Algebraic Subtyping for Algebraic Effects and Handlers

Axel Faes
KULeuven

What are algebraic effects and handlers?

Algebraic effects and handlers

Model impure behaviour such as mutable state or I/O [1]

Exceptions on steroids

[1] Pretnar, M., 2015. An introduction to algebraic effects and handlers. Electr. Notes Theor. Comput. Sci, 319, pp.19-35.

Algebraic effects and handlers

Model impure behaviour such as mutable state or I/O [1]

Exceptions on steroids

```
effect Op : unit -> int

let someFun b =
    handle (
        if (b == 0) then
            let a = #Op ()
            print a
        else
            print b
    ) with
        | #Op () cont -> cont 1
```

[1] Pretnar, M., 2015. An introduction to algebraic effects and handlers. Electr. Notes Theor. Comput. Sci, 319, pp.19-35.

Exception handlers + “Continue”

Eff programming language [1]

```
effect Op : unit -> int

let someFun b =
    handle (
        if (b == 0) then
            let a = #Op ()
            print a
        else
            print b
    ) with
        | #Op () cont -> cont 1
```

Functional language

[1] http://www.eff-lang.org/

Eff programming language [1]

```
effect Op : unit -> int

let someFun b =
    handle (
        if (b == 0) then
            let a = #Op ()
            print a
        else
            print b
    ) with
        | #Op () cont -> cont 1
```

Functional language

Algebraic effects and handlers

[1] http://www.eff-lang.org/

Eff programming language [1]

```
effect Op : unit -> int

let someFun b =
    handle (
        if (b == 0) then
            let a = #Op ()
            print a
        else
            print b
    ) with
        | #Op () cont -> cont 1
```

Functional language

Algebraic effects and handlers

ML style language

[1] http://www.eff-lang.org/

Eff programming language [1]

```
effect Op : unit -> int

let someFun b =
    handle (
        if (b == 0) then
            let a = #Op ()
            print a
        else
            print b
    ) with
        | #Op () cont -> cont 1
```

Functional language

Algebraic effects and handlers

ML style language

Type inference

[1] http://www.eff-lang.org/

Type inference

```
let a = 5
```

Functional language

Algebraic effects and handlers

ML style language

Type inference

Type inference

```
let a = 5
```

‘a’ has type ‘int’

Functional language

Algebraic effects and handlers

ML style language

Type inference

Eff programming language [1]

Functional language

Algebraic effects and handlers

ML style language

- Type inference

[1] http://www.eff-lang.org/

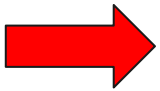
Eff programming language [1]

Functional language

Algebraic effects and handlers

ML style language

Type inference



Problems

[1] http://www.eff-lang.org/

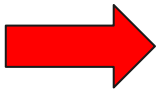
Eff programming language [1]

Functional language

Algebraic effects and handlers

ML style language

Type inference



Problems

Slow to compile

Hard to debug

[1] http://www.eff-lang.org/

Example: Twice

```
let twice f x =
    f (f x)
```

What does this function do?

Example: Twice

```
let twice f x =
    f (f x)
```

What does this function do?

f is a function

Example: Twice

```
let twice f x =
    f (f x)
```

What does this function do?

f is a function
    accepts x and (f x)

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

$x : \alpha$

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

x : α

f : β -> γ

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

$x : \alpha$

$f : \beta \rightarrow \gamma$

$\alpha \leq \beta$

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

$x : \alpha$

$f : \beta \rightarrow \gamma$

$\alpha \leq \beta, \gamma \leq \beta$

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

$x : \alpha$

$f : \beta \to \gamma$

$twice : (\beta \to \gamma) \to \alpha \to \gamma \mid \alpha \leq \beta, \gamma \leq \beta$

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

x : α

f : β -> γ

twice : (β -> γ) -> α -> γ) α ≤ β, γ ≤ β

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

$x : \alpha$

$f : \beta \to \gamma$

$twice : (\beta \to \gamma) \to \alpha \to \gamma \mid \alpha \leq \beta, \gamma \leq \beta$

Example: Twice

```
let twice f x =
    f (f x)
```

Subtyping

x : α

f : β -> γ

twice : (β -> γ) -> α -> γ | α ≤ β, γ ≤ β

If constraints solved:

Get actual type

Problem

https://cdn-images-1.medium.com/max/888/1*XJ7YrEAwzoFSql6HFT4d4g.jpeg

Algebraic Subtyping for Algebraic Effects and Handlers

Dolan, Stephen, and Alan Mycroft. "Polymorphism, subtyping, and type inference in MLsub." In ACM SIGPLAN Notices, vol. 52, no. 1, pp. 60-72. ACM, 2017.

Algebraic Subtyping for Algebraic Effects and Handlers

Stephen Dolan

Dolan, Stephen, and Alan Mycroft. "Polymorphism, subtyping, and type inference in MLsub." In ACM SIGPLAN Notices, vol. 52, no. 1, pp. 60-72. ACM, 2017.

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

Result of f

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

Result of f

    Output of twice

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

Result of f

    Output of twice

    Input of f

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

x : α

return : β

f : α -> ?

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

x : α
return : β
f : α -> α & β

=> accept both α AND β

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

twice : (α -> α & β) -> α -> β

Example: Twice

```
let twice f x =
    f (f x)
```

Algebraic Subtyping

twice : (α -> α & β) -> α -> β

Subtyping:

twice : (β -> γ) -> α -> γ | α ≤ β, γ ≤ β

Effects?

Algebraic effects and handlers

Algebraic subtyping

???

Algebraic subtyping + effects

(pure) type A, B	$::=$	$\texttt{bool}$	bool type
	$\mid$	$A \to \underline{C}$	function type
	$\mid$	$\underline{C} \Rightarrow \underline{D}$	**handler type**
	$\mid$	α	type variable
	$\mid$	$\mu\alpha.A$	recursive type
	$\mid$	$\top$	top
	$\mid$	$\bot$	bottom
	$\mid$	$A \sqcap B$	intersection
	$\mid$	$A \sqcup B$	union
dirty type $\underline{C}, \underline{D}$	$::=$	$A \,!\, \Delta$	

Algebraic subtyping + effects

dirt Δ	$::=$	Op	operation
	$\mid$	δ	dirt variable
	$\mid$	$\emptyset$	empty dirt
	$\mid$	$\Delta_1 \sqcap \Delta_2$	intersection
	$\mid$	$\Delta_1 \sqcup \Delta_2$	union

What are the semantics?

Algebraic subtyping + effects

What are the semantics?

Set operations?

(Op ⊔ Op2) ⊓ (Op ⊔ Op3)

Algebraic subtyping + effects

What are the semantics?

Set operations?

(Op ⊔ Op2) ⊓ (Op ⊔ Op3)

=> Op

Algebraic subtyping + effects

What are the semantics?

Set operations?

(Op ⊔ Op2) ⊓ (Op ⊔ Op3)

=> Op

α & β

accept both α AND β

Algebraic subtyping + effects

What are the semantics?

Set operations?

(Op ⊔ Op2) ⊓ (Op ⊔ Op3)

=> Op

α & β

accept both α AND β

Input vs Output

=> Inputs cause ⊓

=> Outputs cause ⊔

Design of a type-&-effect system

Formulation of typing rules

Define relationship to subtyping

Biunification algorithm (input <-> output)

Type inference algorithm

Keep design close to Dolan’s design

Proofs

Reason to stay close to Dolan

Instantiation
Weakening
Substitution
Soundness
Type preservation

Implementation

Instead of using a ‘toy’ language

Use Eff programming language

Replace the type inference engine

~2700 loc ⇔ ~5700

Implementation

Instead of using a ‘toy’ language

- Use Eff programming language

Replace the type inference engine

- ~2700 loc ⇔ ~5700

Testing against other systems

- Coercion subtyping
- Subtyping
- Row polymorphism

Result

Extension of Algebraic Subtyping
    For algebraic effects and handlers

Type System

Algorithmic

Proofs

Implementation

Simplify constraint generation for types AND EFFECTS by using extended algebraic subtyping

Questions?