Refinement Types for Algebraic Effects

Danel Ahman    Gordon Plotkin
LFCS, University of Edinburgh

TYPES 2015
Tallinn
Today’s plan

Value ref. types → Computation ref. types

→ Algebraic theories → Value types

Computation types ← Value types

+ some examples
Refinement types

- For extending base language’s type system
  - to allow more precise specifications in types
    \[ \vdash \text{Odd} : \text{Ref} (\text{Nat}) \quad \vdash \text{Even} : \text{Ref} (\text{Nat}) \]
  - make it possible to internalize meta-theorems
    \[ n : \text{Odd}, \ m : \text{Odd} \vdash n + m : \text{Even} \]
    and also program optimizations

- In this talk, we discuss propositional ref. types
  - for example Even and Odd, as above
    \[ \text{[Freeman, Pfenning ’91]} \]

- Ideas also apply to FOL-based ref. types \[ \text{[Denney ’98]} \]
  - 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: $TX = \mathcal{P}^+_\text{fin}(X)$
  - read-only memory: $TX = S \to X$
  - write-only memory: $TX = M \times X$ (M a monoid)
  - read-write memory / global state: $TX = S \to (S \times X)$

- And also more recent generalizations from TYPES '13 [Ahman, Uustalu '14]
  - update monads: $TX = S \to (P \times X)$ (P a monoid)
  - dep. typed update monads: $TX = \Pi s : S.(Ps \times X)$
    (where $(S,\downarrow,P,o,\oplus)$ 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** \(!\text{Bool}.?\text{Nat}.S\) for I/O
  - trace effects [Skalka, Smith, van Horn '08]
  - session typed languages [Honda '93] [and many others]

- **effect annotations** \( \varepsilon \) 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 $\text{op} : n$
  - not in this talk: operations with parameters and binding

- For example:
  - **non-determinism**: $T X = \mathcal{P}_{\text{fin}}^+(X)$
    - $x \text{ or } x = x$
    - $x \text{ or } y = y \text{ or } x$
    - $x \text{ or } (y \text{ or } z) = (x \text{ or } y) \text{ or } z$
  - **state**: $T X = 2 \to (2 \times X)$ (where $S = 2$)
    - $\text{lkp}(\text{upd}_0(x), \text{upd}_1(x)) = x$
    - $\text{upd}_i(\text{upd}_j(x)) = \text{upd}_j(x)$
    - $\text{upd}_i(\text{lkp}(x_0, x_1)) = \text{upd}_i(x_i)$
Effectful programs as computation trees

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

\[
\begin{align*}
\text{let } f &= \lambda b : \text{bool}. \text{return } \neg b \text{ in} \\
\text{let } x &= \text{lkp} \text{ in} \\
\text{let } y &= f \times \text{in} \\
\text{let } _ &= \text{output } y \text{ in} \\
\text{let } _ &= \text{if } x = 1 \text{ then } \text{upd } y \text{ in} \\
\text{return } y
\end{align*}
\]

as computation trees

```
let f = \lambda b : bool . return \neg b in
let x = lkp in
let y = f x in
let _ = output y in
let _ = if x = 1 then upd y in
return y
```
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

  \[ \psi ::= [ ] \mid \langle \text{op} \rangle(\psi_1, \ldots, \psi_n) \mid \bot \mid \psi_1 \lor \psi_2 \mid X \mid \mu X.\psi \]

  where

  - “holes” \([ ]\) are placeholders for leaves
  - op. modalities \(\langle \text{op} \rangle\) 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
  \[
  \psi ::= [ ] | \langle \text{op} \rangle (\psi_1, \ldots, \psi_n) | \perp | \psi_1 \lor \psi_2 | X | \mu X.\psi
  \]
- They also come with a satisfiability / subtyping relation
  \[
  \Delta \vdash \psi_1 \sqsubseteq \psi_2
  \]
- \sqsubseteq\text{ includes standard logic}
- also want \sqsubseteq to include algebraic properties of \langle \text{op} \rangle’s
  - can’t just include all the axioms, e.g., \psi = \langle \text{lkp} \rangle (\psi, \psi)
    \[\text{[Gautam '57]}\]
  - need to include derivable semi-linear equations
    \[
    \vec{x} \vdash t = u \text{ derivable in } T_{\text{eff}} \quad t \text{ linear in } \vec{x} \quad Vars(u) \subseteq Vars(t)
    \]
    \[
    \Delta \vdash \psi_1 \quad \ldots \quad \Delta \vdash \psi_n
    \]
    \[
    \Delta \vdash t^\bullet[\vec{\psi}/\vec{x}] \sqsubseteq u^\bullet[\vec{\psi}/\vec{x}]
    \]
About the semantics of effect refinements

- Recall: effect refs. are **indifferent** wrt. specific algebras
- Concretely, they can be interpreted as **monotone maps**
  \[
  [\Delta \vdash \psi]_A : \mathcal{P}(UA) \times [\Delta]_A \to \mathcal{P}(UA)
  \]
  (the first argument corresponds to holes `[]`)
- More abstractly, we interpret them as **functors** on fibres
  \[
  [\Delta \vdash \psi]_A : \text{RefAlg}_A \times [\Delta]_A \to \text{RefAlg}_A
  \]
  where RefAlg results from change-of-base situation in

  ![Diagram]

- When $\vdash \psi$ then we have $[\vdash \psi] : \text{RefAlg} \to \text{RefAlg}$
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
  - $A ::= b | 1 | 0 | A_1 \times A_2 | A_1 + A_2 | UC$
  - $C ::= FA | 1 | C_1 \times C_2 | A \rightarrow C$

- turn into ref. types inspired by effect refinements
  - $\sigma ::= b | 1 | 0 | \sigma_1 \times \sigma_2 | \sigma_1 + \sigma_2 | \hat{U} \tau$
  - $\sigma_1 \lor \sigma_2 | \perp_A$
  - $\tau ::= \hat{F} \sigma | 1 | \tau_1 \times \tau_2 | \sigma \rightarrow \tau$
  - $\langle \text{op} \rangle_C(\tau_1, \ldots, \tau_n) | X | \mu X \cdot \tau | \tau_1 \lor \tau_2 | \perp_C$

- With the accompanying subtyping relations
  - $\Delta \vdash \sigma_1 \sqsubseteq_A \sigma_2$
  - $\Delta \vdash \tau_1 \sqsubseteq_C \tau_2$

  extended with rules for subtyping effect refinements
Adding ref. types to effectful languages

- The term syntax is as in CBPV

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

\[ M ::= \text{return } V \mid M_1 \text{ to } x : \sigma \text{ in } M_2 \mid \ldots \]

- The typing judgments for CBPV become

\[ \Gamma \vdash V : \sigma \quad \Gamma \vdash M : \tau \]

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

\[ \frac{\Gamma \vdash V : \sigma}{\Gamma \vdash \text{return } V : \hat{F} \sigma} \quad \frac{\Gamma \vdash M_1 : \tau_1 \quad \ldots \quad \Gamma \vdash M_n : \tau_n}{\Gamma \vdash \text{op}(M_1, \ldots, M_n) : \langle \text{op} \rangle_C(\tau_1, \ldots, \tau_n)} \]

\[ \frac{\Gamma \vdash M_1 : \psi[\hat{F} \sigma] \quad \Gamma, x : \sigma \vdash M_2 : \tau}{\Gamma \vdash M_1 \text{ to } x : \sigma \text{ in } M_2 : \psi[\tau]} \]

where \( \psi[\tau] \) denotes “filling” of holes \([ \ ]\) in \( \psi \) with \( \tau \)
About the semantics of ref. typed CBPV

- Recall the picture for interpreting effect refs.

- Assume $r$ to have suitable structure for types

- Ref. typed CBPV interpreted in the total categories:

\[
\begin{align*}
[D \vdash \sigma : \text{Ref}(A)] & \in \text{obj}(\mathbb{R}) \text{ such that } r([\sigma]) = [A] \\
[D \vdash \tau : \text{Ref}(C)] & \in \text{obj}(\text{RefAlg}) \text{ such that } U^*(r)([\tau]) = [C] \\
[D \vdash V : \sigma] : [D] & \longrightarrow [\sigma] \\
[D \vdash M : \tau] : [D] & \longrightarrow (\hat{U} \circ [\psi])([\tau])
\end{align*}
\]
Applications: Type-and-effect systems

- **Effect annotations** $\varepsilon$ 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** $\psi_\varepsilon$ by

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

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

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

  as ref. typed judgements

  $$
  \Gamma \vdash_\varepsilon M : \psi_\varepsilon[\hat{F} \sigma]
  $$
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
    \[
    t(x, \ldots, x) = x \text{ in } T_{\text{eff}} \text{ for all } \psi\text{-terms}
    \]
    \[
    \Gamma \vdash_{c} M : \psi[\hat{F} \sigma] \quad \Gamma \vdash_{c} N : \tau
    \]
    \[
    \Gamma \vdash_{c} M \text{ to } x : \sigma \text{ in } N = N : \psi[\tau]
    \]
  - copy
    \[
    t(t(x_{11}, \ldots, x_{1n}), \ldots, t(x_{n1}, \ldots, x_{nn})) = t(x_{11}, \ldots, x_{nn})
    \text{ for all } \psi\text{-terms}
    \]
    \[
    \Gamma \vdash_{c} M : \psi[\hat{F} \sigma] \quad \Gamma, x : \sigma, y : \sigma \vdash_{c} N : \tau
    \]
    \[
    \Gamma \vdash_{c} M \text{ to } x : \sigma \text{ in } (M \text{ to } y : \sigma \text{ in } N) = M \text{ to } x : \sigma \text{ in } N[x/y : \psi[\tau]]
    \]
Applications: Optimizations

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

- Dead code elimination in stateful computation

\[
\Gamma \vdash c \ M : \psi \left[ \langle \text{upd}_{i,0} \rangle ([\tau]) \lor \langle \text{upd}_{i,1} \rangle ([\tau]) \right] \quad \langle \text{lkp}_l \rangle \not\in \psi
\]

\[
\Gamma \vdash \text{upd}_{i,i}(M) = M : \langle \text{upd}_{i,i} \rangle \left( \psi \left[ \langle \text{upd}_{i,0} \rangle ([\tau]) \lor \langle \text{upd}_{i,1} \rangle ([\tau]) \right] \right)
\]

- 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 ⇒ combinatorial definition
- Take the predicates on state to be \( P, Q \subseteq \{0, 1\} \)
- Hoare refinement \( \{P\} \sigma \{Q\} \) defined by case analysis on \( P \)

\[
\begin{align*}
\{\emptyset\} \sigma \{Q\} & \defeq \langle \text{lkp} \rangle (\bigvee_i \langle \text{upd}_i \rangle ([\hat{F} \sigma]), \bigvee_j \langle \text{upd}_j \rangle ([\hat{F} \sigma])) \\
\{\{0\}\} \sigma \{Q\} & \defeq \langle \text{lkp} \rangle (\bigvee_q \langle \text{upd}_q \rangle ([\hat{F} \sigma]), \bigvee_j \langle \text{upd}_j \rangle ([\hat{F} \sigma])) \\
\{\{1\}\} \sigma \{Q\} & \defeq \langle \text{lkp} \rangle (\bigvee_i \langle \text{upd}_i \rangle ([\hat{F} \sigma]), \bigvee_q \langle \text{upd}_q \rangle ([\hat{F} \sigma])) \\
\{\{0, 1\}\} \sigma \{Q\} & \defeq \langle \text{lkp} \rangle (\bigvee_q \langle \text{upd}_q \rangle ([\hat{F} \sigma]), \bigvee_{q'} \langle \text{upd}_{q'} \rangle ([\hat{F} \sigma]))
\end{align*}
\]

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 $P$

\[
\begin{align*}
\{\emptyset\} \sigma \{Q\} & \overset{\text{def}}{=} \langle \text{lkp} \rangle (\lor_i \langle \text{upd}_i \rangle ([\hat{F} \sigma]), \lor_j \langle \text{upd}_j \rangle ([\hat{F} \sigma])) \\
\{\{0\}\} \sigma \{Q\} & \overset{\text{def}}{=} \langle \text{lkp} \rangle (\lor_q \langle \text{upd}_q \rangle ([\hat{F} \sigma]), \lor_j \langle \text{upd}_j \rangle ([\hat{F} \sigma])) \\
\{\{1\}\} \sigma \{Q\} & \overset{\text{def}}{=} \langle \text{lkp} \rangle (\lor_i \langle \text{upd}_i \rangle ([\hat{F} \sigma]), \lor_q \langle \text{upd}_q \rangle ([\hat{F} \sigma])) \\
\{\{0, 1\}\} \sigma \{Q\} & \overset{\text{def}}{=} \langle \text{lkp} \rangle (\lor_q \langle \text{upd}_q \rangle ([\hat{F} \sigma]), \lor_{q'} \langle \text{upd}_{q'} \rangle ([\hat{F} \sigma]))
\end{align*}
\]

where $i, j \in \{0, 1\}$ and $q, q' \in Q$
Applications: Hoare Logic

With the above def., Hoare Logic becomes admissible

\[
\Gamma \vdash c \ M : \ \{ P \cap \{0\} \} \ \sigma \ \{ Q \} \quad \Gamma \vdash c \ N : \ \{ P \cap \{1\} \} \ \sigma \ \{ Q \} \\
\Gamma \vdash c \ \text{lk}(M, N) : \ \{ P \} \ \sigma \ \{ Q \} \\
\Gamma \vdash c \ \text{upd}_i(M) : \ \{ \bigvee_{P \cap \{i\}} \{0, 1\} \} \ \sigma \ \{ Q \} \quad (i \in \{0, 1\})
\]

\[
\Gamma \vdash c \ M : \ \{ P \} \ \sigma_1 \ \{ Q \} \quad \Gamma, x : \sigma_1 \vdash c \ N : \ \{ Q \} \ \sigma_2 \ \{ R \} \\
\Gamma \vdash c \ M \text{ to } x : \sigma_1 \ \text{in} \ N : \ \{ P \} \ \sigma_2 \ \{ R \}
\]

\[
\Gamma \vdash c \ \text{return} \ V : \ \{ P \} \ \sigma \ \{ P \}
\]

\[
P \subseteq P' \quad \Gamma \vdash c \ M : \ \{ P' \} \ \sigma \ \{ Q' \} \\
\Gamma \vdash c \ M : \ \{ P \} \ \sigma \ \{ Q \} \quad Q' \subseteq Q
\]
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:

  \[\psi_{\text{file}} \stackrel{\text{def}}{=} \langle \text{open} \rangle \left( \mu X . \left( \langle \text{close} \rangle ([ ] \lor \langle \text{write}_i \rangle (X) \lor \langle \text{read} \rangle (X, X)) \right) \right)\]

- Using a file correctly repetitively:

  \[\psi_{\text{rep-file}} \stackrel{\text{def}}{=} \mu Y . \left( [ ] \lor \psi_{\text{file}}[Y] \right)\]

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

  \[S ::= !(0).S \mid !(1).S \mid !(0 \lor 1).S \mid ?(S_1, S_2) \mid \text{end}\]
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