

When Subtyping Constraints Liberate

- ▶ LIONEL PARREAUX, HKUST, Hong Kong, China
- ▶ ALEKSANDER BORUCH-GRUSZECKI, EPFL, Switzerland
- ▶ ANDONG FAN, HKUST, Hong Kong, China
- ▶ CHUN YIN CHAU, HKUST, Hong Kong, China

Impoverished Type Inference

```
foo f = (f 123, f True)
```

Satisfactory Typing 1

- ▶ should allow

```
foo (fun x => x)
```

- ▶ where

```
(fun x => x) : ALL a . a -> a
```

Satisfactory Typing 2

- ▶ should allow

```
foo (fun x => some x)
```

- ▶ where

```
(fun x => some x) : ALL a . a -> Option a
```

Intersection as Savior

- ▶ multiple application

```
foo f = (f 123, f True)
```

- ▶ suggests intersection in parameter type

```
foo : ALL a b .  
      ((Int -> a) & (Bool -> b)) -> (a,b)
```

Instantiation as Subtyping

- ▶ application

```
foo (fun x => some x)
```

- ▶ generates subtyping constraint to be checked or solved

```
(ALL a . a -> Option a)  
  <:  
((Int -> c) & (Bool -> d))
```

Constrained Variable as Intersection

- ▶ intersection in parameter type

```
ALL a b .  
  ((Int -> a) & (Bool -> b)) -> (a,b)
```

- ▶ is the weakest interpretation of the parameter type in

```
ALL a b c  
  {c <: Int -> a, c <: Bool -> b} .  
  c -> (a,b)
```

Union as Savior

- ▶ branching

```
bar f x = if (f x) then f else (fn x => x)
```

- ▶ suggests union in return type

```
bar : ALL a b .  
      (a & (b -> Bool)) ->  
      b -> (a | (ALL d . d -> d))
```


Constrained Variable as Union

- ▶ union in return type

```
bar : ALL a b .  
      (a & (b -> Bool)) ->  
      b -> (a | (ALL d . d -> d))
```

- ▶ is the strongest interpretation of the return type in

```
bar : ALL a b c {  
      a <: b -> Bool ,  
      a <: c ,  
      (ALL d . d -> d) <: c  
    } . a -> b -> c
```

Restricted User Annotations

- ▶ bounds/intersections are **not** allowed in annotations

```
foo (  
  add : (Int -> Int) & (Str -> Str)  
) : T = ...
```

- ▶ to avoid backtracking search in constraint solving

```
(Int -> Int) & (Str -> Str) <: U
```

Leaky Bound Variable

- ▶ recall

```
foo f = (f 123, f True)
```

- ▶ consider the expression

```
fn x => foo (fn y => x (y, y))
```

- ▶ the inner function's type is generalized

```
(fn y => x (y, y)) :  
ALL b . b -> c
```

- ▶ unsound if bound variable leaks into outer constraint

```
x : a, a <: (b,b) -> c
```

Subtype Extrusion

- ▶ extrude types that are too polymorphic

```
(fn y => x (y, y)) :  
ALL b {b <: b'} . b -> c
```

- ▶ constrain outer param with extruded type

```
a <: (b', b') -> c
```

- ▶ generate instantiated constraints

```
a <: (b', b') -> c, Int <: b', Bool <: b'
```

- ▶ or as union

```
a <: (Int | Bool, Int | Bool) -> c
```

Transitive Closure

- ▶ suppose some constraint has already been found

$L <: a$

- ▶ and a new constraint is discovered

$a <: U$

- ▶ then solve transitive constraint

$L <: U$

Instantiating Left Parametric Type

- ▶ solve constraint with parametric type on the left

$\text{ALL } a \{L <: U\} . T <: V$

- ▶ free the variables and solve apparent constraints

$[a := a'] L <: [a := a'] U, [a := a'] T <: V$

Freezing Right Parametric Type

- ▶ solve constraint with parametric type on the right

$a \rightarrow b <: \text{ALL } c . c \rightarrow c$

- ▶ treat the variable as "skolem"

$c_sk <: a, b <: c_sk$

- ▶ interpret skolem conservatively

$\text{TOP} <: a, b <: \text{BOT}$

Main Ideas

- ▶ use intersection and union to infer satisfactory types
- ▶ limit intersection/union to negative/positive positions
- ▶ solve for variable bounds in subtyping constraints