Row and Bounded Polymorphism via Disjoint Polymorphism

香 港 大 學
THE UNIVERSITY OF HONG KONG

Ningning Xie, Bruno C. d. S. Oliveira, Xuan Bi, Tom Schrijvers

The University of Hong Kong KU Leuven

KU LEUVEN

Introduction

Object-Oriented Languages — Polymorphism, Subtyping

Row Polymorphism — λ|| [Harper and Pierce 1991]

Bounded Polymorphism — Kernel F<: [Cardelli and Wegner 1985]

elaborate

Disjoint Polymorphism

intersection types + Polymorphism

Fi+ [Bi et al. 2019]

Disjoint Polymorphism

TypeScript

```
function
  extend<A, B>(first: A, second: B): A & B
```

intersection type

```
var jim = extend(new Person('Jim'),
                 new ConsoleLogger());
```

Ambiguity

```
extend(new Person('Jim'), new Person('Alice'));
```

Not Type Safe

```
extend(new Person('Jim'), new Pet());
```

name: 'Jim' name: False

Low-level Biased Implementation

Fi+

disjointness

```
let extend A (B * A) (first: A, second: B)
  : A & B
  = first ,, second
```

merge

Row polymorphism via Disjoint Polymorphism

λ|| Row types [Wand 1989] provide an approach to typing extensible records

- Record Concatenation `{name = 'jim' } || {age = 8}  ==  {name = 'jim', age = 8}`
  `{name = 'jim' } || {name = 'Alice'}` ✗
- Compatibility constraint `A # B : A lacks every field contained in B`

record variables, compatibility

```
let extend A (B # A) (first: A, second: B): A || B
  = first || second
```

record concatenation

Challenge

```
λ||  Λ(A # { l : Bool }).
       \(x : A). \(y : { l : Int }).  x || y
```

`A # { l : Bool }` ⟹ `A # { l : int }`

```
Fi+  Λ(A * { l : Bool }).
       \(x : A). \(y : { l : Int }). x ,, y
```

`A * { l : Bool }` $\not\Longrightarrow$ `A * { l : int }`

Solution

1. A simple yet incomplete encoding from *restricted* λ|| into Fi+

   `A # { l : Bool }` $\not\Longrightarrow$ `A # { l : int }`

2. A complete encoding

```
Λ(A1 * ({ l : Bool } & { l : ⊥ }))
  (A2 * ({ l : Bool } & { l : ⊥ })).
   \(x : A1). \(y : { l : Int }). x ,, y
```

Bounded Polymorphism via Disjoint Polymorphism

F<: Bounded quantification [Cardelli and Wegner 1985] integrates *parametric polymorphism* with *subtyping*

- Single-field record
  `\(x : { age : Int}).{ orig = x, age = x.age + 1 }`
- Subtype constraint using bounded polymorphism
  `Λ(A <: { age : Int}). \(x : A). { orig = x, age = x.age + 1 }`
- Subtype constraint using intersection types
  `Λ A. \(x : A & { age : int}). { orig = x, age = x.age + 1 }`

$$\forall (a <: A).\ B \;\triangleq\; \forall a.\ B\,[a \rightsquigarrow a \,\&\, A]$$

Challenge: No Formalization

"This is not, however, an encoding of bounded quantification in a full sense ..." [Pierce 1991]

Solution

Kernel F<: [Cardelli and Wegner 1985] → Fi+ [Bi et al. 2019]

Extra Power of Disjoint Polymorphism

Distributivity and Nested Composition

```
type R[e] = {lit : Int -> e, neg : e -> e}
compose = Λ A (B * A) (r1: R[A], r2: R[B])
  = r1 ,, r2 : R[A & B]
```

Subtyping and row typing

Conclusion

Disjoint polymorphism can recover forms of both row polymorphism and bounded polymorphism

Theoretically A deep understanding and a precise discussion of the relative expressiveness

Practically Economy of the implementation

More In The Paper

- Variants of Row Polymorphism
- Variants of Bounded Polymorphism
- Variants of Intersection Types

Formalization

https://github.com/xnning/Row-and-Bounded-via-Disjoint

Coq Proof Assistant
Type Safety
Coherence

{nnxie, bruno, xbi}@cs.hku.hk tom.schrijvers@cs.kuleuven.be