Realizability Semantics of Parametric Polymorphism, General References, and Recursive Types

Lars Birkedal

IT University of Copenhagen
Joint work with Kristian Støvring and Jacob Thamsborg

Oct, 2008

Introduction

Two lines of motivation:

- Logics for weak (ML style) references
- Data abstraction qua relational parametricity for languages with effects

Let's briefly discuss both.

- Long version of paper can be found at `www.itu.dk/people/kss/papers/poly-ref-rec.pdf`.

Intro I: Logics for weak refs

- All of the logics mentioned in earlier talk on HOSL (Ideal. ML, HTT, Java, higher-order store) do not take advantage of type safety.
- For references, we reason as if locations are just natural numbers
  - E.g., in HTT we have a typed points-to predicate $x \mapsto_\tau e$, which in Ynot formalization means additional proof obligations
  - E.g., for lookup we need to prove precondition that the pointer points to something, even if the language cannot have "null-pointers" (Ideal. ML, e.g.)
- Wish to combine separation logic with equational reasoning, such as:
  - beta and eta rules for the pure part of the language
  - Plotkin-Power axioms for state

Intro II: Relational Parametricity

- Reynolds 1983: to show equivalence of polymorphic programs and to show representation independence for abstract data types. Setting: λ_2.
- Abadi and Plotkin: logic for parametricity, universal properties of definable types [LB + Møgelberg: categorical models for such]
- Towards relational parametricity for languages with effects:
- I: Equational type theories with effects:
  - Plotkin: linear λ_2 + fixed points, universal properties of recursive types
  - LB + Møgelberg: LAPL + categorical models of such
  - Recent work by Simpson, Møgelberg on general polymorphic type theory for effects and Hasegawa on continuations, related to Paul Levy's CBPV

Relational Parametricity, II

- II: Programming languages with effects
  - Wadler
  - equality = contextual equivalence
  - much research on devising reasoning methods for ctx. equiv. using both logical relations and bisimulation techniques; for state: Pitts-Stark, Benton-Leperchey, LB-Bohr, Koutavas-Wand, Støvring-Lassen, ...
  - relationally parametric models for languages with recursion and inductive/co-inductive types [Pitts, Bierman et. al., Johann and Voigtlaender] and recursive types [Appel et. al.]
- Link between the two approaches: Møgelberg used type theories to give adequate semantics for FPC.
- This talk:
  - relational parametric model for prog. lang. with recursive types and general references.
  - focus on challenge of defining adequate semantics, existence of logical relations
  - future work: combine with LB-Bohr to get better reasoning methods for local state

Outline — Types

Slogan: one domain equation for each of $\forall$, ref, μ.

- $\forall$ impredicative polymorphism: choose to model types as relations *UARel*(V) over a recursively defined predomain V.
- ref general references with dynamic allocation: use Kripke model with recursively defined worlds, approximately of the form:

$$
\begin{aligned}
\mathcal{W} &= \mathbb{N}_0 \to \mathcal{T} \\
\mathcal{T} &= \mathcal{W} \to \mathit{UARel}(V)
\end{aligned}
$$

  Solve in CBUlt.
- μ recursive types: relations interpreting types also recursively defined,
  - non-trivial for reference types, leads to novel modeling of locations involving some approximation information.

Outline — Terms

- Use V to give an "untyped" semantics of terms.
- For well-typed terms: prove the fundamental theorem of logical relations with respect to the relational interpretation of types, to get a typed interpretation.
- In earlier work, shown adequacy of such a denotational semantics wrt. operational semantics:
  - Hence get proof method for proving contextual equivalence of programs.
  - In particular, data abstraction results qua parametricity in a language with general references.

Uniform cpos

A *uniform cpo* $(A, (\varpi_n)_{n\in\omega})$ is a cpo A together with a family $(\varpi_n)_{n\in\omega}$ of continuous functions from A to $A_\perp$, satisfying

$$\varpi_0 \sqsubseteq \varpi_1 \sqsubseteq \cdots \sqsubseteq \varpi_n \sqsubseteq \ldots$$

$$\bigsqcup_{n\in\omega} \varpi_n = \overline{id_A} = \lambda a.\lfloor a \rfloor$$

$$\varpi_m \mathbin{\bar{\circ}} \varpi_n = \varpi_n \mathbin{\bar{\circ}} \varpi_m = \varpi_{\min(m,n)}$$

$$\varpi_0 = \lambda e.\perp.$$

Predomain V of values

Proposition. There exists a uniform cpo $(V, (\pi_n)_{n\in\omega})$ satisfying:
In pCpo:

$$V \;\cong\; \mathbb{Z} + Loc + 1 + (V \times V) + (V + V) + V + TV + (V \to TV) \quad (1)$$

where

$$\begin{aligned} TV &= (V \to S \to Ans) \to S \to Ans \\ S &= \mathbb{N}_0 \rightharpoonup_{fin} V \\ Ans &= (\mathbb{Z} + Err)_\perp \end{aligned}$$

and

$$\begin{aligned} Loc &= \mathbb{N}_0 \times \overline{\omega} \\ Err &= 1\,. \end{aligned}$$

The functions $\pi_n : V \to V_\perp$ satisfy (and are determined by)

$$\begin{aligned}
\pi_0 &= \lambda v.\perp \\
\pi_{n+1}(in_{\mathbb{Z}}(k)) &= \lfloor in_{\mathbb{Z}}(k) \rfloor \\
\pi_{n+1}(in_\times(v_1, v_2)) &= \begin{cases} \lfloor in_\times(v_1', v_2') \rfloor & \text{if } \pi_n\, v_1 = \lfloor v_1' \rfloor \text{ and } \pi_n\, v_2 = \lfloor v_2' \rfloor \\ \perp & \text{otherwise} \end{cases} \\
& \ldots \text{etc. as you'd expect, except:} \\
\pi_{n+1}(in_{Loc}(l, m)) &= \lfloor in_{Loc}(l, \min(n+1, m)) \rfloor
\end{aligned}$$

Untyped Semantics of Terms, I

$[\![t]\!]_X : V^X \to TV$ by induction on t:

$$[\![x]\!]_X\,\rho = \eta(\rho(x))$$

$$[\![\overline{k}]\!]_X\,\rho = \eta(in_{\mathbb{Z}}\,k)$$

$$[\![t_1 \pm t_2]\!]_X\,\rho = [\![t_1]\!]_X\,\rho \star \lambda v_1.\ [\![t_2]\!]_X\,\rho \star \lambda v_2. \begin{cases} \eta(in_{\mathbb{Z}}(k_1 \pm k_2)) & \text{if } v_1 = in_{\mathbb{Z}}\,k_1 \\ error & \text{otherwise} \end{cases}$$

$$[\![\lambda x.\,t]\!]_X\,\rho = \eta(in_{\to}(\lambda v.\ [\![t]\!]_{X,x}\,(\rho[x \mapsto v])))$$

$$[\![t_1\ t_2]\!]_X\,\rho = [\![t_1]\!]_X\,\rho \star \lambda v_1.\ [\![t_2]\!]_X\,\rho \star \lambda v_2. \begin{cases} f\ v_2 & \text{if } v_1 = in_{\to}\,f \\ error & \text{otherwise} \end{cases}$$

$$[\![\Lambda\alpha.\,t]\!]_X\,\rho = \eta(in_{\forall}\,([\![t]\!]_X\,\rho))$$

$$[\![t\ [\tau]]\!]_X\,\rho = [\![t]\!]_X\,\rho \star \lambda v. \begin{cases} c & \text{if } v = in_{\forall}\,c \\ error & \text{otherwise} \end{cases}$$

Untyped Semantics of Terms, II

- For lookup and assignment we need to consider semantic locations:

$$[\![!\, t]\!]_X\, \rho = [\![t]\!]_X\, \rho \star \lambda v.\, \mathit{lookup}\, v$$

- where $\mathit{lookup}\, v =$

$$\lambda k\, \lambda s. \begin{cases} k\, s(l)\, s & \text{if } v = \lambda_l \text{ and } l \in \mathrm{dom}(s) \\ k\, v'\, s & \text{if } v = \lambda_l^{n+1},\ l \in \mathrm{dom}(s), \text{ and } \pi_n(s(l)) = \lfloor v' \rfloor \\ \bot_{Ans} & \text{if } v = \lambda_l^{n+1},\ l \in \mathrm{dom}(s), \text{ and } \pi_n(s(l)) = \bot \\ \mathit{error}_{Ans} & \text{otherwise} \end{cases}$$

Untyped Semantics of Terms, III

- assignment:

$$[\![t_1 := t_2]\!]_X\, \rho = [\![t_1]\!]_X\, \rho \star \lambda v_1.\; [\![t_2]\!]_X\, \rho \star \lambda v_2.\, \mathit{assign}\, v_1\; v_2$$

- where $\mathit{assign}\, v_1\; v_2 = \lambda k\, \lambda s. =$

$$\begin{cases} k\,(in_1 *)\,(s[l \mapsto v_2]) & \text{if } v_1 = \lambda_l \text{ and } l \in \mathrm{dom}(s) \\ k\,(in_1 *)\,(s[l \mapsto v_2']) & \text{if } v_1 = \lambda_l^{n+1},\, l \in \mathrm{dom}(s), \text{ and } \pi_n(v_2) = \lfloor v_2' \rfloor \\ \bot_{Ans} & \text{if } v_1 = \lambda_l^{n+1},\, l \in \mathrm{dom}(s), \text{ and } \pi_n(v_2) = \bot \\ \mathit{error}_{Ans} & \text{otherwise} \end{cases}$$

Untyped Semantics of Terms, IV

Let t be a term of type int with no free term variables or type variables. The *program semantics of t* is the element $[\![t]\!]^{\mathrm{p}}$ of *Ans* defined by

$$[\![t]\!]^{\mathrm{p}} = [\![t]\!]_{\emptyset}\, \emptyset\, k_{init}\, s_{init}$$

where

$$k_{init} = \lambda v.\lambda s. \begin{cases} \lfloor \iota_1\, k \rfloor & \text{if } v = in_{\mathbb{Z}}(k) \\ error_{Ans} & \text{otherwise} \end{cases}$$

and where $s_{init} \in S$ is the empty store.

CBUlt

Recall:

- An *ultrametric space* is a metric space (D, d) that instead of triangle inequality satisfies the stronger *ultrametric inequality*:

$$d(x, z) \leq \max(d(x, y), d(y, z)).$$

- A function $f : D_1 \to D_2$ from a metric space (D_1, d_1) to a metric space (D_2, d_2) is *non-expansive* if $d_2(f(x), f(y)) \leq d_1(x, y)$ for all x and y in D_1.
- A function $f : D_1 \to D_2$ from a metric space (D_1, d_1) to a metric space (D_2, d_2) is *contractive* if there exists $\delta < 1$ such that $d_2(f(x), f(y)) \leq \delta \cdot d_1(x, y)$ for all x and y in D_1.
- CBUlt is the category with complete 1-bounded ultrametric spaces and non-expansive functions.

CBUlt, II

- CBUlt is cartesian closed; the exponential $(D_1, d_1) \to (D_2, d_2)$ is the set of non-expansive maps with the "sup"-metric $d_{D_1 \to D_2}$ as distance function:

$$d_{D_1 \to D_2}(f, g) = \sup\{d_2(f(x), g(x)) \mid x \in D_1\}\,.$$

- Solutions to recursive domain equations for locally contractive functors.
- A functor $F : \mathrm{CBUlt}^{\mathrm{op}} \times \mathrm{CBUlt} \to \mathrm{CBUlt}$ is *locally contractive* if there exists $\delta < 1$ such that

$$d(F(f, g), F(f', g')) \leq \delta \cdot \max(d(f, f'), d(g, g'))$$

for all non-expansive functions f, f', g, and g'.

$UARel(V) \in$ CBUlt

Recall [Amadio, Abadi-Plotkin]:

- $UARel(V)$ is the set of admissible relations that are *unifom*: $\varpi_n \in R \to R_\perp$, for all n.
- Such relations are determined by its elements of the form $(\varpi_n\, e, \varpi_n\, e')$.
- $UARel(V) \in$ CBUlt, distance function:

$$d(R,S) = \begin{cases} 2^{-\max\{\, n \in \omega \mid \varpi_n \in R \to S \,\wedge\, \varpi_n \in S \to R \,\}} & \text{if } R \neq S \\ 0 & \text{if } R = S. \end{cases}$$

Worlds

- **Proposition.** Let $(D, d) \in$ CBUlt. The set $\mathbb{N}_0 \rightharpoonup_{fin} D$ with distance function:

$$d'(\Delta, \Delta') = \begin{cases} \max\{d(\Delta(l), \Delta'(l)) \mid l \in \mathrm{dom}(\Delta)\} & \text{if } \mathrm{dom}(\Delta) = \mathrm{dom}(\\ 1 & \text{otherwise.} \end{cases}$$

  is in CBUlt.
- Extension ordering: $\Delta \leq \Delta'$ iff

$$\mathrm{dom}(\Delta) \subseteq \mathrm{dom}(\Delta') \wedge \forall l \in \mathrm{dom}(\Delta).\, \Delta(l) = \Delta'(l)\,.$$

Space of types

- **Proposition.**

$$F(D) = (\mathbb{N}_0 \rightharpoonup_{fin} D) \to_{mon} \mathit{UARel}(V)$$

(monotone, non-expansive maps) defines a functor $F : \mathit{CBUlt}^{\mathrm{op}} \to \mathit{CBUlt}$.

- **Theorem.** There exists $\widehat{\mathcal{T}} \in$ CBUlt such that the isomorphism

$$\widehat{\mathcal{T}} \cong \tfrac{1}{2}((\mathbb{N}_0 \rightharpoonup_{fin} \widehat{\mathcal{T}}) \to_{mon} \mathit{UARel}(V)) \tag{2}$$

holds in CBUlt.

Space of Types, II

Define:

- Worlds: $\mathcal{W} = \mathbb{N}_0 \rightharpoonup_{fin} \widehat{\mathcal{T}}$
- Types: $\mathcal{T} = \mathcal{W} \to_{mon} \mathit{UARel}(V)$
- Computations: $\mathcal{T}_T = \mathcal{W} \to_{mon} \mathit{UARel}(TV)$
- Continuations: $\mathcal{T}_K = \mathcal{W} \to_{mon} \mathit{UARel}(K)$
- States: $\mathcal{T}_S = \mathcal{W} \to \mathit{UARel}(S)$ (note: not monotone)

Semantics of Types

For every $\Xi \vdash \tau$, define the non-expansive $[\![\tau]\!]_{\Xi} : \mathcal{T}^{\Xi} \to \mathcal{T}$ by induction on τ:

$$
\begin{aligned}
[\![\alpha]\!]_{\Xi}\,\varphi &= \varphi(\alpha) \\
[\![\mathsf{int}]\!]_{\Xi}\,\varphi &= \lambda\Delta.\,\{\,(in_{\mathbb{Z}}\,k, in_{\mathbb{Z}}\,k) \mid k \in \mathbb{Z}\,\} \\
[\![1]\!]_{\Xi}\,\varphi &= \lambda\Delta.\,\{\,(in_1\,*, in_1\,*)\,\} \\
[\![\tau_1 \times \tau_2]\!]_{\Xi}\,\varphi &= [\![\tau_1]\!]_{\Xi}\,\varphi \times [\![\tau_2]\!]_{\Xi}\,\varphi \\
[\![0]\!]_{\Xi}\,\varphi &= \lambda\Delta.\,\emptyset \\
[\![\tau_1 + \tau_2]\!]_{\Xi}\,\varphi &= [\![\tau_1]\!]_{\Xi}\,\varphi + [\![\tau_2]\!]_{\Xi}\,\varphi \\
[\![\mathsf{ref}\,\tau]\!]_{\Xi}\,\varphi &= \mathit{ref}([\![\tau]\!]_{\Xi}\,\varphi) \\
[\![\forall\alpha.\tau]\!]_{\Xi}\,\varphi &= \lambda\Delta.\,\{\,(in_{\forall}\,c, in_{\forall}\,c') \mid \forall\nu \in \mathcal{T}.\,(c, c') \in \\
&= \qquad\qquad \mathit{comp}([\![\tau]\!]_{\Xi,\alpha}\,\varphi[\alpha \mapsto \nu])(\Delta)\,\} \\
[\![\mu\alpha.\tau]\!]_{\Xi}\,\varphi &= \mathit{fix}\left(\lambda\nu.\,\lambda\Delta.\,\{\,(in_{\mu}\,v, in_{\mu}\,v') \mid (v, v') \in [\![\tau]\!]_{\Xi,\alpha}\,\varphi[\alpha \mapsto \nu]\,\Delta\,\}\right) \\
[\![\tau_1 \to \tau_2]\!]_{\Xi}\,\varphi &= ([\![\tau_1]\!]_{\Xi}\,\varphi) \to (\mathit{comp}([\![\tau_2]\!]_{\Xi}\,\varphi))
\end{aligned}
$$

Semantic Type Constructors

$$
\begin{aligned}
(\nu_1 \times \nu_2)(\Delta) = \{\, & (in_\times(v_1, v_2), in_\times(v_1', v_2')) \mid \\
& (v_1, v_1') \in \nu_1(\Delta) \wedge (v_2, v_2') \in \nu_2(\Delta) \,\}
\end{aligned}
$$

$$
\begin{aligned}
ref(\nu)(\Delta) = \{\, & (\lambda_l, \lambda_l) \mid l \in \mathrm{dom}(\Delta) \wedge \\
& \forall \Delta_1 \geq \Delta.\ App\,(\Delta(l))\,\Delta_1 = \nu(\Delta_1) \,\} \\
\cup \{\, & (\lambda_l^{n+1}, \lambda_l^{n+1}) \mid l \in \mathrm{dom}(\Delta) \wedge \\
& \forall \Delta_1 \geq \Delta.\ App\,(\Delta(l))\,\Delta_1 \stackrel{n}{=} \nu(\Delta_1) \,\}
\end{aligned}
$$

- Note the use of semantic locations to ensure non-expansiveness in *ref* case.
- Necessary: for earlier version we proved that relations did not exist if we didn't use semantic locations.
- Because of relational parametricity, we need to model *open* types; hence need to compare semantic types above, cannot simply use syntactic worlds and compare types syntactically.

Semantic Type Constructors, II

$$(\nu \to \xi)(\Delta) = \{\, (in_{\to}\, f, in_{\to}\, f') \mid \forall \Delta_1 \geq \Delta. \\ \forall (v, v') \in \nu(\Delta_1)\,.(f\,v, f'\,v') \in \xi(\Delta_1)\,\}$$

$$cont(\nu)(\Delta) = \{\, (k, k') \mid \forall \Delta_1 \geq \Delta. \forall (v, v') \in \nu(\Delta_1). \\ \forall (s, s') \in states(\Delta_1).\, (k\,v\,s, k'\,v'\,s') \in R_{Ans}\,\}$$

$$comp(\nu)(\Delta) = \{\, (c, c') \mid \forall \Delta_1 \geq \Delta. \forall (k, k') \in cont(\nu)(\Delta_1). \\ \forall (s, s') \in states(\Delta_1).\, (c\,k\,s, c'\,k'\,s') \in R_{Ans}\,\}$$

$$states(\Delta) = \{\, (s, s') \mid \mathrm{dom}(s) = \mathrm{dom}(s') = \mathrm{dom}(\Delta) \\ \wedge\ \forall l \in \mathrm{dom}(\Delta).\, (s(l), s'(l)) \in App\,(\Delta(l))\,(\Delta)\,\}$$

$$R_{Ans} = \{\, (\bot, \bot)\,\} \cup \{\, (\lfloor \iota_1\, k \rfloor, \lfloor \iota_1\, k \rfloor) \mid k \in \mathbb{Z}\,\}$$

Lemmas for interpreting $\forall$ and μ

- **Lemma.** Let τ and τ' be types such that $\Xi, \alpha \vdash \tau$ and $\Xi \vdash \tau'$. For all φ in $\mathcal{T}^{\Xi}$,

$$[\![\tau[\tau'/\alpha]]\!]_{\Xi}\,\varphi = [\![\tau]\!]_{\Xi,\alpha}\,(\varphi[\alpha \mapsto [\![\tau']\!]_{\Xi}\,\varphi])\,.$$

- **Corollary.** For $\Xi, \alpha \vdash \tau$ and $\varphi \in \mathcal{T}^{\Xi}$,

$$[\![\mu\alpha.\tau]\!]_{\Xi}\,\varphi = \lambda\Delta.\,\{\,(\mathit{in}_{\mu}\,v, \mathit{in}_{\mu}\,v') \mid (v, v') \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]_{\Xi}\,\varphi\,\Delta\,\}\,.$$

Typed Semantics of Terms

- For $\Xi \vdash \Gamma$ and $\varphi \in \mathcal{T}^{\Xi}$, let $[\![\Gamma]\!]_{\Xi}\,\varphi$ be the binary relation on $V^{\mathrm{dom}(\Gamma)}$ defined by

$$[\![\Gamma]\!]_{\Xi}\,\varphi = \{\, (\rho, \rho') \mid \forall x \in \mathrm{dom}(\Gamma).\, (\rho(x), \rho'(x)) \in [\![\Gamma(x)]\!]_{\Xi}\,\varphi \,\}\,.$$

- Two typed terms $\Xi \mid \Gamma \vdash t : \tau$ and $\Xi \mid \Gamma \vdash t' : \tau$ of the same type are *semantically related*, written $\Xi \mid \Gamma \models t \sim t' : \tau$, if for all $\varphi \in \mathcal{T}^{\Xi}$, all $(\rho, \rho') \in [\![\Gamma]\!]_{\Xi}\,\varphi$, and all $\Delta \in \mathcal{W}$,

$$\left([\![t]\!]_{\mathrm{dom}(\Gamma)}\,\rho,\ [\![t']\!]_{\mathrm{dom}(\Gamma)}\,\rho' \right) \in \mathit{comp}([\![\tau]\!]_{\Xi}\,\varphi)(\Delta)\,.$$

Typed Semantics of Terms, II

- **Theorem.** Semantic relatedness is a congruence.
- **Corollary.** (FTLR) If $\Xi \mid \Gamma \vdash t : \tau$, then $\Xi \mid \Gamma \models t \sim t : \tau$.
- **Corollary.** (Type Soundness) If $\emptyset \mid \emptyset \vdash t : \tau$ is a closed term of type τ, then $[\![t]\!]_\emptyset \emptyset \neq \mathit{error}$.

Simple Example: counter-module

- Type for counter-module client:

$$\tau_{\mathrm{cl}} = \forall\alpha.((1 \to \alpha) \times (\alpha \to \alpha) \times (\alpha \to \mathtt{int}) \to \mathtt{int})\,.$$

- Two implementations:

$$\begin{array}{lcl} l_1 & = & (\lambda x : 1.\,0,\ \lambda x : \mathtt{int}.\,x + 1,\ \lambda x : \mathtt{int}.\,x) \\ l_2 & = & (\lambda x : 1.\,0,\ \lambda x : \mathtt{int}.\,x - 1,\ \lambda x : \mathtt{int}.\,-x). \end{array}$$

- Can show

$$\emptyset \mid \emptyset \mid c : \tau_{\mathrm{cl}} \vdash c[\mathtt{int}]l_1 =_{\mathrm{ctx}} c[\mathtt{int}]l_2 : \mathtt{int}\,.$$

  (using adequacy of denotational semantics wrt. operational).
- Simple example, no reference types in the module implementations, but note that the client may use all features of the language, including references.

Conclusion & Future Work

Conclusion:

- Developed a realizability model of call-by-value prog. lang. with parametric polymorphism, general references, and recursive types.
  - Kripke model over a recursively defined set of worlds.
  - Introduced semantic locations to model reference types involving comparison of semantic types (as needed for modelling of syntactic *open* types, as needed for relational parametricity).

Future Work:

- Refine worlds to achieve better reasoning methods for *local* state.
- Will combine with earlier work by Bohr-Birkedal [2006], and also recent related work by Ahmed-Dreyer-Rossberg [2008].
- Formal relationship with recent step-indexed models of recursive types and state by Appel, Ahmed, et. al.
- Monadic language and program logic.