Type Inference with Polarization: ∀ ± ∃

Ilya Kaysin

Supervisor:
Dr Neel Krishnaswami

UNIVERSITY OF
CAMBRIDGE

Types

```
False          : Bool
239            : Int
[False]        : [Bool]
[2, 3]         : [Int]

length         : [Bool] → Int
```

Types

```
True          : Bool
239           : Int
[False]       : [Bool]
[2, 3]        : [Int]

length        : [Bool] → Int
```

Types

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3]        : [Int]

length        : [Bool] → Int
```

Types

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : [Bool] → Int
```

Types

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : [Bool] → Int


length [True, False] = 2
```

Types

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : [Bool] → Int


length [True, False] = 2

length [2, 3, 9]     = 💀TypeError💀
```

Types

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : [Int] → Bool


length [True, False] = 2

length [2, 3, 9]     = 💀TypeError💀
```

Types

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : [Int] → Bool


length [True, False] = 💀TypeError💀

length [2, 3, 9]     = 3
```

Types: ∀

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : ∀α.[α] → Int


length [True, False] =

length [2, 3, 9]     =
```

Types: ∀

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : ∀α.[α] → Int


length [True, False] = 2

length [2, 3, 9]     = 3
```

Types: ∀

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : ∀α.[α] → Int
```

$\preccurlyeq$ `[Bool] → Int`

$\preccurlyeq$ `[Int] → Int`

```
length [True, False] = 2

length [2, 3, 9]     = 3
```

Types: ∀

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : ∀a.[a] → Int
```

$\preccurlyeq$ `[Bool] → Int`

$\preccurlyeq$ `[Int] → Int`

```
length [True, False] = 2

length [2, 3, 9]     = 3
```

Types: ∀

```
True          : Bool
42            : Int
[False]       : [Bool]
[2, 3, 9]     : [Int]

length        : ∀a.[a] → Int
```

$\preccurlyeq$ `[Bool] → Int`

$\preccurlyeq$ `[Int] → Int`

```
length [True, False] = 2

length [2, 3, 9]     = 3
```

Subtyping (≤)

is the key to type inference

`P ≤ Q` = any `expr:P` is safe to use where `Q` is expected

$$\frac{\text{If: } [T/\alpha]P \le Q}{\text{Then: } \forall\alpha.P \le Q} \qquad \frac{}{P \le P}$$

$$\frac{N \le M \qquad P \ge Q}{P \to N \;\le\; Q \to M} \qquad \frac{P \approx Q}{[P] \le [Q]}$$

Types: ∃

initial state **old** state output **new** state

```
Int × (Int → Bool × Int)
```

Stream of booleans

```
(0,     λs.(odd s, s + 1))
```

F, T, F, T, ...

Types: ∃

initial state **old** state output **new** state

```
Int × (Int → Bool × Int)
```

Stream of booleans

```
(0,     λs.(odd s, s + 1))
```

F, T, F, T, ...

```
(F,     λs.(s, not s))
```

F, T, F, T, ...

```
Bool × (Bool → Bool × Bool)
```

Stream of booleans

Types: ∃

initial state **old** state output **new** state

```
Int × (Int → Bool × Int)
```

Stream of booleans

```
Bool × (Bool → Bool × Bool)
```

Stream of booleans

Types: ∃

initial state **old** state output **new** state

```
Int × (Int → Bool × Int)
```

~~Stream of booleans~~

```
∃α.α × (α → Bool × α)
```

Stream of booleans

```
Bool × (Bool → Bool × Bool)
```

~~Stream of booleans~~

Types: ∃

initial state **old** state output **new** state

```
Int × (Int → Bool × Int)
```

~~Stream of booleans~~

```
∃α.α × (α → Bool × α)
```

Stream of booleans

```
Bool × (Bool → Bool × Bool)
```

~~Stream of booleans~~

Goal: developing a type checking algorithm for polymorphic systems with existential types

Objectives:

1. Develop the declarative system specifying the **decidable segment** of the language
2. Devise a sound and complete **subtyping** algorithm
3. Apply the subtyping for the type **inference**

The Algorithm

$$\frac{?}{\forall a.\ [a] \to \mathtt{Int} \leq [\mathtt{Bool}] \to \mathtt{Int}}$$

Subtyping → Inference

The Algorithm

$$\dfrac{\dfrac{?}{[\hat{\alpha}] \to \texttt{Int} \leq [\texttt{Bool}] \to \texttt{Int}}}{\forall \alpha.\ [\alpha] \to \texttt{Int} \leq [\texttt{Bool}] \to \texttt{Int}}$$

Subtyping → Inference

The Algorithm

$$\dfrac{\dfrac{\dfrac{?}{[\hat{\alpha}] \geq [\mathtt{Bool}]} \qquad \mathtt{Int} \leq \mathtt{Int}}{[\hat{\alpha}] \to \mathtt{Int} \leq [\mathtt{Bool}] \to \mathtt{Int}}}{\forall \alpha.\ [\alpha] \to \mathtt{Int} \leq [\mathtt{Bool}] \to \mathtt{Int}}$$

Subtyping ➙ Inference

The Algorithm

$$\dfrac{\dfrac{\dfrac{\hat{\alpha} \approx \texttt{Bool}}{[\hat{\alpha}] \geq [\texttt{Bool}]} \qquad \texttt{Int} \leq \texttt{Int}}{[\hat{\alpha}] \to \texttt{Int} \leq [\texttt{Bool}] \to \texttt{Int}}}{\forall a.\ [a] \to \texttt{Int} \leq [\texttt{Bool}] \to \texttt{Int}}$$

Unification → Subtyping → Inference

The Algorithm

$$\frac{?}{\forall a.a \to a \to \mathtt{Int} \leq [\mathtt{Bool}] \to [\mathtt{Int}] \to \mathtt{Int}}$$

Unification → **Subtyping** → Inference

The Algorithm

$$\frac{\dfrac{?}{\hat{\alpha}\to\hat{\alpha}\to\texttt{Int} \le [\texttt{Bool}]\to[\texttt{Int}]\to\texttt{Int}}}{\forall\alpha.\alpha\to\alpha\to\texttt{Int} \le [\texttt{Bool}]\to[\texttt{Int}]\to\texttt{Int}}$$

Unification → **Subtyping** → Inference

The Algorithm

$$\dfrac{\dfrac{?}{\hat{\alpha} \geq [\mathtt{Bool}]} \quad \dfrac{?}{\hat{\alpha} \geq [\mathtt{Int}]} \quad \mathtt{Int} \leq \mathtt{Int}}{\hat{\alpha} \to \hat{\alpha} \to \mathtt{Int} \leq [\mathtt{Bool}] \to [\mathtt{Int}] \to \mathtt{Int}}$$

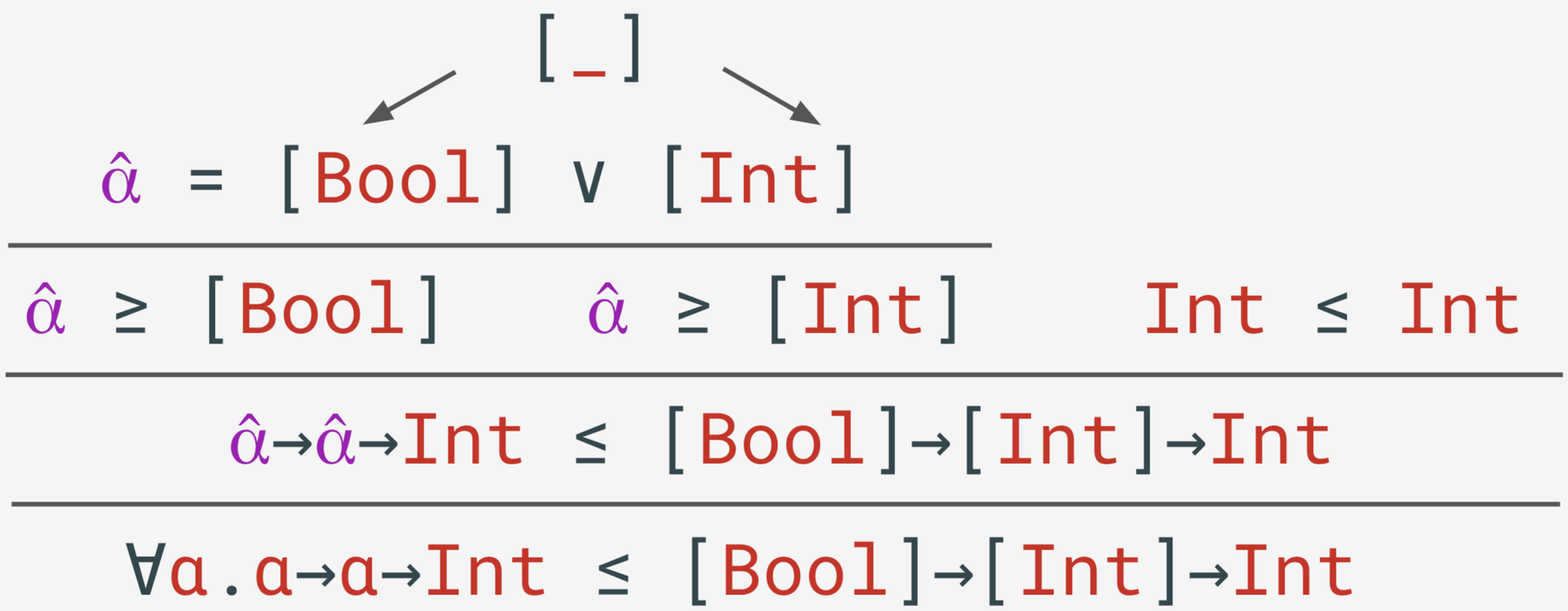
$$\forall \alpha.\, \alpha \to \alpha \to \mathtt{Int} \leq [\mathtt{Bool}] \to [\mathtt{Int}] \to \mathtt{Int}$$

Unification → **Subtyping** → Inference

The Algorithm

$$\frac{\dfrac{\dfrac{\hat{\alpha} = [\mathtt{Bool}] \vee [\mathtt{Int}]}{\hat{\alpha} \geq [\mathtt{Bool}] \quad \hat{\alpha} \geq [\mathtt{Int}] \quad \mathtt{Int} \leq \mathtt{Int}}}{\hat{\alpha} \to \hat{\alpha} \to \mathtt{Int} \leq [\mathtt{Bool}] \to [\mathtt{Int}] \to \mathtt{Int}}}{\forall a.\, a \to a \to \mathtt{Int} \leq [\mathtt{Bool}] \to [\mathtt{Int}] \to \mathtt{Int}}$$

The Algorithm



Anti-unification → P ∨ Q → Subtyping → Inference

Unification → Subtyping → Inference

The Algorithm

$$\dfrac{\dfrac{\dfrac{\dfrac{\exists\beta.[\beta]}{\hat{\alpha} = [\mathtt{Bool}] \vee [\mathtt{Int}]}}{\hat{\alpha} \geq [\mathtt{Bool}] \quad \hat{\alpha} \geq [\mathtt{Int}]} \quad \mathtt{Int} \leq \mathtt{Int}}{\hat{\alpha}\to\hat{\alpha}\to\mathtt{Int} \leq [\mathtt{Bool}]\to[\mathtt{Int}]\to\mathtt{Int}}}{\forall\alpha.\alpha\to\alpha\to\mathtt{Int} \leq [\mathtt{Bool}]\to[\mathtt{Int}]\to\mathtt{Int}}$$

Anti-unification → P ∨ Q → Subtyping → Inference

Unification →

The Algorithm

Does it work? NO!

The Algorithm

Does it work? NO! Unless we restrict the system:

Polarization

Does it work? NO! Unless we restrict the system:

Positive:	α^+	$[P]$	$\exists \alpha^+ . P$	$\downarrow N$
Negative:	α^-	$P \to N$	$\forall \alpha^- . N$	$\uparrow P$

Polarization

Does it work? NO! Unless we restrict the system:

Positive:	α^+	$[P]$	$\exists \alpha^+ . P$	$\downarrow N$
Negative:	α^-	$P \to N$	$\forall \alpha^- . N$	$\uparrow P$

Shifts are invariant:

$$\frac{P \approx Q}{\uparrow P \leq \uparrow Q} \qquad \frac{N \approx M}{\downarrow N \geq \downarrow M}$$

$\hat{\alpha}^{\pm}$ occur only on one side $\Rightarrow$ unification = matching

α^- are "protected" by $\downarrow$ $\Rightarrow$ $N \wedge M$ is not required

Summary

- Inference and ≤ for a large segment of System F with ∃
- Anti-unification *and* unification are applied
- Polarization reveals dualities:

Data	Computations
∃-types	∀-types
Supertypes	Subtypes
Checking	Inference
Anti-Unification	Unification

Plans

- Submit to POPL'24
- Categorization
- Dependent Types
- Mechanization