Algebraic Subtyping for Algebraic Effects and Handlers

Axel Faes
KULeuven

What are algebraic effects and handlers?

Exception handlers on steroids

Programming language

Programming language

Eff

Exception class in Java

```
class DivisionByZero extends Exception {
	public DivisionByZero() {
		super("Division by zero");
	}
}
```

Defining a new effect

```
class DivisionByZero extends Exception {
    public DivisionByZero() {
        super("Division by zero");
    }
}
```

```
effect DivisionByZero : unit -> empty
```

Throw an exception

```
{
        throw new DivisionByZero();
}
```

Using an effect

```
{
    throw new DivisionByZero();
}
```

#DivisionByZero ()

Throwing an exception

```
public static int divide(int a, int b) {
    if (b == 0) {
        throw new DivisionByZero();
    } else {
        return a / b;
    }
}
```

Using an effect

```
public static int divide(int a, int b) {
    if (b == 0) {
        throw new DivisionByZero();
    } else {
        return a / b;
    }
}
```

```
let divide a b =
    if (b == 0) then
        #DivisionByZero ()
    else
        a / b
```

Exception handlers

```
public static int safeDivide(int a, int b) {
    try {
        return divide(a, b) + 0;
    } catch(DivisionByZero ex) {
        ...
    }
}
```

Effect handlers

```
public static int safeDivide(int a, int b) {
    try {
        return divide(a, b) + 0;
    } catch(DivisionByZero ex) {
        ...
    }
}
```

```
let safeDivide a b =
    handle ((divide a b) + 0) with
        | #DivisionByZero () k -> ...
```

Effect handlers

```
public static int safeDivide(int a, int b) {
    try {
        return divide(a, b) + 0;
    } catch(DivisionByZero ex) {
        ...
    }
}
```

```
let safeDivide a b =
    handle ((divide a b) + 0) with
        | #DivisionByZero () k -> ...
```

Effect handlers

```
public static int safeDivide(int a, int b) {
    try {
        return divide(a, b) + 0;
    } catch(DivisionByZero ex) {
        ...
    }
}
```

```
let safeDivide a b =
    handle ((divide a b) + 0) with
        | #DivisionByZero () k -> ...
```

Continuation

Continuations

Control-flow context

A model of the runtime stack

A model of “the rest of the program”

A program with a hole (written **[ ]**)

```
return [ ] + 0;
```

Exception handlers + Continuations

What is algebraic subtyping?

Subtyping without constraints

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

What is **?**

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

What is **?**

In Java

Could be **Animal**

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

What is **?**

=> constraints

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

What is **?**

=> constraints

Dog ≤ ?, Cat ≤ ?

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

Why so complicated?

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

Why so complicated?

- if statement

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

Why so complicated?

- return dog

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

Why so complicated?

- OR return cat

Subtyping

```
public static ? select(bool b, Dog d, Cat c) {
    if (b) {
        return d;
    } else {
        return c;
    }
}
```

Why so complicated?

So the return type is:

Dog or Cat

The select function with (algebraic) subtyping

Dog ≤ **?**, **Cat** ≤ **?** → **Animal** → **Dog** or **Cat**

What is our goal?

Current situation

Eff programming language

Current situation

Eff programming language

Functional language

Algebraic effect handlers

subtyping

Current situation

Eff programming language

Functional language

Algebraic effect handlers

subtyping

Problem

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

Solution

Remove the constraints

Simplify the constraints

Subtyping → Algebraic subtyping

Simplify the constraints

Subtyping

Algebraic subtyping

Extend algebraic subtyping to algebraic effects and handlers

By example

Effect inference

What should be inferred?

```
public static int test(boolean b) throws .... {
    if (b) {
        throw new IOException();
    } else {
        throw new SQLException();
    }
}
```

Effect inference

```
public static int test(boolean b) throws IOException, SQLException {
    if (b) {
        throw new IOException();
    } else {
        throw new SQLException();
    }
}
```

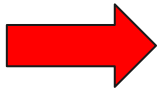
Effect inference

Inference of effects

```
public static int test(boolean b) throws (IOException or SQLException) {
    if (b) {
        throw new IOException();
    } else {
        throw new SQLException();
    }
}
```

Effect inference after handling

```
public static int safeTest(boolean b) {
    try {
        return test(b);
    } catch(IOException ex) {
        ...
    } catch(SQLException ex) {
        ...
    }
}
```



Doesn’t throw exception

Does this work?

Implementation

Eff programming language

Testing against other systems

- Coercion subtyping
- Subtyping
- Row polymorphism

Theory

Algebraic subtyping

Algebraic effects and handlers

Proofs

Simplify constraint generation by using algebraic subtyping

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)
```

What does this function do?

f is a function

accepts x and (f x)

Let's call:

type x = a

type f = a -> ?

type twice = (a -> ?) -> a -> b

Example: Twice

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

What does this function do?

f is a function

accepts x and (f x)

Let's call:

type x = a

type f = a -> ?

type twice = (a -> ?) -> a -> b

? = a & b

The select function

```
public static ?1 select(?2 p, ?3 v, ?4 d) {
    if (p(v)) {
        return v;
    } else {
        return d;
    }
}
```

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

The select function in ML

```
public static ?1 select(?2 p, ?3 v, ?4 d) {
    if (p(v)) {
        return v;
    } else {
        return d;
    }
}
```

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

$?_1 = ?_3 = ?_4$
$?_2 = ?_1 \rightarrow \text{bool}$

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

The select function with subtyping

```
public static ?1 select(?2 p, ?3 v, ?4 d) {
    if (p(v)) {
        return v;
    } else {
        return d;
    }
}
```

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

$(\alpha \to \text{bool}) \to \alpha \to \beta \to \gamma \mid \alpha \leq \gamma, \beta \leq \gamma$

The select function with subtyping

```
public static ?1 select(?2 p, Dog v, Cat d) {
    if (p(v)) {
        return v;
    } else {
        return d;
    }
}
```

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

$\mathbf{Dog} \leq ?_1$, $\mathbf{Cat} \leq ?_1$

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

The select function with subtyping

```
public static Animal select(?2 p, Dog v, Cat d) {
    if (p(v)) {
        return v;
    } else {
        return d;
    }
}
```

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

Dog ≤ **Animal**, **Cat** ≤ **Animal**

α ≤ γ, β ≤ γ

The select function with (algebraic) subtyping

Dog $\leq ?_1$, **Cat** $\leq ?_1$ → **Animal** → **Dog** or **Cat**

The select function with algebraic subtyping

```
public static (Dog or Cat)
  select(? p, Dog v, Cat d) {
        if (p(v)) {
              return v;
        } else {
              return d;
        }
}
```

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

$(\alpha \to \text{bool}) \to \alpha \to \beta \to \alpha \sqcup \beta$

What is the goal?

Algebraic subtyping with effects

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

$\forall \alpha, \beta . (\alpha \to \text{bool}) \to \alpha \to \beta \to \alpha \sqcup \beta$

Add effect information

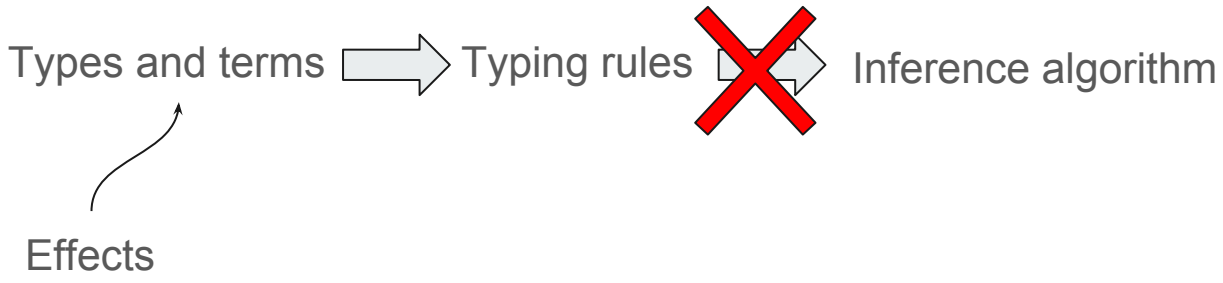
How do you build a type system?

Algebraic subtyping with effect

Types and terms ⇨ Typing rules ⇨ Inference algorithm

Effects

Algebraic subtyping with effect



Algebraic subtyping with effect

Types and terms → Typing rules → Transformations → Inference algorithm

Effects

Transformations

Typing rules → Reformulate Typing rules → Add polarity → Inference algorithm

Transformations

What inference algorithm?

Transformations

What inference algorithm?

Hindley-Milner type inference (with minor changes)

What have I done?

Done

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
```

TODO

Theory

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

TODO

Validation

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

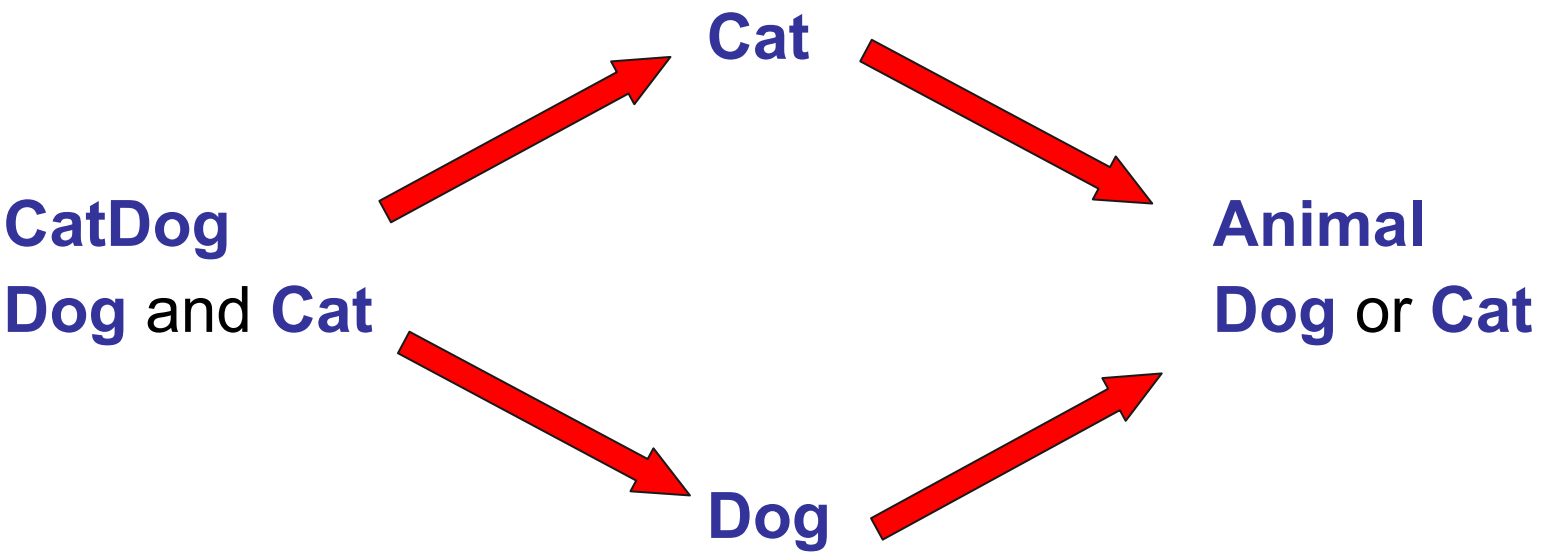
Usecase
- Optimized compilation

Summarize

Algebraic effects and handlers

Algebraic subtyping

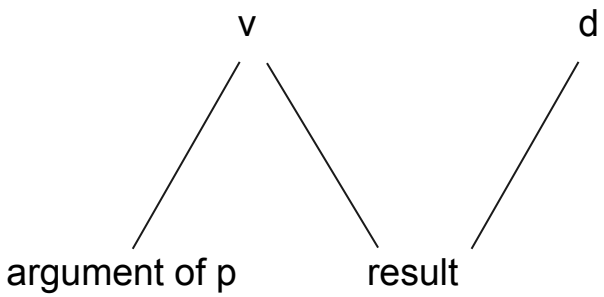
The select function with (algebraic) subtyping



The select function in ML

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

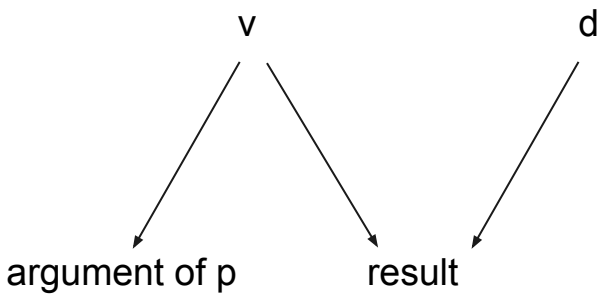
$\forall \alpha \,.\, (\alpha \to \text{bool}) \to \alpha \to \alpha \to \alpha$



The select function with Subtyping

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

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



The select function with Subtyping

$\alpha \leq \gamma$, $\beta \leq \gamma$ → Dog ≤ Animal,
Cat ≤ Animal

The select function with Subtyping

Dog ≤ Animal,
Cat ≤ Animal

Animal is a Dog or a Cat

The select function with algebraic subtyping

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

$\forall \alpha, \beta . (\alpha \to \text{bool}) \to \alpha \to \beta \to \alpha \sqcup \beta$