![\mathtt{name}]\!]^{\perp}) \stackrel{\text{def}}{=} \sigma(a).$$

- Intuitively:
  - You give $\mathbf{new}$ a frame stack.
  - It fills the hole with a fresh atom.
  - Evaluation then continues.
- Note that the computation encoded in the stack cannot depend on which particular atom was chosen.

An application

- We have used FM-domain theory and frame stack semantics to produce formal correctness results for a small language Mini-FreshML.
- Mini-FreshML provides for:
  - runtime generation of fresh names;
  - manipulation of abstraction values, to represent object language name-binding;
  - recursive datatype declarations involving abstraction types, to model object language syntax trees with binding.
- In some ways, Mini-FreshML is the ‘natural’ language which arises from FM-domain theory.

Computational adequacy

- These computational adequacy results establish links between the operational and denotational models.
- For closed frame stacks S and closed expressions e,

$$\langle S, e \rangle{\downarrow} \Leftrightarrow [\![e]\!][\![S]\!] = \top.$$

- For closed frame stacks S and closed canonical forms v,

$$\langle S, v \rangle{\downarrow} \Leftrightarrow [\![S]\!][\![v]\!] = \top.$$

- These results are not straightforward to prove:
  - nearly 80 cases to check in inductive proofs;
  - various tricky constructions on relations.

Operational equivalence

- In the setting of frame stack semantics, it is natural to use the relation of CIU-equivalence,

$$\Gamma \vdash e \approx_{\text{ciu}} e' : \tau$$

which holds iff $\Gamma \vdash e : \tau$, $\Gamma \vdash e' : \tau$ and for all frame stacks S and closing substitutions ψ,

$$\langle S, e[\psi] \rangle\!\downarrow \Leftrightarrow \langle S, e'[\psi] \rangle\!\downarrow .$$

- CIU-equivalence in Mini-FreshML coincides with:
  - a certain logical relation used in the proof of adequacy;
  - the traditional notion of contextual equivalence.

Correctness results

- We are interested in proving results concerning a Mini-FreshML data type such as:

  $$\texttt{type}\ \delta\ \texttt{=}\ \texttt{Var of name}\ \texttt{|}\ \texttt{Lam of}\ \langle\texttt{name}\rangle\delta\ \texttt{|}\ \texttt{App of}\ \delta \times \delta.$$

- λ-terms t can be translated into expressions $[t]_{\mathrm{e}}$ of type δ.
- *Theorem.* For any λ-terms t and t', with free variables in the set $\{x_0, \ldots, x_n\}$,

  $$t \equiv_\alpha t' \Leftrightarrow \{x_0 : \texttt{name}, \cdots, x_n : \texttt{name}\} \vdash [t]_{\mathrm{e}} \approx_{\mathsf{ctx}} [t']_{\mathrm{e}} : \delta. \qquad \square$$

Thus object-level α-equivalence coincides with meta-level contextual equivalence.

Conclusions

- There is growing evidence that FM-domain theory forms a very good foundation for reasoning about names.
  - Correctness results for Mini-FreshML (Pitts and Shinwell, 2004).
  - Denotational semantics for MetaML (Pitts and Sheard, 2004).
  - Full abstraction for the nu-calculus (Stark et al., 2004).
- The use of the continuation monad in FM-domain theory provides an elegant model of dynamic allocation.
- It appears that the same, or a very similar monad could be used for languages involving references, exceptions, . . .
- To see the 'freshness' features of Mini-FreshML in action, please try out Fresh O'Caml:

  `http://www.freshml.org/foc/`