Algebraic Subtyping for Algebraic Effects and Handlers

Axel Faes
KULeuven

What does this mean?

Algebraic effect handlers

Exception handlers on steroids

Formally model side-effects
(Matija Pretnar, Gordon Plotkin)

Expressions vs computations

```
effect Decide : unit -> bool;;

let choose_all = handler
    | #Decide () k -> k true @ k false
    | val x -> [x];;

with choose_all handle (if #Decide () then 10 else 20)
(* Output: [10; 20] *)
```

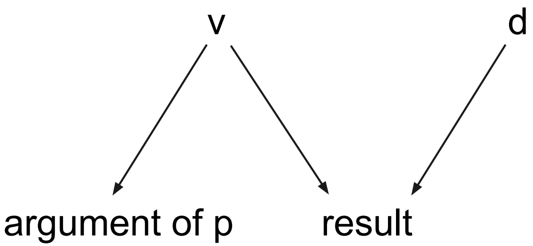
Algebraic subtyping

$\forall \alpha\,.\,(\alpha \rightarrow \mathrm{bool}) \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha$

$(\alpha \rightarrow \mathrm{bool}) \rightarrow \alpha \rightarrow \beta \rightarrow \gamma \mid \alpha \leq \gamma, \beta \leq \gamma$

$\forall \alpha, \beta\,.\,(\alpha \rightarrow \mathrm{bool}) \rightarrow \alpha \rightarrow \beta \rightarrow \alpha \sqcup \beta$

```
let select p v d = if (p v) then v else d
```



Algebraic subtyping

$\alpha \sqcup \beta \equiv \beta \leftrightarrow \alpha \leq \beta$

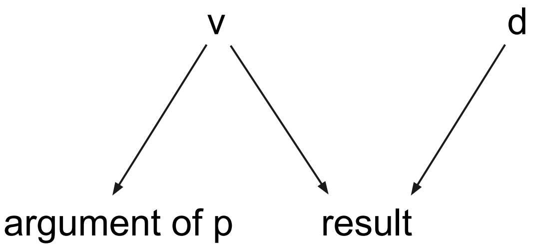
$\alpha \sqcap \beta \equiv \alpha \leftrightarrow \alpha \leq \beta$

$\sqcup$ => outputs

$\sqcap$ => inputs

let-bound vs lambda-bound variables

```
let select p v d = if (p v) then v else d
```

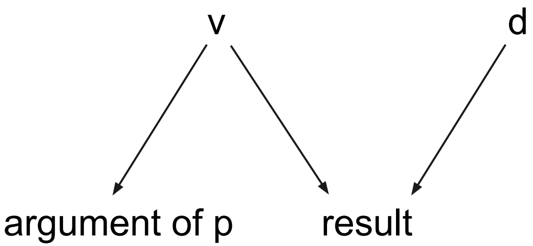


What is the goal?

Algebraic subtyping with effects

let select p v d = if (p v) then v else d

How to represent effects?



What has been done?

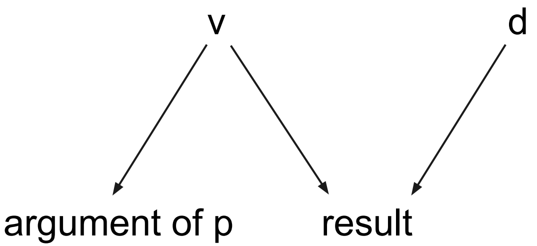
Algebraic subtyping with effects

$$
\begin{aligned}
\text{dirt } \Delta \; ::= \;& \mathsf{Op} \\
\mid \;& \delta \\
\mid \;& \emptyset \\
\mid \;& \Delta_1 \sqcap \Delta_2 \\
\mid \;& \Delta_1 \sqcup \Delta_2
\end{aligned}
$$

$$\Delta_1 \leqslant \Delta_2 \leftrightarrow \Delta_1 \sqcup \Delta_2 \equiv \Delta_2$$

$$\Delta_1 \leqslant \Delta_2 \leftrightarrow \Delta_1 \equiv \Delta_1 \sqcap \Delta_2$$

```
let select p v d = if (p v) then v else d
```



Reformulated typing rules

$$
\begin{array}{lcl}
\Xi & ::= & \epsilon \mid \Xi, x : A \\
\Pi & ::= & \epsilon \mid \Pi, \hat{\mathbf{x}} : [\Xi]A
\end{array}
$$

$$
\textsc{SubVal}\quad \frac{\Pi \Vdash v : [\Xi_1]A_1 \qquad [\Xi_1]A_1 \leqslant^{\forall} [\Xi_2]A_2}{\Pi \Vdash v : [\Xi_2]A_2}
$$

$$
\textsc{Fun}\quad \frac{\Pi \Vdash c : [\Xi, x : A]\underline{C}}{\Pi \Vdash \lambda x.c : [\Xi](A \to \underline{C})}
$$

$$
\textsc{Var-}\Xi\quad \frac{}{\Pi \Vdash x : [x : A]A}
$$

$$
\textsc{Var-}\Pi\quad \frac{(\hat{\mathbf{x}} : [\Xi]A) \in \Pi}{\Pi \Vdash \hat{\mathbf{x}} : [\Xi]A}
$$

Polarity

Positive = output => has union

Negative = input => has intersection

$$\Pi \ ::= \ \epsilon \ | \ \Pi, \hat{\mathbf{x}} : [\Xi^-]A^+$$

$$\begin{array}{lll}
\text{(pure) type } A^+, B^+ & ::= & \texttt{bool} \\
 & | & A^- \to \underline{C}^+
\end{array}$$

$$\begin{array}{lll}
\text{dirt } \Delta^+ & ::= & \texttt{Op} \\
 & | & \delta \\
 & | & \emptyset \\
 & | & \Delta_1^+ \sqcup \Delta_2^+ \\
\text{dirt } \Delta^- & ::= & \texttt{Op} \\
 & | & \delta \\
 & | & \Omega \\
 & | & \Delta_1^- \sqcup \Delta_2^- \\
 & | & \Delta_1^- \sqcap \Delta_2^-
\end{array}$$

BUT:

Negative dirt => also union

Type inference

App

$$\frac{\Pi \triangleright v_1 : [\Xi_1^-]A_1^+ \qquad \Pi \triangleright v_2 : [\Xi_2^-]A_2^+}{\Pi \triangleright v_1\ v_2 : \xi([\Xi_1^- \sqcap \Xi_2^-](\alpha \,!\, \delta))}\ \xi = biunify(A_1^+ \leqslant A_2^+ \to (\alpha \,!\, \delta))$$

Hand

$$\frac{\Pi \Vdash c_r : [\Xi_r^-](B^+ \,!\, \Delta^+) \qquad \left[(\mathsf{Op} : A^+_{\mathsf{Op}} \to B^-_{\mathsf{Op}}) \in \Sigma \qquad \Pi \Vdash c_{\mathsf{Op}} : [\Xi^-_{\mathsf{Op}}](\underline{C}^+_{\mathsf{Op}})\right]_{\mathsf{Op}\in O}}{\Pi \Vdash \{\mathsf{return}\ x \mapsto c_r, [\mathsf{Op}\ y\ k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op}\in O}\} : [\Xi_r^- \sqcap (\bigsqcap \Xi^-_{\mathsf{Op}} | \mathsf{Op} \in O)](\alpha_1 \,!\, \delta_1 \sqcup O \Rightarrow \alpha_2 \,!\, \delta_2)}$$

$$\xi = biunify\left(\begin{array}{l} B^+ \,!\, \Delta^+ \leqslant \alpha_2 \,!\, \delta_2 \\ \alpha_1 \leqslant \Xi_r^-(x) \\ \delta_1 \leqslant \delta_2 \\ \left[\begin{array}{l} A^+_{\mathsf{Op}} \leqslant \Xi^-_{\mathsf{Op}}(y) \\ B^-_{\mathsf{Op}} \to \underline{C}^+_{\mathsf{Op}} \leqslant \Xi^-_{\mathsf{Op}}(k) \\ \underline{C}^+_{\mathsf{Op}} \leqslant \alpha_2 \,!\, \delta_2 \end{array}\right]_{\mathsf{Op}\in O} \end{array}\right)$$

Start

$$biunify(C) = biunify(\emptyset; C)$$

Empty

$$biunify(H; \epsilon) = 1$$

Redundant

$$\frac{c \in H}{biunify(H; c :: C) = biunify(H; C)}$$

Atomic

$$\frac{atomic(c) = \theta_c}{biunify(H; c :: C) = biunify(\theta_c(H \cup \{c\}); \theta_c(C)) \cdot \theta_c}$$

Decompose

$$\frac{subi(c) = C'}{biunify(H; c :: C) = biunify(H \cup \{c\}; C' +\!\!\!+ C)}$$

Implementation

Eff programming language

written in OCaml

Fully featured

Todo: simplification using finite automata

```
and type_expr st {Untyped.term=expr; Untyped.location=loc} = type_plain_e

(* Type a plain expression *)
and type_plain_expr st loc = function
  | Untyped.Var x ->
    let ty_sch, st = get_var_scheme_env ~loc st x in
    Ctor.var ~loc x ty_sch, st
  | Untyped.Const const ->
    Ctor.const ~loc const, st
  | Untyped.Tuple es ->
    let els = List.map (fun (e, _) -> e) (List.map (type_expr st) es) in
    Ctor.tuple ~loc els, st
  | Untyped.Record lst ->
    let lst = List.map (fun (f, (e, _)) -> (f, e)) (Common.assoc_map (typ
    Ctor.record ~loc lst, st
  | Untyped.Variant (lbl, e) ->
    let exp = Common.option_map (fun (e, _) -> e) (Common.option_map (typ
    Ctor.variant ~loc (lbl, exp), st
  | Untyped.Lambda (p, c) ->
    let pat = type_pattern st p in
    let comp, st = type_comp st c in
    Ctor.lambda ~loc pat comp, st
  | Untyped.Effect eff ->
    let eff = infer_effect ~loc st eff in
```

Theory

Proofs
- Instantiation
- Weakening
- Substitution
- Soundness
- Type preservation
- Reformulated typing rules

Validation

Testing against other systems
- Coercion subtyping
- Subtyping
- Row polymorphism

Usecase
- Optimized compilation

Summarize

Algebraic subtyping is very elegant

Separation of inputs/outputs

Dirts are special => union always needed

Intuition:

Algebraic subtyping with effects possible