{ Refinement Types | Algebraic Effects }

Danel Ahman
LFCS, University of Edinburgh

HOPE Workshop, 28 September 2013

informatics

lfcs | Laboratory for Foundations of Computer Science

THE UNIVERSITY of EDINBURGH

Overview

- Refinement types & effects
- What do we feel is missing from refinement type systems?
  - A uniform treatment of various computational effects
  - General logical specifications for arbitrary effects
- Our way of bridging this gap
  - Algebraic effects and their logics
  - General effectful ref. types through algebraic effectful reasoning
  - Hopefully leads us to a general theory of effectful refinement types
- Some examples
  - State and pre-/post-conditions
  - Communication and sessions
  - Combination of the two

Overview

- Refinement types & effects
- What do we feel is missing from refinement type systems?
  - A uniform treatment of various computational effects
  - General logical specifications for arbitrary effects
- Our way of bridging this gap
  - Algebraic effects and their logics
  - General effectful ref. types through algebraic effectful reasoning
  - Hopefully leads us to a general theory of effectful refinement types
- Some examples
  - State and pre-/post-conditions
  - Communication and sessions
  - Combination of the two

Overview

- Refinement types & effects
- What do we feel is missing from refinement type systems?
  - A uniform treatment of various computational effects
  - General logical specifications for arbitrary effects
- Our way of bridging this gap
  - Algebraic effects and their logics
  - General effectful ref. types through algebraic effectful reasoning
  - Hopefully leads us to a general theory of effectful refinement types
- Some examples
  - State and pre-/post-conditions
  - Communication and sessions
  - Combination of the two

Overview

- Refinement types & effects
- What do we feel is missing from refinement type systems?
  - A uniform treatment of various computational effects
  - General logical specifications for arbitrary effects
- Our way of bridging this gap
  - Algebraic effects and their logics
  - General effectful ref. types through algebraic effectful reasoning
  - Hopefully leads us to a general theory of effectful refinement types
- Some examples
  - State and pre-/post-conditions
  - Communication and sessions
  - Combination of the two

Effects in refinement type systems

- Most current refinement type systems target specific effects:
  - F7 extended with a refined state monad Borgström et. al. '09
    - by adding a new computation type $\{(s_0)\varphi_0\}x : \sigma\{(s_1)\varphi_1\}$
  - Monadic F* with a Dijkstra monad Swamy et. al. '13
    - by adding a comp. type $M\,\sigma\,wp$
  - Session types with linear refinement types Baltazar et. al. '12
    - by adding ref. ty. $\{x : T \mid \varphi\}$ to session types (with φ in MLL)
- Some systems are more abstract in effects they consider:
  - Effective theory of type refinements Mandelbaum et. al. '03
    - term refinements φ: bool, its(t), $\varphi_1 \to \varphi_2$, $(\varphi_1, \psi_1) \rightharpoonup (\varphi_2, \psi_2)$
    - world refinements ψ: formulas in linear logic
    - parametrized by a set of operations (together with a signature of operation refinements and a transition function for operations)

Effects in refinement type systems

- Most current refinement type systems target specific effects:
  - F7 extended with a refined state monad Borgström et. al. '09
    - by adding a new computation type $\{(s_0)\varphi_0\}x : \sigma\{(s_1)\varphi_1\}$
  - Monadic F* with a Dijkstra monad Swamy et. al. '13
    - by adding a comp. type $M\,\sigma\,wp$
  - Session types with linear refinement types Baltazar et. al. '12
    - by adding ref. ty. $\{x : T \mid \varphi\}$ to session types (with φ in MLL)
- Some systems are more abstract in effects they consider:
  - Effective theory of type refinements Mandelbaum et. al. '03
    - term refinements φ: bool, its(t), $\varphi_1 \to \varphi_2$, $(\varphi_1, \psi_1) \rightharpoonup (\varphi_2, \psi_2)$
    - world refinements ψ: formulas in linear logic
    - parametrized by a set of operations (together with a signature of operation refinements and a transition function for operations)

Effects in refinement type systems

- Most current refinement type systems target specific effects:
  - F7 extended with a refined state monad Borgström et. al. '09
    - by adding a new computation type $\{(s_0)\varphi_0\}x : \sigma\{(s_1)\varphi_1\}$
  - Monadic F* with a Dijkstra monad Swamy et. al. '13
    - by adding a comp. type $M\,\sigma\,wp$
  - Session types with linear refinement types Baltazar et. al. '12
    - by adding ref. ty. $\{x : T \mid \varphi\}$ to session types (with φ in MLL)
- Some systems are more abstract in effects they consider:
  - Effective theory of type refinements Mandelbaum et. al. '03
    - term refinements φ: bool, its(t), $\varphi_1 \to \varphi_2$, $(\varphi_1, \psi_1) \rightharpoonup (\varphi_2, \psi_2)$
    - world refinements ψ: formulas in linear logic
    - parametrized by a set of operations (together with a signature of operation refinements and a transition function for operations)

Effects in refinement type systems

- Most current refinement type systems target specific effects:
  - F7 extended with a refined state monad Borgström et. al. '09
    - by adding a new computation type $\{(s_0)\varphi_0\}x : \sigma\{(s_1)\varphi_1\}$
  - Monadic F* with a Dijkstra monad Swamy et. al. '13
    - by adding a comp. type $M\,\sigma\,wp$
  - Session types with linear refinement types Baltazar et. al. '12
    - by adding ref. ty. $\{x : T \mid \varphi\}$ to session types (with φ in MLL)
- Some systems are more abstract in effects they consider:
  - Effective theory of type refinements Mandelbaum et. al. '03
    - term refinements φ: bool, its(t), $\varphi_1 \to \varphi_2$, $(\varphi_1, \psi_1) \rightharpoonup (\varphi_2, \psi_2)$
    - world refinements ψ: formulas in linear logic
    - parametrized by a set of operations (together with a signature of operation refinements and a transition function for operations)

Different effects, languages and specifications

- Consider a (fragment of a) simple communication language:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}\, t : FA} \qquad \frac{\Gamma \vdash t : FA \quad \Gamma, x : A \vdash u : FB}{\Gamma \vdash t \,\mathsf{to}\, x.\, u : FB}$$

$$\frac{\Gamma, x : \mathsf{nat} \vdash t : FA}{\Gamma \vdash \mathsf{receive}(x.t) : FA} \qquad \frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : FA}{\Gamma \vdash \mathsf{send}_t(u) : FA}$$

- Session refinements (inspired by session types)
  - $S(A) ::= end(A) \mid ?(x : \mathsf{nat}).S(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A)$
- Example programs with their refinements:
  - $\Gamma \vdash \mathsf{receive}(x.\mathsf{receive}(y.t)) : ?(x : \mathsf{nat}).?(y : \mathsf{nat}).S(1)$
  - $\Gamma \vdash \mathsf{send}_t(\mathsf{send}_{t+1}(u)) : !(x : \mathsf{nat} \mid \top).!(y : \mathsf{nat} \mid y > x).S(1)$

Different effects, languages and specifications

- Consider a (fragment of a) simple communication language:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}\, t : \mathit{end}(A)} \qquad \frac{\Gamma \vdash t : S(A) \quad \Gamma, x : A \vdash u : S(B)}{\Gamma \vdash t\, \mathsf{to}\, x.\, u : S(A); S(B)}$$

$$\frac{\Gamma, x : \mathsf{nat} \vdash t : S(A)}{\Gamma \vdash \mathsf{receive}(x.t) : ?(x : \mathsf{nat}).S(A)} \qquad \frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash \varphi[t/x] \quad \Gamma \vdash u : S(A)}{\Gamma \vdash \mathsf{send}_t(u) : !(x : \mathsf{nat} \mid \varphi).S(A)}$$

- Session refinements (similar syntax to session types):
  - $S(A) ::= \mathit{end}(A) \mid ?(x : \mathsf{nat}).S(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A)$
- Example programs with their refinements:
  - $\Gamma \vdash \mathsf{receive}(x.\mathsf{receive}(y.t)) : ?(x : \mathsf{nat}).?(y : \mathsf{nat}).S(1)$
  - $\Gamma \vdash \mathsf{send}_t(\mathsf{send}_{t+1}(u)) : !(x : \mathsf{nat} \mid \top).!(y : \mathsf{nat} \mid y > x).S(1)$

Different effects, languages and specifications

- Consider a (fragment of a) simple communication language:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}\, t : \mathit{end}(A)} \qquad \frac{\Gamma \vdash t : S(A) \quad \Gamma, x : A \vdash u : S(B)}{\Gamma \vdash t\, \mathsf{to}\, x.\, u : S(A); S(B)}$$

$$\frac{\Gamma, x : \mathsf{nat} \vdash t : S(A)}{\Gamma \vdash \mathsf{receive}(x.t) : ?(x : \mathsf{nat}).S(A)} \qquad \frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash \varphi[t/x] \quad \Gamma \vdash u : S(A)}{\Gamma \vdash \mathsf{send}_t(u) : !(x : \mathsf{nat} \mid \varphi).S(A)}$$

- Session refinements (similar syntax to session types):
  - $S(A) ::= \mathit{end}(A) \mid ?(x : \mathsf{nat}).S(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A)$
- Example programs with their refinements:
  - $\Gamma \vdash \mathsf{receive}(x.\mathsf{receive}(y.t)) : ?(x : \mathsf{nat}).?(y : \mathsf{nat}).S(1)$
  - $\Gamma \vdash \mathsf{send}_t(\mathsf{send}_{t+1}(u)) : !(x : \mathsf{nat} \mid \top).!(y : \mathsf{nat} \mid y > x).S(1)$

Different effects, languages and specifications

- Consider a (fragment of a) simple state language:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}\, t : FA} \qquad \frac{\Gamma \vdash t : FA \quad \Gamma, x : A \vdash u : FB}{\Gamma \vdash t \,\mathsf{to}\, x.\, u : FB}$$

$$\frac{\Gamma, x : \mathsf{nat} \vdash t : FA}{\Gamma \vdash \mathsf{lookup}(x.t) : FA} \qquad \frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : FA}{\Gamma \vdash \mathsf{update}_t(u) : FA}$$

- Pre- & post-condition specifications:

$$\forall \vec{x}.\{(x_0).\varphi_P\}x.A\{(x_1).\varphi_Q\}$$

- Example program with its refinement:

$$\Gamma \vdash \mathsf{lookup}(x.\mathsf{update}_{x+1}(\mathsf{return}\,\star)) : \{(x_0).odd(x_0)\}x : 1\{(x_1).even(x_1)\}$$

Different effects, languages and specifications

- Consider a (fragment of a) simple state language:

$$\frac{\Gamma, x : \mathsf{nat} \vdash t : \forall \vec{x}, x_0.\{(x_1).\varphi_Q\}y : A\{(x_2).\varphi_R\} \qquad \Gamma \vdash \forall \vec{x}.\{(x_0).\top\}x : \mathsf{nat}\{(x_1).x_1 = x_0 \wedge x_1 = y\} \sqsubseteq \forall \vec{x}.\{(x_0).\varphi_P\}x : \mathsf{nat}\{(x_1).\varphi_Q\}}{\Gamma \vdash \mathsf{lookup}(x.t) : \forall \vec{x}.\{(x_0).\varphi_P\}y : A\{(x_2).\varphi_R\}}$$

$$\frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : \forall \vec{x}, x_0.\{(x_1).\varphi_Q\}x : A\{(x_2).\varphi_R\} \qquad \Gamma \vdash \forall \vec{x}.\{(x_0).\top\}\_ : 1\{(x_1).x_1 = t\} \sqsubseteq \forall \vec{x}.\{(x_0).\varphi_P\}\_ : 1\{(x_1).\varphi_Q\}}{\Gamma \vdash \mathsf{update}_t(u) : \forall \vec{x}.\{(x_0).\varphi_P\}x : A\{(x_2).\varphi_R\}}$$

- Pre- & post-condition specifications:

$$\forall \vec{x}.\{(x_0).\varphi_P\}x : A\{(x_1).\varphi_Q\}$$

- Example program with its refinement:

$$\Gamma \vdash \mathsf{lookup}(x.\mathsf{update}_{x+1}(\mathsf{return}\,\star)) : \{(x_0).odd(x_0)\}x : 1\{(x_1).even(x_1)\}$$

Different effects, languages and specifications

- Consider a (fragment of a) simple state language:

$$\frac{\Gamma, x : \mathsf{nat} \vdash t : \forall \vec{x}, x_0.\{(x_1).\varphi_Q\}y : A\{(x_2).\varphi_R\} \qquad \Gamma \vdash \forall \vec{x}.\{(x_0).\top\}x : \mathsf{nat}\{(x_1).x_1 = x_0 \wedge x_1 = y\} \sqsubseteq \forall \vec{x}.\{(x_0).\varphi_P\}x : \mathsf{nat}\{(x_1).\varphi_Q\}}{\Gamma \vdash \mathsf{lookup}(x.t) : \forall \vec{x}.\{(x_0).\varphi_P\}y : A\{(x_2).\varphi_R\}}$$

$$\frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : \forall \vec{x}, x_0.\{(x_1).\varphi_Q\}x : A\{(x_2).\varphi_R\} \qquad \Gamma \vdash \forall \vec{x}.\{(x_0).\top\}\_ : 1\{(x_1).x_1 = t\} \sqsubseteq \forall \vec{x}.\{(x_0).\varphi_P\}\_ : 1\{(x_1).\varphi_Q\}}{\Gamma \vdash \mathsf{update}_t(u) : \forall \vec{x}.\{(x_0).\varphi_P\}x : A\{(x_2).\varphi_R\}}$$

- Pre- & post-condition specifications:

$$\forall \vec{x}.\{(x_0).\varphi_P\}x : A\{(x_1).\varphi_Q\}$$

- Example program with its refinement:

$$\Gamma \vdash \mathsf{lookup}(x.\mathsf{update}_{x+1}(\mathsf{return}\,\star)) : \{(x_0).odd(x_0)\}x : 1\{(x_1).even(x_1)\}$$

Different effects, languages and specifications

- Also want a combination of these languages and specifications
- For example, combining state and communication:

$$\forall \vec{x}.\{(x_0).\varphi_P\}(S(A) \rightarrowtail x : A)\{(x_1).\varphi_Q\}$$

- Example program with a composite refinement:

$\langle\rangle \vdash \mathsf{receive}(x.\mathsf{lookup}(y.\mathsf{if}\ y > x\ \mathsf{then}\ \mathsf{update}_{y-x}(\mathsf{return}\,\star)\ \mathsf{else}\ \mathsf{return}\,\star)) :$

$$\{(x_0).\top\}(?(x : \mathsf{nat}).end(1) \rightarrowtail y : 1)\{(x_1).(x > x_0) \implies x_1 = x_0 - x\}$$

- Other effects and their specs.?
- Non-standard combinations of specs.?

Different effects, languages and specifications

- Also want a combination of these languages and specifications
- For example, combining state and communication:

$$\forall \vec{x}.\{(x_0).\varphi_P\}(S(A) \succ x : A)\{(x_1).\varphi_Q\}$$

- Example program with a composite refinement:

$$\langle\rangle \vdash \mathsf{receive}(x.\mathsf{lookup}(y.\mathsf{if}\ y > x\ \mathsf{then}\ \mathsf{update}_{y-x}(\mathsf{return}\,\star)\ \mathsf{else}\ \mathsf{return}\,\star)) :$$

$$\{(x_0).\top\}(?(x : \mathsf{nat}).end(1) \succ y : 1)\{(x_1).(x > x_0) \implies x_1 = x_0 - x\}$$

- Other effects and their specs.?
- Non-standard combinations of specs.?

Different effects, languages and specifications

- Also want a combination of these languages and specifications
- For example, combining state and communication:

$$\forall \vec{x}.\{(x_0).\varphi_P\}(S(A) \succ x : A)\{(x_1).\varphi_Q\}$$

- Example program with a composite refinement:

$\langle\rangle \vdash \mathsf{receive}(x.\mathsf{lookup}(y.\mathsf{if}\ y > x\ \mathsf{then}\ \mathsf{update}_{y-x}(\mathsf{return}\,\star)\ \mathsf{else}\ \mathsf{return}\,\star))$:

$$\{(x_0).\top\}(?(x : \mathsf{nat}).\mathit{end}(1) \succ y : 1)\{(x_1).(x > x_0) \implies x_1 = x_0 - x\}$$

- Other effects and their specs.?
- Non-standard combinations of specs.?

Different effects, languages and specifications

- Also want a combination of these languages and specifications
- For example, combining state and communication:

$$\forall \vec{x}. \{(x_0).\varphi_P\}(S(A) \succ x : A)\{(x_1).\varphi_Q\}$$

- Example program with a composite refinement:

$\langle\rangle \vdash \mathsf{receive}(x.\mathsf{lookup}(y.\mathsf{if}\ y > x\ \mathsf{then}\ \mathsf{update}_{y-x}(\mathsf{return}\,\star)\ \mathsf{else}\ \mathsf{return}\,\star))$:

$$\{(x_0).\top\}(?(x : \mathsf{nat}).end(1) \succ y : 1)\{(x_1).(x > x_0) \implies x_1 = x_0 - x\}$$

- Other effects and their specs.?
- Non-standard combinations of specs.?

Our proposed approach

A computational language with algebraic effects

$+$

– ref. types for general effectful specs.
– using algebraic effectful reasoning

$\subseteq$ $\quad$ $\subseteq$ $\quad$ $\subseteq$

State language $\quad$ Communication language $\quad$ Language X

Refinement types

- The style of ref. types we work with (no effects for time being):
  - λ-calculus with types $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2$
  - Refinement types $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid \{x : \sigma \mid \varphi\}$
  - Well-formed refinement types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$, e.g.:

$$\frac{\vdash \Gamma \ \mathsf{wf}}{\Gamma \vdash \alpha : \mathsf{Ref}(\alpha)} \qquad \frac{\Gamma \vdash \sigma_1 : \mathsf{Ref}(A_1) \qquad \Gamma, x : \sigma_1 \vdash \sigma_2 : \mathsf{Ref}(A_2)}{\Gamma \vdash \Pi_{x:\sigma_1}\sigma_2 : \mathsf{Ref}(A_1 \to A_2)}$$

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)}$$

  - Well-typed refined terms $\Gamma \vdash t : \sigma$, e.g.:

$$\frac{\Gamma \vdash t : \sigma \qquad |\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}{\Gamma \vdash t : \{x : \sigma \mid \varphi\}}$$

Denney '98

Refinement types

- The style of ref. types we work with (no effects for time being):
  - λ-calculus with types $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2$
  - Refinement types $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid \{x : \sigma \mid \varphi\}$
  - Well-formed refinement types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$, e.g.:

$$\frac{\vdash \Gamma \ \mathsf{wf}}{\Gamma \vdash \alpha : \mathsf{Ref}(\alpha)} \qquad \frac{\Gamma \vdash \sigma_1 : \mathsf{Ref}(A_1) \qquad \Gamma, x : \sigma_1 \vdash \sigma_2 : \mathsf{Ref}(A_2)}{\Gamma \vdash \Pi_{x:\sigma_1}\sigma_2 : \mathsf{Ref}(A_1 \to A_2)}$$

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)}$$

  - Well-typed refined terms $\Gamma \vdash t : \sigma$, e.g.:

$$\frac{\Gamma \vdash t : \sigma \qquad |\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}{\Gamma \vdash t : \{x : \sigma \mid \varphi\}}$$

Denney '98

Refinement types

- The style of ref. types we work with (no effects for time being):
  - λ-calculus with types $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2$
  - Refinement types $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid \{x : \sigma \mid \varphi\}$
  - Well-formed refinement types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$, e.g.:

$$\frac{\vdash \Gamma \ \ \mathsf{wf}}{\Gamma \vdash \alpha : \mathsf{Ref}(\alpha)} \qquad \frac{\Gamma \vdash \sigma_1 : \mathsf{Ref}(A_1) \qquad \Gamma, x : \sigma_1 \vdash \sigma_2 : \mathsf{Ref}(A_2)}{\Gamma \vdash \Pi_{x:\sigma_1}\sigma_2 : \mathsf{Ref}(A_1 \to A_2)}$$

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)}$$

  - Well-typed refined terms $\Gamma \vdash t : \sigma$, e.g.:

$$\frac{\Gamma \vdash t : \sigma \qquad |\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}{\Gamma \vdash t : \{x : \sigma \mid \varphi\}}$$

Denney '98

Refinement types

- The style of ref. types we work with (no effects for time being):
  - λ-calculus with types $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2$
  - Refinement types $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid \{x : \sigma \mid \varphi\}$
  - Well-formed refinement types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$, e.g.:

$$\frac{\vdash \Gamma \ \ \mathsf{wf}}{\Gamma \vdash \alpha : \mathsf{Ref}(\alpha)} \qquad \frac{\Gamma \vdash \sigma_1 : \mathsf{Ref}(A_1) \qquad \Gamma, x : \sigma_1 \vdash \sigma_2 : \mathsf{Ref}(A_2)}{\Gamma \vdash \Pi_{x:\sigma_1}\sigma_2 : \mathsf{Ref}(A_1 \to A_2)}$$

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)}$$

  - Well-typed refined terms $\Gamma \vdash t : \sigma$, e.g.:

$$\frac{\Gamma \vdash t : \sigma \qquad |\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}{\Gamma \vdash t : \{x : \sigma \mid \varphi\}}$$

Denney '98

Algebraic effects

- Let's look at effects algebraically (for example: state)
- Types (sets) of values (countable) and locations (fin.): Val, Loc
- Operation symbols:
  - lookup : Loc $\to$ Val
  - update : Loc, Val $\to 1$
- Enforce equations on derived terms:
  - $\mathsf{update}_{l,v}(\mathsf{lookup}_l(x.t)) = \mathsf{update}_{l,v}(t[v/x])$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l,v'}(t)) = \mathsf{update}_{l,v'}(t)$
  - $t = \mathsf{lookup}_l(x.\mathsf{update}_{l,x}(t))$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l',v'}(t)) = \mathsf{update}_{l',v'}(\mathsf{update}_{l,v}(t)) \qquad (l \neq l')$
  - ...

Plotkin & Power '02

- Your usual monad through free algebra construction:
  - $T = UF = (\mathsf{Val}^{\mathsf{Loc}} \times -)^{\mathsf{Val}^{\mathsf{Loc}}}$

Algebraic effects

- Let's look at effects algebraically (for example: state)
- Types (sets) of values (countable) and locations (fin.): $\mathsf{Val}, \mathsf{Loc}$
- Operation symbols:
  - $\mathsf{lookup} : \mathsf{Loc} \to \mathsf{Val}$
  - $\mathsf{update} : \mathsf{Loc}, \mathsf{Val} \to 1$
- Enforce equations on derived terms:
  - $\mathsf{update}_{l,v}(\mathsf{lookup}_l(x.t)) = \mathsf{update}_{l,v}(t[v/x])$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l,v'}(t)) = \mathsf{update}_{l,v'}(t)$
  - $t = \mathsf{lookup}_l(x.\mathsf{update}_{l,x}(t))$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l',v'}(t)) = \mathsf{update}_{l',v'}(\mathsf{update}_{l,v}(t)) \qquad (l \neq l')$
  - ...

Plotkin & Power '02

- Your usual monad through free algebra construction:
  - $T = UF = (\mathsf{Val}^{\mathsf{Loc}} \times -)^{\mathsf{Val}^{\mathsf{Loc}}}$

Algebraic effects

- Let's look at effects algebraically (for example: state)
- Types (sets) of values (countable) and locations (fin.): Val, Loc
- Operation symbols:
  - lookup : Loc $\to$ Val
  - update : Loc, Val $\to 1$
- Enforce equations on derived terms:
  - $\mathsf{update}_{l,v}(\mathsf{lookup}_l(x.t)) = \mathsf{update}_{l,v}(t[v/x])$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l,v'}(t)) = \mathsf{update}_{l,v'}(t)$
  - $t = \mathsf{lookup}_l(x.\mathsf{update}_{l,x}(t))$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l',v'}(t)) = \mathsf{update}_{l',v'}(\mathsf{update}_{l,v}(t)) \qquad (l \neq l')$
  - ...

Plotkin & Power '02

- Your usual monad through free algebra construction:
  - $T = UF = (\mathsf{Val}^{\mathsf{Loc}} \times -)^{\mathsf{Val}^{\mathsf{Loc}}}$

Algebraic effects

- Let's look at effects algebraically (for example: state)
- Types (sets) of values (countable) and locations (fin.): $\mathsf{Val}, \mathsf{Loc}$
- Operation symbols:
  - $\mathsf{lookup} : \mathsf{Loc} \to \mathsf{Val}$
  - $\mathsf{update} : \mathsf{Loc}, \mathsf{Val} \to 1$
- Enforce equations on derived terms:
  - $\mathsf{update}_{l,v}(\mathsf{lookup}_l(x.t)) = \mathsf{update}_{l,v}(t[v/x])$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l,v'}(t)) = \mathsf{update}_{l,v'}(t)$
  - $t = \mathsf{lookup}_l(x.\mathsf{update}_{l,x}(t))$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l',v'}(t)) = \mathsf{update}_{l',v'}(\mathsf{update}_{l,v}(t)) \qquad (l \neq l')$
  - ...

Plotkin & Power '02

- Your usual monad through free algebra construction:
  - $T = UF = (\mathsf{Val}^{\mathsf{Loc}} \times -)^{\mathsf{Val}^{\mathsf{Loc}}}$

Algebraic effects

- Let's look at effects algebraically (for example: state)
- Types (sets) of values (countable) and locations (fin.): $\mathsf{Val}, \mathsf{Loc}$
- Operation symbols:
  - $\mathsf{lookup} : \mathsf{Loc} \to \mathsf{Val}$
  - $\mathsf{update} : \mathsf{Loc}, \mathsf{Val} \to 1$
- Enforce equations on derived terms:
  - $\mathsf{update}_{l,v}(\mathsf{lookup}_l(x.t)) = \mathsf{update}_{l,v}(t[v/x])$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l,v'}(t)) = \mathsf{update}_{l,v'}(t)$
  - $t = \mathsf{lookup}_l(x.\mathsf{update}_{l,x}(t))$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l',v'}(t)) = \mathsf{update}_{l',v'}(\mathsf{update}_{l,v}(t)) \qquad (l \neq l')$
  - ...

Plotkin & Power '02

- Your usual monad through free algebra construction:
  - $T = UF = (\mathsf{Val}^{\mathsf{Loc}} \times -)^{\mathsf{Val}^{\mathsf{Loc}}}$

The programming language

- We use a variant of the Effect Calculus (closely related to Call-by-Push-Value) Egger et. al. '09, '12 Levy '01,'04
- Value and computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to \underline{A_2} \mid FA$
  - $\underline{A} ::= \underline{A_1} \times \underline{A_2} \mid A_1 \to \underline{A_2} \mid FA$
- Terms t:

$t ::= x \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_i\, t \mid \lambda x.t \mid t_1(t_2) \mid \mathsf{return}\, t \mid t_1 \,\mathsf{to}\, x.\, t_2 \mid \mathsf{op}_{t_1}(x.t_2)$

- Well-typed terms $\Gamma \vdash t : A$, e.g.:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}\, t : FA} \qquad \frac{\Gamma \vdash t_1 : FA_1 \quad \Gamma, x : A_1 \vdash t_2 : \underline{A_2}}{\Gamma \vdash t_1 \,\mathsf{to}\, x.\, t_2 : \underline{A_2}}$$

$$\frac{\Gamma \vdash t_1 : \beta \quad \Gamma, x : \alpha \vdash t_2 : \underline{A}}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \underline{A}} \ (\mathsf{op} : \beta \to \alpha)$$

The programming language

- We use a variant of the Effect Calculus (closely related to Call-by-Push-Value)

Egger et. al. '09, '12

Levy '01,'04

- Value and computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to \underline{A}_2 \mid F A$
  - $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid F A$
- Terms t:

$t ::= x \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_i\, t \mid \lambda x.t \mid t_1(t_2) \mid \mathsf{return}\, t \mid t_1\, \mathsf{to}\, x.\, t_2 \mid \mathsf{op}_{t_1}(x.t_2)$

- Well-typed terms $\Gamma \vdash t : A$, e.g.:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}\, t : F A} \qquad \frac{\Gamma \vdash t_1 : F A_1 \quad \Gamma, x : A_1 \vdash t_2 : \underline{A}_2}{\Gamma \vdash t_1\, \mathsf{to}\, x.\, t_2 : \underline{A}_2}$$

$$\frac{\Gamma \vdash t_1 : \beta \quad \Gamma, x : \alpha \vdash t_2 : \underline{A}}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \underline{A}} \ (\mathsf{op} : \beta \to \alpha)$$

The programming language

- We use a variant of the Effect Calculus (closely related to Call-by-Push-Value)

Egger et. al. '09, '12

Levy '01,'04

- Value and computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid FA$
  - $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
- Terms t:

$$t ::= x \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_i\, t \mid \lambda x.t \mid t_1(t_2) \mid \mathsf{return}\, t \mid t_1\, \mathsf{to}\, x.\, t_2 \mid \mathsf{op}_{t_1}(x.t_2)$$

- Well-typed terms $\Gamma \vdash t : A$, e.g.:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{return}\, t : FA} \qquad \frac{\Gamma \vdash t_1 : FA_1 \quad \Gamma, x : A_1 \vdash t_2 : \underline{A}_2}{\Gamma \vdash t_1\, \mathsf{to}\, x.\, t_2 : \underline{A}_2}$$

$$\frac{\Gamma \vdash t_1 : \beta \quad \Gamma, x : \alpha \vdash t_2 : \underline{A}}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \underline{A}} \ (\mathsf{op} : \beta \to \alpha)$$

Algebraic effectful reasoning

- This algebraic treatment of effects induces an effectful multi-sorted logic on EC:
  - Value types: $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid FA$
  - Computation types: $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
  - Terms: $t ::=$
    $x \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_i\, t \mid \lambda x.t \mid t_1(t_2) \mid \mathsf{return}\, t \mid t_1\, \mathsf{to}\, x.\, t_2 \mid \mathsf{op}_{t_1}(x.t_2)$
  - Formulas: $\varphi ::= t_1 = t_2 \mid R(\vec{t}) \mid \pi(\vec{t}) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi$
  - Predicates: $\pi ::= X \mid (\vec{x}).\varphi \mid \mu X.\pi \mid \nu X.\pi$
- Allows algebraic effectful reasoning:
  - Reasoning in terms of equivalence classes of computation trees
- Based on the logic of algebraic effects for CBPV

Plotkin & Pretnar '08

Algebraic effectful reasoning

- This algebraic treatment of effects induces an effectful multi-sorted logic on EC:
  - Value types: $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid FA$
  - Computation types: $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
  - Terms: $t ::=$
    $x \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_i\, t \mid \lambda x.t \mid t_1(t_2) \mid \mathsf{return}\, t \mid t_1\, \mathsf{to}\, x.\, t_2 \mid \mathsf{op}_{t_1}(x.t_2)$
  - Formulas: $\varphi ::= t_1 = t_2 \mid R(\vec{t}) \mid \pi(\vec{t}) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi$
  - Predicates: $\pi ::= X \mid (\vec{x}).\varphi \mid \mu X.\pi \mid \nu X.\pi$
- Allows algebraic effectful reasoning:
  - Reasoning in terms of equivalence classes of computation trees
- Based on the logic of algebraic effects for CBPV

Plotkin & Pretnar '08

Algebraic effectful reasoning

- This algebraic treatment of effects induces an effectful multi-sorted logic on EC:
  - Value types: $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid FA$
  - Computation types: $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
  - Terms: $t ::=$
    $x \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_i\, t \mid \lambda x.t \mid t_1(t_2) \mid \mathsf{return}\, t \mid t_1\, \mathsf{to}\, x.\, t_2 \mid \mathsf{op}_{t_1}(x.t_2)$
  - Formulas: $\varphi ::= t_1 = t_2 \mid R(\vec{t}) \mid \pi(\vec{t}) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi$
  - Predicates: $\pi ::= X \mid (\vec{x}).\varphi \mid \mu X.\pi \mid \nu X.\pi$
- Allows algebraic effectful reasoning:
  - Reasoning in terms of equivalence classes of computation trees
- Based on the logic of algebraic effects for CBPV

Plotkin & Pretnar '08

Algebraic effectful reasoning

- This algebraic treatment of effects induces an effectful multi-sorted logic on EC:
  - Value types: $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid F\,A$
  - Computation types: $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid F\,A$
  - Terms: $t ::=$

    $x \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_i\, t \mid \lambda x.t \mid t_1(t_2) \mid \mathsf{return}\, t \mid t_1 \,\mathsf{to}\, x.\, t_2 \mid \mathsf{op}_{t_1}(x.t_2)$
  - Formulas: $\varphi ::= t_1 = t_2 \mid R(\vec{t}) \mid \pi(\vec{t}) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi$
  - Predicates: $\pi ::= X \mid (\vec{x}).\varphi \mid \mu X.\pi \mid \nu X.\pi$
- Allows algebraic effectful reasoning:
  - Reasoning in terms of equivalence classes of computation trees
- Based on the logic of algebraic effects for CBPV

Plotkin & Pretnar '08

Refinement types for effectful computations

- The story is similar to the λ-calc. ref. types $\Gamma \vdash \sigma$: Ref(A)
- We start with EC and its value & computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid FA$
  - $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
- We define the refinement types as:
  - $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid F\sigma \mid \{x : \sigma \mid \varphi\}$
  - $\underline{\tau} ::= \underline{\tau}_1 \times \underline{\tau}_2 \mid \Pi_{x:\sigma}\underline{\tau} \mid F\sigma$
- Notice: no refinements on computation types
  - φ's do not induce subalgebras in general
  - would break the adj. model principle (comp. types as algebras)
- Well-formed ref. types similar to λ-calc wf. ref. types, e.g.:

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)} \qquad \frac{\Gamma \vdash \sigma : \mathsf{Ref}(A)}{\Gamma \vdash F\sigma : \underline{\mathsf{Ref}}(FA)}$$

Refinement types for effectful computations

- The story is similar to the λ-calc. ref. types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$
- We start with EC and its value & computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid F\underline{A}$
  - $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid F A$
- We define the refinement types as:
  - $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid F\underline{\sigma} \mid \{x : \sigma \mid \varphi\}$
  - $\underline{\tau} ::= \underline{\tau}_1 \times \underline{\tau}_2 \mid \Pi_{x:\sigma}\underline{\tau} \mid F\sigma$
- Notice: no refinements on computation types
  - φ's do not induce subalgebras in general
  - would break the adj. model principle (comp. types as algebras)
- Well-formed ref. types similar to λ-calc wf. ref. types, e.g.:

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)} \qquad \frac{\Gamma \vdash \sigma : \mathsf{Ref}(A)}{\Gamma \vdash F\sigma : \underline{\mathsf{Ref}}(FA)}$$

Refinement types for effectful computations

- The story is similar to the λ-calc. ref. types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$
- We start with EC and its value & computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid FA$
  - $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
- We define the refinement types as:
  - $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid F\sigma \mid \{x : \sigma \mid \varphi\}$
  - $\underline{\tau} ::= \underline{\tau}_1 \times \underline{\tau}_2 \mid \Pi_{x:\sigma}\underline{\tau} \mid F\sigma$
- Notice: no refinements on computation types
  - φ's do not induce subalgebras in general
  - would break the adj. model principle (comp. types as algebras)
- Well-formed ref. types similar to λ-calc wf. ref. types, e.g.:

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)} \qquad \frac{\Gamma \vdash \sigma : \mathsf{Ref}(A)}{\Gamma \vdash F\sigma : \underline{\mathsf{Ref}}(FA)}$$

Refinement types for effectful computations

- The story is similar to the λ-calc. ref. types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$
- We start with EC and its value & computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid FA$
  - $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
- We define the refinement types as:
  - $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid F\sigma \mid \{x : \sigma \mid \varphi\}$
  - $\underline{\tau} ::= \underline{\tau}_1 \times \underline{\tau}_2 \mid \Pi_{x:\sigma}\underline{\tau} \mid F\sigma$
- Notice: no refinements on computation types
  - φ's do not induce subalgebras in general
  - would break the adj. model principle (comp. types as algebras)
- Well-formed ref. types similar to λ-calc wf. ref. types, e.g.:

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)} \qquad \frac{\Gamma \vdash \sigma : \mathsf{Ref}(A)}{\Gamma \vdash F\sigma : \underline{\mathsf{Ref}}(FA)}$$

Refinement types for effectful computations

- The story is similar to the λ-calc. ref. types $\Gamma \vdash \sigma : \mathsf{Ref}(A)$
- We start with EC and its value & computation types:
  - $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid A_1 \to A_2 \mid F\underline{A}$
  - $\underline{A} ::= \underline{A}_1 \times \underline{A}_2 \mid A_1 \to \underline{A}_2 \mid FA$
- We define the refinement types as:
  - $\sigma ::= \alpha \mid 1 \mid \Sigma_{x:\sigma_1}\sigma_2 \mid \Pi_{x:\sigma_1}\sigma_2 \mid F\sigma \mid \{x : \sigma \mid \varphi\}$
  - $\underline{\tau} ::= \underline{\tau}_1 \times \underline{\tau}_2 \mid \Pi_{x:\sigma}\underline{\tau} \mid F\sigma$
- Notice: no refinements on computation types
  - φ's do not induce subalgebras in general
  - would break the adj. model principle (comp. types as algebras)
- Well-formed ref. types similar to λ-calc wf. ref. types, e.g.:

$$\frac{\Gamma \vdash \sigma : \mathsf{Ref}(A) \qquad \Gamma, x : A \vdash \varphi : \mathsf{prop}}{\Gamma \vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)} \qquad \frac{\Gamma \vdash \sigma : \mathsf{Ref}(A)}{\Gamma \vdash F\sigma : \underline{\mathsf{Ref}}(FA)}$$

Refinement types for effectful computations ctd.

- Well-typed terms follow the adj. model considerations:

$$\frac{\Gamma \vdash t : \sigma \quad |\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}{\Gamma \vdash t : \{x : \sigma \mid \varphi\}} \qquad \frac{\Gamma \vdash t : \{x : \sigma \mid \varphi\}}{|\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash \mathsf{return}\, t : F\sigma} \qquad \frac{\Gamma \vdash t_1 : F\sigma_1 \quad \Gamma, x : \sigma_1 \vdash t_2 : \underline{\tau}}{\Gamma \vdash t_1 \,\mathsf{to}\, x.\, t_2 : \underline{\tau}}$$

$$\frac{\Gamma \vdash t_1 : \beta \quad \Gamma, x : \alpha \vdash t_2 : \underline{\tau}}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \underline{\tau}}$$

- Also, more modular verification rules are derivable, e.g.:

$$\frac{\Gamma \vdash t_1 : \sigma_1 \quad |\sigma_1| = \beta \qquad \Gamma, x : \alpha \vdash t_2 : \sigma_2 \qquad |\sigma| = |\sigma_2| = \underline{A} \qquad \Gamma \vdash \{x : \underline{A} \mid \exists x', x''. x = \mathsf{op}_{x'}(x.x''(x)) \wedge \sigma_1^\circ[x'/x] \wedge \forall x'''. \sigma_2^\circ[x''(x''')/x]\} \sqsubseteq \sigma}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \sigma}$$

Refinement types for effectful computations ctd.

- Well-typed terms follow the adj. model considerations:

$$\frac{\Gamma \vdash t : \sigma \quad |\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}{\Gamma \vdash t : \{x : \sigma \mid \varphi\}} \qquad \frac{\Gamma \vdash t : \{x : \sigma \mid \varphi\}}{|\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash \mathsf{return}\, t : F\sigma} \qquad \frac{\Gamma \vdash t_1 : F\sigma_1 \quad \Gamma, x : \sigma_1 \vdash t_2 : \underline{\tau}}{\Gamma \vdash t_1\, \mathsf{to}\, x.\, t_2 : \underline{\tau}}$$

$$\frac{\Gamma \vdash t_1 : \beta \quad \Gamma, x : \alpha \vdash t_2 : \underline{\tau}}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \underline{\tau}}$$

- Also, more modular verification rules are derivable, e.g.:

$$\frac{\Gamma \vdash t_1 : \sigma_1 \quad |\sigma_1| = \beta \qquad \Gamma, x : \alpha \vdash t_2 : \sigma_2 \qquad |\sigma| = |\sigma_2| = \underline{A} \qquad \Gamma \vdash \{x : \underline{A} \mid \exists x', x''. x = \mathsf{op}_{x'}(x.x''(x)) \wedge \sigma_1^\circ[x'/x] \wedge \forall x'''. \sigma_2^\circ[x''(x''')/x]\} \sqsubseteq \sigma}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \sigma}$$

Refinement types for effectful computations ctd.

- Well-typed terms follow the adj. model considerations:

$$\frac{\Gamma \vdash t : \sigma \quad |\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}{\Gamma \vdash t : \{x : \sigma \mid \varphi\}} \qquad \frac{\Gamma \vdash t : \{x : \sigma \mid \varphi\}}{|\Gamma| \mid \Gamma^\circ \vdash \varphi[|t|/x]}$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash \mathsf{return}\, t : F\sigma} \qquad \frac{\Gamma \vdash t_1 : F\sigma_1 \quad \Gamma, x : \sigma_1 \vdash t_2 : \underline{\tau}}{\Gamma \vdash t_1 \,\mathsf{to}\, x.\, t_2 : \underline{\tau}}$$

$$\frac{\Gamma \vdash t_1 : \beta \quad \Gamma, x : \alpha \vdash t_2 : \underline{\tau}}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \underline{\tau}}$$

- Also, more modular verification rules are derivable, e.g.:

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : \sigma_1 \quad |\sigma_1| = \beta \qquad \Gamma, x : \alpha \vdash t_2 : \sigma_2 \qquad |\sigma| = |\sigma_2| = \underline{A} \\ \Gamma \vdash \{x : \underline{A} \mid \exists x', x''.x = \mathsf{op}_{x'}(x.x''(x)) \wedge \sigma_1^\circ[x'/x] \wedge \forall x'''.\sigma_2^\circ[x''(x''')/x]\} \sqsubseteq \sigma\end{array}}{\Gamma \vdash \mathsf{op}_{t_1}(x.t_2) : \sigma}$$

Examples: communication

- Recall the small state language:
  - induced by the 1-location state theory
  - receive : $1 \to \mathsf{nat}$, send : $\mathsf{nat} \to 1$
- Recall the a grammar of session refinements:
  - $S(A) ::= end(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A) \mid$
    $?(y : \mathsf{nat}).S(A) \mid S_1(B); S_2(A)$
- They are defined as operations on predicates, e.g.:
  - $end(A) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{return}\, x'$
  - $!(x : \mathsf{nat} \mid \varphi).S(A) \stackrel{\text{def}}{=} (x : FA).\exists x', x''.x = \mathsf{send}_{x'}(x'') \wedge$
    $\varphi[x'/x] \wedge (S(A)[x'/x])(x'')$
  - $?(x : \mathsf{nat}.S(A)) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$
    $\forall x''.(S(A)[x''/x])(x'(x''))$
  - $S(A); S(B) \stackrel{\text{def}}{=} \ldots$

Examples: communication

- Recall the small state language:
  - induced by the 1-location state theory
  - receive : $1 \to$ nat , send : nat $\to 1$
- Recall the a grammar of session refinements:
  - $S(A) ::= end(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A) \mid$
    $?(y : \mathsf{nat}).S(A) \mid S_1(B); S_2(A)$
- They are defined as operations on predicates, e.g.:
  - $end(A) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{return}\, x'$
  - $!(x : \mathsf{nat} \mid \varphi).S(A) \stackrel{\text{def}}{=} (x : FA).\exists x', x''.x = \mathsf{send}_{x'}(x'') \wedge$
    $\varphi[x'/x] \wedge (S(A)[x'/x])(x'')$
  - $?(x : \mathsf{nat}.S(A)) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$
    $\forall x''.(S(A)[x''/x])(x'(x''))$
  - $S(A); S(B) \stackrel{\text{def}}{=} \ldots$

Examples: communication

- Recall the small state language:
  - induced by the 1-location state theory
  - receive : $1 \to$ nat , send : nat $\to 1$
- Recall the a grammar of session refinements:
  - $S(A) ::= \mathit{end}(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A) \mid$
    $?(y : \mathsf{nat}).S(A) \mid S_1(B); S_2(A)$
- They are defined as operations on predicates, e.g.:
  - $\mathit{end}(A) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{return}\, x'$
  - $!(x : \mathsf{nat} \mid \varphi).S(A) \stackrel{\text{def}}{=} (x : FA).\exists x', x''.x = \mathsf{send}_{x'}(x'') \wedge \varphi[x'/x] \wedge (S(A)[x'/x])(x'')$
  - $?(x : \mathsf{nat}.S(A)) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge \forall x''.(S(A)[x''/x])(x'(x''))$
  - $S(A); S(B) \stackrel{\text{def}}{=} \ldots$

Examples: communication

- Recall the small state language:
  - induced by the 1-location state theory
  - receive : $1 \to$ nat , send : nat $\to 1$
- Recall the a grammar of session refinements:
  - $S(A) ::= end(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A) \mid$
    $?(y : \mathsf{nat}).S(A) \mid S_1(B); S_2(A)$
- They are defined as operations on predicates, e.g.:
  - $end(A) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{return}\, x'$
  - $!(x : \mathsf{nat} \mid \varphi).S(A) \stackrel{\text{def}}{=} (x : FA).\exists x', x''.x = \mathsf{send}_{x'}(x'') \wedge$
    $\varphi[x'/x] \wedge (S(A)[x'/x])(x'')$
  - $?(x : \mathsf{nat}.S(A)) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$
    $\forall x''.(S(A)[x''/x])(x'(x''))$
  - $S(A); S(B) \stackrel{\text{def}}{=} \ldots$

Examples: communication

- Recall the small state language:
  - induced by the 1-location state theory
  - receive : $1 \to \mathsf{nat}$, send : $\mathsf{nat} \to 1$
- Recall the a grammar of session refinements:
  - $S(A) ::= \mathit{end}(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A) \mid$
    $?(y : \mathsf{nat}).S(A) \mid S_1(B); S_2(A)$
- They are defined as operations on predicates, e.g.:
  - $\mathit{end}(A) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{return}\, x'$
  - $!(x : \mathsf{nat} \mid \varphi).S(A) \stackrel{\text{def}}{=} (x : FA).\exists x', x''.x = \mathsf{send}_{x'}(x'') \wedge \varphi[x'/x] \wedge (S(A)[x'/x])(x'')$
  - $?(x : \mathsf{nat}.S(A)) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge \forall x''.(S(A)[x''/x])(x'(x''))$
  - $S(A); S(B) \stackrel{\text{def}}{=} \ldots$

Examples: communication

- Recall the small state language:
  - induced by the 1-location state theory
  - receive : $1 \to$ nat , send : nat $\to 1$
- Recall the a grammar of session refinements:
  - $S(A) ::= end(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A) \mid$
    $?(y : \mathsf{nat}).S(A) \mid S_1(B); S_2(A)$
- They are defined as operations on predicates, e.g.:
  - $end(A) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{return}\, x'$
  - $!(x : \mathsf{nat} \mid \varphi).S(A) \stackrel{\text{def}}{=} (x : FA).\exists x', x''.x = \mathsf{send}_{x'}(x'') \wedge$
    $\varphi[x'/x] \wedge (S(A)[x'/x])(x'')$
  - $?(x : \mathsf{nat}.S(A)) \stackrel{\text{def}}{=} (x : FA).\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$
    $\forall x''.(S(A)[x''/x])(x'(x''))$
  - $S(A); S(B) \stackrel{\text{def}}{=} \ldots$

Examples: communication

- Recall the small state language:
  - induced by the 1-location state theory
  - receive : $1 \to \mathsf{nat}$, send : $\mathsf{nat} \to 1$
- Recall the a grammar of session refinements:
  - $S(A) ::= end(A) \mid !(x : \mathsf{nat} \mid \varphi).S(A) \mid$
    $?(y : \mathsf{nat}).S(A) \mid S_1(B); S_2(A)$
- They are defined as operations on predicates, e.g.:
  - $end(A) \stackrel{\mathsf{def}}{=} (x : FA).\exists x'.x = \mathsf{return}\, x'$
  - $!(x : \mathsf{nat} \mid \varphi).S(A) \stackrel{\mathsf{def}}{=} (x : FA).\exists x', x''.x = \mathsf{send}_{x'}(x'') \wedge$
    $\varphi[x'/x] \wedge (S(A)[x'/x])(x'')$
  - $?(x : \mathsf{nat}.S(A)) \stackrel{\mathsf{def}}{=} (x : FA).\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$
    $\forall x''.(S(A)[x''/x])(x'(x''))$
  - $S(A); S(B) \stackrel{\mathsf{def}}{=} \ldots$

Examples: state

- Recall the small state language:
  - induced by the 1-location state theory
  - lookup : $1 \to$ nat , update : nat $\to 1$
- Formulas φ_P and φ_Q on states (on natural numbers)
- The pre- & post-condition spec.:

$$\forall \vec{x}.\{(x_0).\varphi_P\}y : A\{(x_1).\varphi_Q\} \stackrel{\text{def}}{=}$$

$$(x : FA).(\forall \vec{x'}.\forall x_s.\, \pi_P[\vec{x'}/\vec{x}, x_s/x_0] \implies \pi_{aux}(\vec{x'}, x_s, x_s, x))$$

where (for total correctness)

$$\pi_{aux} \stackrel{\text{def}}{=} \mu X.((\vec{x}, x_0, x_1, x).$$

$$\Big(\exists y.x = \text{return}\, y \wedge \varphi_Q\Big)$$

$$\vee \Big(\exists x'.x = \text{lookup}(x.x'(x)) \wedge X\big(\vec{x}, x_0, x_1, x'(x_1)\big)\Big)$$

$$\vee \Big(\exists x', x''.x = \text{update}_{x'}(x'') \wedge X\big(\vec{x}, x_0, x', x''\big)\Big))$$

Examples: state

- Recall the small state language:
  - induced by the 1-location state theory
  - lookup : $1 \to$ nat , update : nat $\to 1$
- Formulas φ_P and φ_Q on states (on natural numbers)
- The pre- & post-condition spec.:

$$\forall \vec{x}.\{(x_0).\varphi_P\}y : A\{(x_1).\varphi_Q\} \stackrel{\text{def}}{=}$$

$$(x : FA).(\forall \vec{x'}.\forall x_s.\, \pi_P[\vec{x'}/\vec{x}, x_s/x_0] \implies \pi_{aux}(\vec{x'}, x_s, x_s, x))$$

where (for total correctness)

$$\pi_{aux} \stackrel{\text{def}}{=} \mu X.((\vec{x}, x_0, x_1, x).$$

$$\left(\exists y.x = \text{return}\, y \wedge \varphi_Q\right)$$

$$\vee \left(\exists x'.x = \text{lookup}(x.x'(x)) \wedge X\left(\vec{x}, x_0, x_1, x'(x_1)\right)\right)$$

$$\vee \left(\exists x', x''.x = \text{update}_{x'}(x'') \wedge X\left(\vec{x}, x_0, x', x''\right)\right))$$

Examples: state

- Recall the small state language:
  - induced by the 1-location state theory
  - lookup : $1 \to$ nat , update : nat $\to 1$
- Formulas φ_P and φ_Q on states (on natural numbers)
- The pre- & post-condition spec.:

$$\forall \vec{x}.\{(x_0).\varphi_P\}y : A\{(x_1).\varphi_Q\} \stackrel{\text{def}}{=}$$
$$(x : FA).(\forall \vec{x'}.\forall x_s.\, \pi_P[\vec{x'}/\vec{x}, x_s/x_0] \implies \pi_{aux}(\vec{x'}, x_s, x_s, x))$$

where (for total correctness)

$$\pi_{aux} \stackrel{\text{def}}{=} \mu X.((\vec{x}, x_0, x_1, x).$$
$$\left(\exists y.x = \mathsf{return}\, y \wedge \varphi_Q\right)$$
$$\vee \left(\exists x'.x = \mathsf{lookup}(x.x'(x)) \wedge X\left(\vec{x}, x_0, x_1, x'(x_1)\right)\right)$$
$$\vee \left(\exists x', x''.x = \mathsf{update}_{x'}(x'') \wedge X\left(\vec{x}, x_0, x', x''\right)\right))$$

Examples: state $\otimes$ communication

- Recall the combined spec. on state & communication:

$$\forall \vec{x}.\{(x_0).\varphi_P\}(S(A) \rightarrowtail x : A)\{(x_1).\varphi_Q\}$$

- How well can we represent it in our ref. ty. system?
- Combining underlying state & comm. calculi is easy:
  - induced by the tensor of effect theories
  - semantics induced similarly (i.e., $T_\otimes = (T_{IO}(\mathsf{Val}^{\mathsf{Loc}} \times -))^{\mathsf{Val}^{\mathsf{Loc}}}$)
- Combining refinement specs.:
  - not so straightforward, no obvious good combinators
  - similarity between ref. specs. and monads
  - $\ldots \vee \Big(\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$

$$\exists Y.\Big(S(x) \iff (?(y : \mathsf{nat}).Y)\Big) \wedge X(\vec{x}, x_0, x_1, x, Y)\Big)$$

Examples: state $\otimes$ communication

- Recall the combined spec. on state & communication:

$$\forall\vec{x}.\{(x_0).\varphi_P\}(S(A) \rightarrowtail x : A)\{(x_1).\varphi_Q\}$$

- How well can we represent it in our ref. ty. system?
- Combining underlying state & comm. calculi is easy:
  - induced by the tensor of effect theories
  - semantics induced similarly (i.e., $T_\otimes = (T_{IO}(\mathsf{Val}^{\mathsf{Loc}} \times -))^{\mathsf{Val}^{\mathsf{Loc}}}$)
- Combining refinement specs.:
  - not so straightforward, no obvious good combinators
  - similarity between ref. specs. and monads
  - $\ldots \vee \Big(\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$

$$\exists Y.\Big(S(x) \Longleftrightarrow (?(y : \mathsf{nat}).Y)\Big) \wedge X(\vec{x}, x_0, x_1, x, Y)\Big)$$

Examples: state $\otimes$ communication

- Recall the combined spec. on state & communication:

$$\forall \vec{x}.\{(x_0).\varphi_P\}(S(A) \rightarrowtail x : A)\{(x_1).\varphi_Q\}$$

- How well can we represent it in our ref. ty. system?
- Combining underlying state & comm. calculi is easy:
  - induced by the tensor of effect theories
  - semantics induced similarly (i.e., $T_\otimes = (T_{IO}(\mathsf{Val}^{\mathsf{Loc}} \times -))^{\mathsf{Val}^{\mathsf{Loc}}}$)
- Combining refinement specs.:
  - not so straightforward, no obvious good combinators
  - similarity between ref. specs. and monads
  - $\ldots \vee \Big(\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$

$$\exists Y.\Big(S(x) \iff (?(y : \mathsf{nat}).Y)\Big) \wedge X(\vec{x}, x_0, x_1, x, Y)\Big)$$

Examples: state $\otimes$ communication

- Recall the combined spec. on state & communication:

$$\forall \vec{x}.\{(x_0).\varphi_P\}(S(A) \rightarrowtail x : A)\{(x_1).\varphi_Q\}$$

- How well can we represent it in our ref. ty. system?
- Combining underlying state & comm. calculi is easy:
  - induced by the tensor of effect theories
  - semantics induced similarly (i.e., $T_\otimes = (T_{IO}(\mathsf{Val}^{\mathsf{Loc}} \times -))^{\mathsf{Val}^{\mathsf{Loc}}}$)
- Combining refinement specs.:
  - not so straightforward, no obvious good combinators
  - similarity between ref. specs. and monads
  - ... $\vee \Big(\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$

$$\exists Y.\Big(S(x) \Longleftrightarrow (?(y : \mathsf{nat}).Y)\Big) \wedge X(\vec{x}, x_0, x_1, x, Y)\Big)$$

Examples: state $\otimes$ communication

- Recall the combined spec. on state & communication:

$$\forall \vec{x}.\{(x_0).\varphi_P\}(S(A) \rightarrowtail x : A)\{(x_1).\varphi_Q\}$$

- How well can we represent it in our ref. ty. system?
- Combining underlying state & comm. calculi is easy:
  - induced by the tensor of effect theories
  - semantics induced similarly (i.e., $T_\otimes = (T_{IO}(\mathsf{Val}^{\mathsf{Loc}} \times -))^{\mathsf{Val}^{\mathsf{Loc}}}$)
- Combining refinement specs.:
  - not so straightforward, no obvious good combinators
  - similarity between ref. specs. and monads
  - ... $\vee \Big(\exists x'.x = \mathsf{receive}(x.x'(x)) \wedge$

$$\exists Y.\Big(S(x) \iff (?(y : \mathsf{nat}).Y)\Big) \wedge X(\vec{x}, x_0, x_1, x, Y)\Big)$$

To sum it up

A computational language with algebraic effects

$+$

– ref. types for general effectful specs.
– using algebraic effectful reasoning

$\subseteq$ $\subseteq$ $\subseteq$

State language | Communication language | Language X

- For the future:
  - allow ref. types in logic?
  - combinations of specs. more painlessly
  - other algebraic machinery (locality, handlers)
  - extension of simple ty. sys. with dependent refs. fibrationally