Refinement Types for Algebraic Effects

Danel Ahman Gordon Plotkin

LFCS, University of Edinburgh

TYPES 2015
Tallinn

Today's plan

$$
\begin{tikzcd}
\text{Value ref. types} \arrow[rr, bend left] \arrow[d] & & \text{Computation ref. types} \arrow[ll, bend left] \arrow[d] \\
& \text{Algebraic theories} \arrow[ur, dashed] \arrow[dr, dashed] & \\
\text{Value types} \arrow[rr, bend left] & & \text{Computation types} \arrow[ll, bend left]
\end{tikzcd}
$$

$+$ some examples

Refinement types

- For extending base language's type system
  - to allow more precise specifications in types

    $\vdash \mathsf{Odd} : \mathsf{Ref}(\mathsf{Nat}) \qquad \vdash \mathsf{Even} : \mathsf{Ref}(\mathsf{Nat})$

  - make it possible to internalize meta-theorems

    $n : \mathsf{Odd}, m : \mathsf{Odd} \vdash n + m : \mathsf{Even}$

    and also program optimizations

- In this talk, we discuss propositional ref. types
  - for example Even and Odd, as above

    [Freeman, Pfenning '91]

- Ideas also apply to FOL-based ref. types [Denney '98]
  - for example $\{x : \sigma \mid \varphi(x)\}$

  but with some additional technical challenges

Computational effects

- Ever-present in the various languages we work with
  - regardless of being lazy, strict, object-oriented, ...

- First unifying account using monads, e.g.: [Moggi '89]
  - non-determinism: $T\,X = \mathcal{P}^{+}_{\mathrm{fin}}(X)$
  - read-only memory: $T\,X = S \to X$
  - write-only memory: $T\,X = M \times X$ (M *a monoid*)
  - read-write memory / global state: $T\,X = S \to (S \times X)$

- And also more recent generalizations from TYPES '13 [Ahman, Uustalu '14]
  - update monads: $T\,X = S \to (P \times X)$ (P *a monoid*)
  - dep. typed update monads: $T\,X = \Pi s : S.(P\,s \times X)$

    (*where* $(S, \downarrow, P, \mathrm{o}, \oplus)$ *a directed container*)

Refinement types for computational effects

- Plenty of work in the literature that either
  - target particular computational effects, or
  - cover particular kinds of specifications

- For example:
  - pre- and postconditions $\{P\}\,\sigma\,\{Q\}$ for state
    - Hoare Type Theory [Nanevski et. al. '08]
    - Refined state monad in F7 [Borgström et. al. '11]
    - Dijkstra Monad in F* [Swamy et. al. '13]
  - sessions and protocols $!\mathsf{Bool}.?\mathsf{Nat}.S$ for I/O
    - trace effects [Skalka, Smith, van Horn '08]
    - session typed languages [Honda '93] [and many others]
  - effect annotations ε in type-and-effect systems
    - sets of operation symbols [Kammar, Plotkin '12]
    - ordered monoids [Katsumata '14]

Computational effects, algebraically

- Take algebraic theories as a primitive, rather than the monads they generate [Plotkin,Power '02]
  - in this talk: "standard" n-ary operations $\mathrm{op} : n$
  - not in this talk: operations with parameters and binding

- For example:
  - non-determinism: $TX = \mathcal{P}^+_{\mathrm{fin}}(X)$

    $x \text{ or } x = x$

    $x \text{ or } y = y \text{ or } x$

    $x \text{ or } (y \text{ or } z) = (x \text{ or } y) \text{ or } z$

  - state: $TX = 2 \to (2 \times X)$ (where $S = 2$)

    $\mathrm{lkp}(\mathrm{upd}_0(x), \mathrm{upd}_1(x)) = x$

    $\mathrm{upd}_i(\mathrm{upd}_j(x)) = \mathrm{upd}_j(x)$

    $\mathrm{upd}_i(\mathrm{lkp}(x_0, x_1)) = \mathrm{upd}_i(x_i)$

Effectful programs as computation trees

- Algebraic modeling of effects is somewhat eyeopening
- Immediately allows to think of programs such as

$$
\begin{aligned}
&\textbf{let } f &&= \lambda b : \mathsf{bool}\,.\,\mathsf{return}\,\neg b \textbf{ in}\\
&\textbf{let } x &&= \mathsf{lkp} \textbf{ in}\\
&\textbf{let } y &&= f\,x \textbf{ in}\\
&\textbf{let } \_ &&= \mathsf{output}\ y \textbf{ in}\\
&\textbf{let } \_ &&= \textbf{if } x = 1 \textbf{ then } \mathsf{upd}\ y \textbf{ in}\\
&\textbf{return } y
\end{aligned}
$$

as computation trees

$$
\begin{tikzcd}
& \mathsf{lkp} \arrow[dl, no head] \arrow[dr, no head] & \\
\mathsf{output}_1 \arrow[d, no head] & & \mathsf{output}_0 \arrow[d, no head] \\
1 & & \mathsf{upd}_0 \arrow[d, no head] \\
& & 0
\end{tikzcd}
$$

Ref. types for algebraic effects

- Reason about effectful programs as if they would simply be comp. trees built from operations

- Would like to build:
  - single trees from operations
  - combine them into finite and infinite sets of trees
  - with clean and finite syntax

- Define effect refinements, based on modal formulae
  - $\psi ::= [\,] \mid \langle \mathsf{op} \rangle(\psi_1, \ldots, \psi_n) \mid \bot \mid \psi_1 \vee \psi_2 \mid X \mid \mu X.\psi$

    where

  - "holes" $[\,]$ are placeholders for leaves
  - op. modalities $\langle \mathsf{op} \rangle$ are used to build trees from ops.

- Note: effect refs. are indifferent wrt. specific algebras

Ref. types for algebraic effects

- Think effect refinements as a small logic on comp. trees
  - $\psi ::= [\,] \mid \langle \mathsf{op} \rangle(\psi_1, \ldots, \psi_n) \mid \bot \mid \psi_1 \vee \psi_2 \mid X \mid \mu X.\psi$
- They also come with a satisfiability / subtyping relation

$$\Delta \vdash \psi_1 \sqsubseteq \psi_2$$

- $\sqsubseteq$ includes standard logic
- also want $\sqsubseteq$ to include algebraic properties of $\langle \mathsf{op} \rangle$'s
  - can't just include all the axioms, e.g., $\psi = \langle \mathsf{lkp} \rangle(\psi, \psi)$ [Gautam '57]
  - need to include derivable semi-linear equations

$$\frac{\vec{x} \vdash t = u \text{ derivable in } \mathcal{T}_{\mathrm{eff}} \qquad t \text{ linear in } \vec{x} \qquad \mathit{Vars}(u) \subseteq \mathit{Vars}(t) \qquad \Delta \vdash \psi_1 \quad \ldots \quad \Delta \vdash \psi_n}{\Delta \vdash t^{\bullet}[\vec{\psi}/\vec{x}] \sqsubseteq u^{\bullet}[\vec{\psi}/\vec{x}]}$$

About the semantics of effect refinements

- Recall: effect refs. are indifferent wrt. specific algebras
- Concretely, they can be interpreted as monotone maps
  - $[\![\Delta \vdash \psi]\!]_{\mathcal{A}} : \mathcal{P}(\mathsf{U}\mathcal{A}) \times [\![\Delta]\!]_{\mathcal{A}} \longrightarrow \mathcal{P}(\mathsf{U}\mathcal{A})$

    (the first argument corresponds to holes [])
- More abstractly, we interpret them as functors on fibres
  - $[\![\Delta \vdash \psi]\!]_{\mathcal{A}} : \mathsf{RefAlg}_{\mathcal{A}} \times [\![\Delta]\!]_{\mathcal{A}} \longrightarrow \mathsf{RefAlg}_{\mathcal{A}}$

  where RefAlg results from change-of-base situation in

$$
\begin{tikzcd}
\mathbb{R} \arrow[rr, bend left, "\hat{\mathsf{F}}"] \arrow[rr, phantom, "\perp"] \arrow[d, "r"'] & & \mathsf{RefAlg} \arrow[ll, bend left, "\hat{\mathsf{U}}"] \arrow[d, "\mathsf{U}^*(r)"] \arrow[dl, phantom, very near start, "\lrcorner"] \\
\mathbb{V} \arrow[rr, bend left, "\mathsf{F}"] \arrow[rr, phantom, "\perp"] & & \mathsf{Alg} \arrow[ll, bend left, "\mathsf{U}"]
\end{tikzcd}
$$

- When $\vdash \psi$ then we have $[\![\vdash \psi]\!] : \mathsf{RefAlg} \longrightarrow \mathsf{RefAlg}$

Adding ref. types to effectful languages

- Fairly straightforward to add them to effectful languages, e.g., FGCBV or CBPV: [Levy et. al. '03] [Levy '04]

- For example, CBPV types
  - $A ::= \mathbf{b} \mid 1 \mid 0 \mid A_1 \times A_2 \mid A_1 + A_2 \mid \mathsf{U}\underline{C}$
  - $\underline{C} ::= \mathsf{F}A \mid 1 \mid \underline{C}_1 \times \underline{C}_2 \mid A \longrightarrow \underline{C}$

  turn into ref. types inspired by effect refinements

  - $\sigma ::= \mathbf{b} \mid 1 \mid 0 \mid \sigma_1 \times \sigma_2 \mid \sigma_1 + \sigma_2 \mid \hat{\mathsf{U}}\,\underline{\tau} \mid$
    $\sigma_1 \vee \sigma_2 \mid \bot_A$
  - $\underline{\tau} ::= \hat{\mathsf{F}}\,\sigma \mid 1 \mid \underline{\tau}_1 \times \underline{\tau}_2 \mid \sigma \longrightarrow \underline{\tau} \mid$
    $\langle \mathsf{op} \rangle_{\underline{C}}\,(\underline{\tau}_1, \ldots, \underline{\tau}_n) \mid X \mid \mu X \,.\, \underline{\tau} \mid \underline{\tau}_1 \vee \underline{\tau}_2 \mid \bot_{\underline{C}}$

- With the accompanying subtyping relations

  $$\Delta \vdash \sigma_1 \sqsubseteq_A \sigma_2 \qquad\qquad \Delta \vdash \underline{\tau}_1 \sqsubseteq_{\underline{C}} \underline{\tau}_2$$

  extended with rules for subtyping effect refinements

Adding ref. types to effectful languages

- The term syntax is as in CBPV

$$V \ ::= \ x \mid \langle V_1, V_2 \rangle \mid \ \ldots$$

$$M \ ::= \ \mathsf{return}\, V \mid M_1 \,\mathsf{to}\, x : \sigma \ \mathsf{in}\ M_2 \mid \ \ldots$$

- The typing judgments for CBPV become

$$\Gamma \models_{\mathsf{v}} V : \sigma \qquad \Gamma \models_{\mathsf{c}} M : \underline{\tau}$$

with the typing rules modified accordingly, e.g.:

$$\frac{\Gamma \models_{\mathsf{v}} V : \sigma}{\Gamma \models_{\mathsf{c}} \mathsf{return}\, V : \hat{\mathsf{F}}\, \sigma} \qquad \frac{\Gamma \models_{\mathsf{c}} M_1 : \underline{\tau}_1 \quad \ldots \quad \Gamma \models_{\mathsf{c}} M_n : \underline{\tau}_n}{\Gamma \models_{\mathsf{c}} \mathsf{op}(M_1, \ldots, M_n) : \langle \mathsf{op} \rangle_{\underline{C}}\, (\underline{\tau}_1, \ldots, \underline{\tau}_n)}$$

$$\frac{\Gamma \models_{\mathsf{c}} M_1 : \psi[\hat{\mathsf{F}}\, \sigma] \qquad \Gamma, x : \sigma \models_{\mathsf{c}} M_2 : \underline{\tau}}{\Gamma \models_{\mathsf{c}} M_1 \,\mathsf{to}\, x : \sigma \ \mathsf{in}\ M_2 : \psi[\underline{\tau}]}$$

where $\psi[\underline{\tau}]$ denotes "filling" of holes $[\ ]$ in ψ with $\underline{\tau}$

About the semantics of ref. typed CBPV

- Recall the picture for interpreting effect refs.

$$\begin{tikzcd}
\mathbb{R} \arrow[rr, bend left, "\hat{\mathsf{F}}"] \arrow[rr, phantom, "\perp"] \arrow[d, "r"'] & & \mathsf{RefAlg} \arrow[ll, bend left, "\hat{\mathsf{U}}"] \arrow[d, "\mathsf{U}^*(r)"] \\
\mathbb{V} \arrow[rr, bend left, "\mathsf{F}"] \arrow[rr, phantom, "\perp"] & & \mathsf{Alg} \arrow[ll, bend left, "\mathsf{U}"]
\end{tikzcd}$$

- Assume r to have suitable structure for types

- Ref. typed CBPV interpreted in the total categories:

$$[\![\vdash \sigma : \mathsf{Ref}(A)]\!] \in obj(\mathbb{R}) \text{ such that } r([\![\sigma]\!]) = [\![A]\!]$$

$$[\![\vdash \underline{\tau} : \mathsf{Ref}(\underline{C})]\!] \in obj(\mathsf{RefAlg}) \text{ such that } U^*(r)([\![\underline{\tau}]\!]) = [\![\underline{C}]\!]$$

$$[\![\Gamma \vDash_{\mathsf{v}} V : \sigma]\!] : [\![\Gamma]\!] \longrightarrow [\![\sigma]\!]$$

$$[\![\Gamma \vDash_{\mathsf{c}} M : \underline{\tau}]\!] : [\![\Gamma]\!] \longrightarrow (\hat{\mathsf{U}} \circ [\![\psi]\!])([\![\underline{\tau}]\!])$$

Applications: Type-and-effect systems

- Effect annotations ε in effect-and-type systems usually consist of sets of operation / effect symbols

- To represent type-and-effect systems in our system, we define effect refinements ψ_ε by

$$\psi_\varepsilon \stackrel{\text{def}}{=} \mu X \,.\, [\,] \;\vee \bigvee_{\mathsf{op}:n \in \varepsilon} \langle \mathsf{op} \rangle (X, \ldots, X)$$

- So we can talk of effect-and-type judgements

$$\Gamma \vdash M : \sigma \,!\, \varepsilon$$

as ref. typed judgements

$$\Gamma \vDash_{\mathsf{c}} M : \psi_\varepsilon[\hat{\mathsf{F}}\,\sigma]$$

Applications: Optimizations

- With a PER-based semantics also possible to validate
  effect-dependent optimizations
  [Benton et. al. '06-'09] [Kammar, Plotkin '12]
- For example:
  - discard

$$t(x, \dots, x) = x \text{ in } \mathcal{T}_{\mathrm{eff}} \text{ for all } \psi\text{-terms}$$

$$\frac{\Gamma \vDash_{\mathsf{c}} M : \psi[\hat{\mathsf{F}}\,\sigma] \qquad \Gamma \vDash_{\mathsf{c}} N : \underline{\tau}}{\Gamma \vDash_{\mathsf{c}} M \,\mathsf{to}\, x : \sigma \text{ in } N = N : \psi[\underline{\tau}]}$$

  - copy

$$t(t(x_{11}, \dots, x_{1n}), \dots, t(x_{n1}, \dots, x_{nn})) = t(x_{11}, \dots, x_{nn})$$

for all ψ-terms

$$\frac{\Gamma \vDash_{\mathsf{c}} M : \psi[\hat{\mathsf{F}}\,\sigma] \qquad \Gamma, x : \sigma, y : \sigma \vDash_{\mathsf{c}} N : \underline{\tau}}{\Gamma \vDash_{\mathsf{c}} M \,\mathsf{to}\, x : \sigma \text{ in } (M \,\mathsf{to}\, y : \sigma \text{ in } N) = M \,\mathsf{to}\, x : \sigma \text{ in } N[x/y] : \psi[\underline{\tau}]}$$

Applications: Optimizations

- But we can also validate more involved optimizations
  - effect refs. contain more temporal information

- Dead code elimination in stateful computation

$$\frac{\Gamma \vDash_{\mathsf{c}} M : \psi\Big[\langle \mathsf{upd}_{l,0}\rangle([\underline{\mathcal{T}}]) \vee \langle \mathsf{upd}_{l,1}\rangle([\underline{\mathcal{T}}])\Big] \qquad \langle \mathsf{lkp}_l\rangle \not\in \psi}{\Gamma \vDash_{\mathsf{c}} \mathsf{upd}_{l,i}(M) = M : \langle \mathsf{upd}_{l,i}\rangle\Big(\psi\Big[\langle \mathsf{upd}_{l,0}\rangle([\underline{\mathcal{T}}]) \vee \langle \mathsf{upd}_{l,1}\rangle([\underline{\mathcal{T}}])\Big]\Big)}$$

- Plus various other patterns describing how write- and read-information propagates through the terms

Applications: Hoare Logic

- Pre- and post-conditions on state turn out to be yet another example of formulae on computation trees
- Lack of value parameters $\Rightarrow$ combinatorial definition
- Take the predicates on state to be $P, Q \subseteq \{0, 1\}$
- Hoare refinement $\{P\}\sigma\{Q\}$ defined by case analysis on P

$$
\begin{aligned}
\{\emptyset\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle\mathsf{lkp}\rangle(\textstyle\bigvee_i \langle\mathsf{upd}_i\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_j \langle\mathsf{upd_j}\rangle([\hat{\mathsf{F}}\,\sigma])) \\
\{\{0\}\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle\mathsf{lkp}\rangle(\textstyle\bigvee_q \langle\mathsf{upd}_q\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_j \langle\mathsf{upd}_j\rangle([\hat{\mathsf{F}}\,\sigma])) \\
\{\{1\}\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle\mathsf{lkp}\rangle(\textstyle\bigvee_i \langle\mathsf{upd}_i\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_q \langle\mathsf{upd}_q\rangle([\hat{\mathsf{F}}\,\sigma])) \\
\{\{0,1\}\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle\mathsf{lkp}\rangle(\textstyle\bigvee_q \langle\mathsf{upd}_q\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_{q'} \langle\mathsf{upd}_{q'}\rangle([\hat{\mathsf{F}}\,\sigma]))
\end{aligned}
$$

where $i, j \in \{0, 1\}$ and $q, q' \in Q$

Applications: Hoare Logic

- Pre- and post-conditions on state turn out to be yet another example of formulae on computation trees
- Lack of value parameters $\Rightarrow$ combinatorial definition
- Take the predicates on state to be $P, Q \subseteq \{0,1\}$
- Hoare refinement $\{P\}\sigma\{Q\}$ defined by case analysis on P

$$
\begin{aligned}
\{\emptyset\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle \mathsf{lkp}\rangle(\textstyle\bigvee_i \langle \mathsf{upd}_i\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_j \langle \mathsf{upd}_j\rangle([\hat{\mathsf{F}}\,\sigma])) \\
\{\{0\}\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle \mathsf{lkp}\rangle(\textstyle\bigvee_q \langle \mathsf{upd}_q\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_j \langle \mathsf{upd}_j\rangle([\hat{\mathsf{F}}\,\sigma])) \\
\{\{1\}\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle \mathsf{lkp}\rangle(\textstyle\bigvee_i \langle \mathsf{upd}_i\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_q \langle \mathsf{upd}_q\rangle([\hat{\mathsf{F}}\,\sigma])) \\
\{\{0,1\}\}\,\sigma\,\{Q\} &\stackrel{\text{def}}{=} \langle \mathsf{lkp}\rangle(\textstyle\bigvee_q \langle \mathsf{upd}_q\rangle([\hat{\mathsf{F}}\,\sigma]), \bigvee_{q'} \langle \mathsf{upd}_{q'}\rangle([\hat{\mathsf{F}}\,\sigma]))
\end{aligned}
$$

where $i, j \in \{0,1\}$ and $q, q' \in Q$

Applications: Hoare Logic

- With the above def., Hoare Logic becomes admissible

$$\frac{\Gamma \vDash_{\mathrm{c}} M : \{P \cap \{0\}\}\,\sigma\,\{Q\} \qquad \Gamma \vDash_{\mathrm{c}} N : \{P \cap \{1\}\}\,\sigma\,\{Q\}}{\Gamma \vDash_{\mathrm{c}} \mathsf{lkp}(M, N) : \{P\}\,\sigma\,\{Q\}}$$

$$\frac{\Gamma \vDash_{\mathrm{c}} M : \{P\}\,\sigma\,\{Q\}}{\Gamma \vDash_{\mathrm{c}} \mathsf{upd}_i(M) : \{\bigvee_{P \cap \{i\}} \{0,1\}\}\,\sigma\,\{Q\}} \ (i \in \{0,1\})$$

$$\frac{\Gamma \vDash_{\mathrm{c}} M : \{P\}\,\sigma_1\,\{Q\} \qquad \Gamma, x : \sigma_1 \vDash_{\mathrm{c}} N : \{Q\}\,\sigma_2\,\{R\}}{\Gamma \vDash_{\mathrm{c}} M\,\mathsf{to}\,x : \sigma_1\ \mathsf{in}\ N : \{P\}\,\sigma_2\,\{R\}}$$

$$\frac{\Gamma \vDash_{\mathrm{v}} V : \sigma}{\Gamma \vDash_{\mathrm{c}} \mathsf{return}\ V : \{P\}\,\sigma\,\{P\}}$$

$$\frac{P \subseteq P' \qquad \Gamma \vDash_{\mathrm{c}} M : \{P'\}\,\sigma\,\{Q'\} \qquad Q' \subseteq Q}{\Gamma \vDash_{\mathrm{c}} M : \{P\}\,\sigma\,\{Q\}}$$

Applications: Protocols and sessions

- Protocol and session specifications are yet another example of formulae on computation trees
- For example, the correct usage of files
- Using a file correctly once:

$$\psi_{\mathsf{file}} \stackrel{\mathsf{def}}{=} \langle \mathsf{open} \rangle \Big(\mu X \,.\, \Big(\langle \mathsf{close} \rangle ([\,]) \vee \langle \mathsf{write}_i \rangle (X) \vee \langle \mathsf{read} \rangle (X, X) \Big) \Big)$$

- Using a file correctly repetitively:

$$\psi_{\mathsf{rep\text{-}file}} \stackrel{\mathsf{def}}{=} \mu Y \,.\, \Big([\,] \vee \psi_{\mathsf{file}}[Y] \Big)$$

- Finally, also straightforward to define session-type style refinements, e.g., I/O corresponding to the grammar

$$S \ ::= \ !(0).S \mid !(1).S \mid !(0 \vee 1).S \mid ?(S_1, S_2) \mid \mathit{end}$$

Conclusions

- In this talk:
  - Effect refs. as formulae on equiv. classes of comp. trees
  - Ref. types in computational languages (CBPV)
  - Importance of (semi-)linearity in equations
  - Specification and optimization examples
- Not in this talk:
  - Handlers (need ref. types to have free vars X)
  - Effect refs. for operations with parameters and binding
  - Type-dependency in ref. types over simple types