香 港 大 學
THE UNIVERSITY OF HONG KONG

OOPSLA'24

Imperative Compositional Programming

Type Sound Distributive Intersection Subtyping with References via Bidirectional Typing

Wenjia Ye, Yaozhu Sun, Bruno C. d. S. Oliveira

23 October 2024

Keywords

1. Imperative Compositional Programming
2. Distributive Intersection Subtyping
3. References
4. Type Sound
5. Bidirectional Typing

Compositional Programming

Solving the Expression Problem

Expression

```
type SeqSig<Exp> = {
  Var: String → Exp;
  Ass: String → Exp
              → Exp;
  Seq: Exp → Exp
           → Exp;
}
```

```
type LoopSig<Exp> = {
  WhileDo: Exp → Exp
               → Exp;
  DoWhile: Exp → Exp
               → Exp;
}
```

```
printSeq = trait implements SeqSig<Print> ⇒ {
  (Var     x).print = x;
  (Ass   x e).print = x ++ " = " ++ e.print;
  (Seq e1 e2).print = e1.print ++ "; "
                    ++ e2.print;
}
```

```
printLoop = trait implements LoopSig<Print> ⇒ {
  (WhileDo cond body).print =
      "while (" ++ cond.print ++ ") do { "
                ++ body.print ++ " }";
  (DoWhile body cond).print =
      "do { "  ++ body.print ++ " } while ("
               ++ cond.print ++ ")";
}
```

```
graphSeq = trait
    implements SeqSig<Print ⇒ Graph> ⇒ {
  [self]@(Var x).graph =
      let r = ref (newNode self.print [] [x]) in
      mkPair r r;
  ……
}
```

```
graphLoop = trait implements LoopSig<Graph> ⇒ {
  (WhileDo cond body).graph =
      addAdj (snk cond) [src body] >>
      addAdj (snk cond) [src cond] >>
      mkPair (src cond) (snk cond);
  ……
}
```

```
type Print = { print: String }
```

```
type Graph = { graph: PairRefNode }
```

Dependencies

Operation

Imperative Compositional Programming

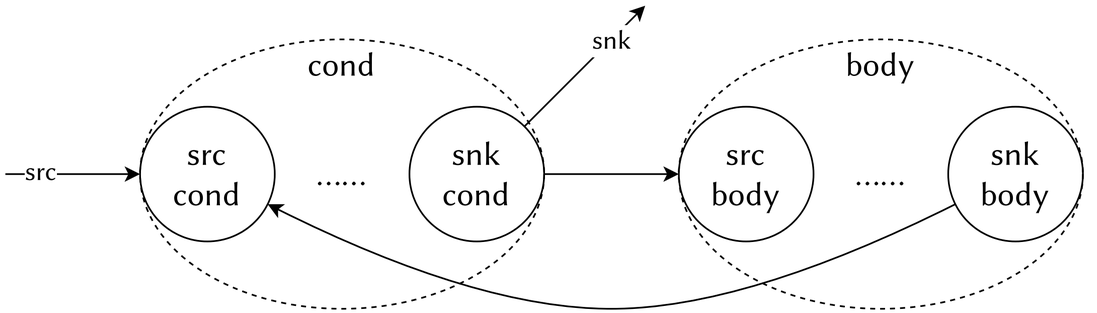
```
type Graph = { graph: PairRefNode };
                    -- (src*,snk*)

type Node = {
  name: String;
  adj: Ref [Ref Node]; -- adjacency array
  ......
};

addAdj (x: Ref Node) (ys: [Ref Node]) =
  !x.adj := !(!x.adj) ++ ys;
```

Dereference

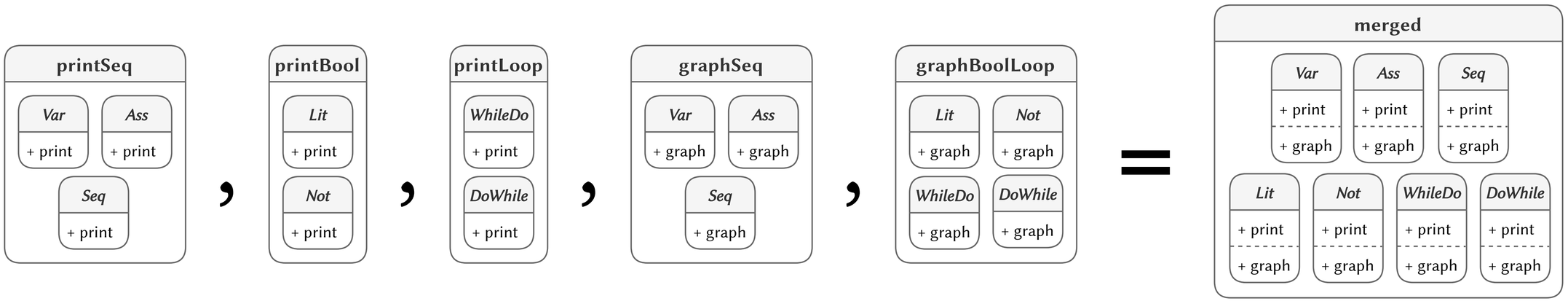
```
graphLoop = trait implements LoopSig<Graph> ⇒ {
  (WhileDo cond body).graph =
      addAdj (snk cond) [src body] >>
      addAdj (snk cond) [src cond] >>
      mkPair (src cond) (snk cond);
  ......
}
```



Nested Trait Composition

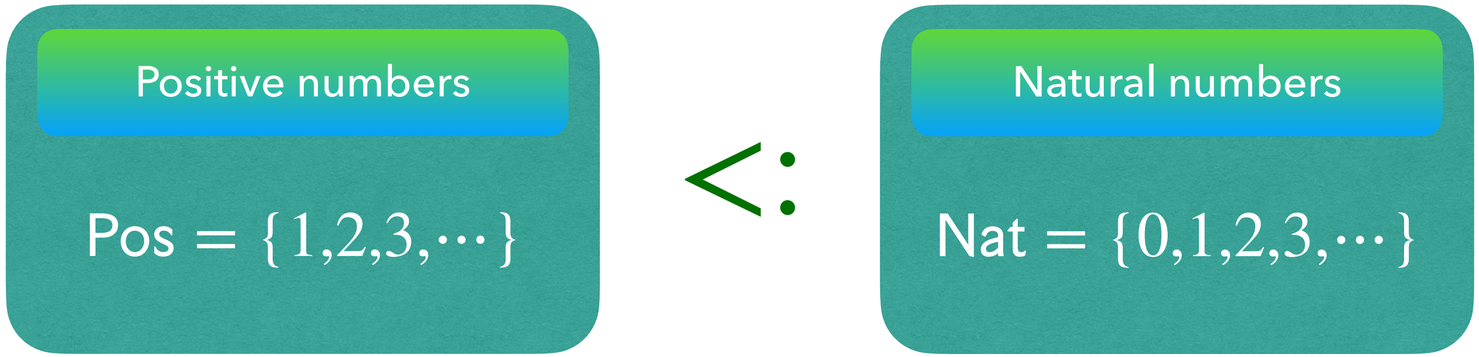
Intersection types

```
merged: SeqSig<Print&Graph> & LoopSig<Print&Graph>
  = new printSeq,graphSeq,printLoop,graphLoop;
```



Intersection Types

$$\frac{e : A \qquad e : B}{e : A \,\&\, B}\ \text{T-Intro}$$



$$\frac{e : A \qquad A <: B}{e : B}\ \text{T-Sub}$$

$$\dfrac{\text{T-Sub}\ \dfrac{1 : \text{Pos} \qquad \text{Pos} <: \text{Nat}}{1 : \text{Nat}} \qquad 1 : \text{Pos}}{1 : \text{Nat} \,\&\, \text{Pos}}\ \text{T-Intro}$$

Intersection Subtyping

$$\frac{}{A \,\&\, B <: A}\ \text{S-AndL} \qquad \frac{}{A \,\&\, B <: B}\ \text{S-AndR} \qquad \frac{A <: B \qquad A <: C}{A <: B \,\&\, C}\ \text{S-And}$$

$$\frac{e : \text{Nat} \,\&\, \text{Pos} \qquad \dfrac{}{\text{Nat} \,\&\, \text{Pos} <: \text{Nat}}\ \text{S-AndL}}{e : \text{Nat}}\ \text{T-Sub} \qquad \frac{e : \text{Nat} \,\&\, \text{Pos} \qquad \dfrac{}{\text{Nat} \,\&\, \text{Pos} <: \text{Pos}}\ \text{S-AndR}}{e : \text{Pos}}\ \text{T-Sub}$$

You can use e as if it simply has type Nat / Pos.

$$\frac{\dfrac{}{\text{Nat} \,\&\, \text{Pos} <: \text{Pos}}\ \text{S-AndR} \qquad \dfrac{}{\text{Nat} \,\&\, \text{Pos} <: \text{Nat}}\ \text{S-AndL}}{\text{Nat} \,\&\, \text{Pos} <: \text{Pos} \,\&\, \text{Nat}}\ \text{S-And}$$

Nat & Pos is equivalent to Pos & Nat.

Reference Types

$$\frac{e : A}{\text{ref } e : \text{Ref } A}\ \text{T-Ref} \qquad \frac{e_1 : \text{Ref } A \qquad e_2 : A}{e_1 := e_2 : ()}\ \text{T-Ass} \qquad \frac{e : \text{Ref } A}{!e : A}\ \text{T-Deref}$$

$$\frac{1 : \text{Pos}}{\text{ref } 1 : \text{Ref Pos}}\ \text{T-Ref}$$

$$\text{T-Sub}\ \frac{x : \text{Ref Pos} \qquad \cancel{\text{Ref Pos} <: \text{Ref Nat}}}{\cancel{x : \text{Ref Nat}}} \Big/ \cancel{(x := 0) : ()}$$

$$\frac{A <: B \qquad B <: A}{\text{Ref } A <: \text{Ref } B}\ \text{S-Ref}$$

[TYPE ERROR]
Subtyping for Ref is invariant!

```
let x = ref 1 : Ref Pos;
x := 0;
!x : Pos
```

Intersection Subtyping with References (1)

$$\frac{e : A}{\mathsf{ref}\ e : \mathsf{Ref}\ A}\ \text{T-Ref} \qquad \frac{e_1 : \mathsf{Ref}\ A \qquad e_2 : A}{e_1 := e_2 : ()}\ \text{T-Ass} \qquad \frac{e : \mathsf{Ref}\ A}{!e : A}\ \text{T-Deref}$$

$$\dfrac{\text{T-Ref}\ \dfrac{1 : \mathsf{Nat}}{\mathsf{ref}\ 1 : \mathsf{Ref}\ \mathsf{Nat}} \qquad \dfrac{1 : \mathsf{Pos}}{\mathsf{ref}\ 1 : \mathsf{Ref}\ \mathsf{Pos}}\ \text{T-Ref}}{\mathsf{ref}\ 1 : \mathsf{Ref}\ \mathsf{Nat} \,\&\, \mathsf{Ref}\ \mathsf{Pos}}\ \text{T-Intro}$$

The definition type-checks!
Initially x points to 1.

```
let x = ref 1 : Ref Nat & Ref Pos;
x := 0;
!x : Pos
```

Intersection Subtyping with References (2)

$$\frac{e : A}{\mathsf{ref}\ e : \mathsf{Ref}\ A}\ \text{T-Ref} \qquad \frac{e_1 : \mathsf{Ref}\ A \qquad e_2 : A}{e_1 := e_2 : ()}\ \text{T-Ass} \qquad \frac{e : \mathsf{Ref}\ A}{!e : A}\ \text{T-Deref}$$

$$\text{T-Ass}\ \frac{\text{T-Sub}\ \dfrac{x : \mathsf{Ref\ Nat\ \&\ Ref\ Pos} \qquad \mathsf{Ref\ Nat\ \&\ Ref\ Pos} <: \mathsf{Ref\ Nat}}{x : \mathsf{Ref\ Nat}} \qquad 0 : \mathsf{Nat}}{(x := 0) : ()}$$

The assignment type-checks!
Now x points to 0.

```
let x = ref 1 : Ref Nat & Ref Pos;
x := 0;
!x : Pos
```

Intersection Subtyping with References (3)

$$\frac{e : A}{\mathsf{ref}\ e : \mathsf{Ref}\ A}\ \text{T-Ref} \qquad \frac{e_1 : \mathsf{Ref}\ A \qquad e_2 : A}{e_1 := e_2 : ()}\ \text{T-Ass} \qquad \frac{e : \mathsf{Ref}\ A}{!e : A}\ \text{T-Deref}$$

$$\dfrac{\dfrac{x : \mathsf{Ref\ Nat\ \&\ Ref\ Pos} \qquad \mathsf{Ref\ Nat\ \&\ Ref\ Pos} <: \mathsf{Ref\ Pos}}{x : \mathsf{Ref\ Pos}}\ \text{T-Sub}}{!x : \mathsf{Pos}}\ \text{T-Deref}$$

[RUNTIME ERROR]
0 does not have type Pos!

```
let x = ref 1 : Ref Nat & Ref Pos;
x := 0;
!x : Pos
```

Type unsound!

Solution by Davies and Pfenning [2000]

$$\frac{e : A \qquad e : B}{e : A \,\&\, B}\ \text{T-Intro}$$

Value restriction

$$\frac{v : A \qquad v : B}{v : A \,\&\, B}\ \text{T-IntroV}$$

$$\frac{\text{T-Ref}\ \dfrac{1 : \text{Nat}}{\text{ref } 1 : \text{Ref Nat}} \qquad \dfrac{1 : \text{Pos}}{\text{ref } 1 : \text{Ref Pos}}\ \text{T-Ref}}{\text{ref } 1 : \text{Ref Nat} \,\&\, \text{Ref Pos}}\ \cancel{\text{T-Intro}}$$

[TYPE ERROR]
"ref 1" is not a value!

```
let x = ref 1 : Ref Nat & Ref Pos;
x := 0;
!x : Pos
```

Type sound!

Distributive Subtyping Breaks Soundness

$$\frac{}{(A \to B)\ \&\ (A \to C) <: A \to B\ \&\ C}\ \text{S-Dist}$$

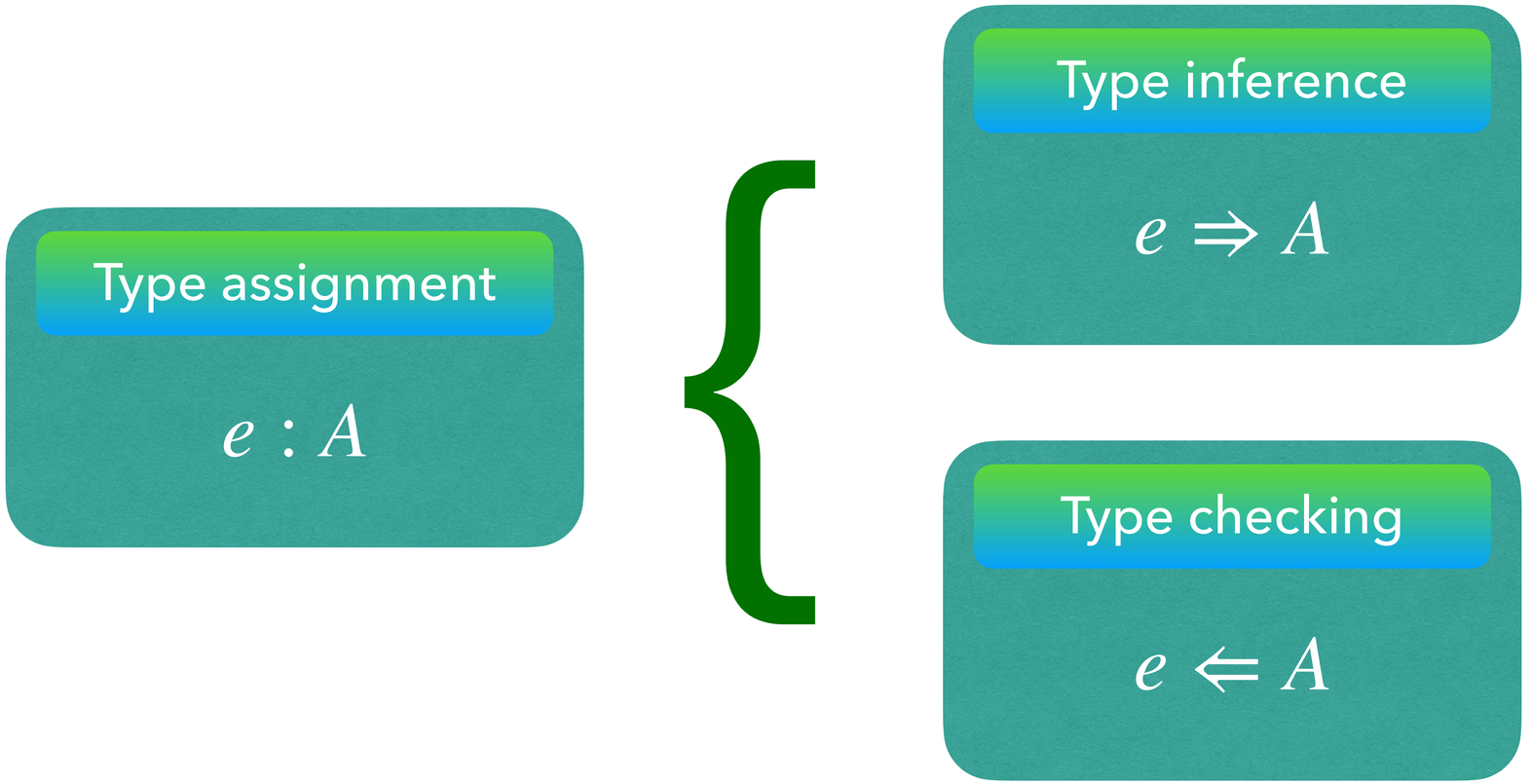
$$\dfrac{\text{T-IntroV}\dfrac{\dfrac{\text{ref } 1 : \text{Ref Nat}}{\lambda x\,.\,\text{ref } 1 : () \to \text{Ref Nat}} \quad \dfrac{\text{ref } 1 : \text{Ref Pos}}{\lambda x\,.\,\text{ref } 1 : () \to \text{Ref Pos}}}{\lambda x\,.\,\text{ref } 1 : (() \to \text{Ref Nat}) \,\&\, (() \to \text{Ref Pos})} \quad \dfrac{}{(() \to \text{Ref Nat}) \,\&\, (() \to \text{Ref Pos}) <: () \to \text{Ref Nat} \,\&\, \text{Ref Pos}}\ \text{S-Dist}}{\lambda x\,.\,\text{ref } 1 : () \to \text{Ref Nat} \,\&\, \text{Ref Pos}}\ \text{T-Sub}$$

The definition type-checks because "λx. ref 1" is a value!

```
let x = (λx. ref 1) () : Ref Nat & Ref Pos;
x := 0;
!x : Pos
```

Type unsound!

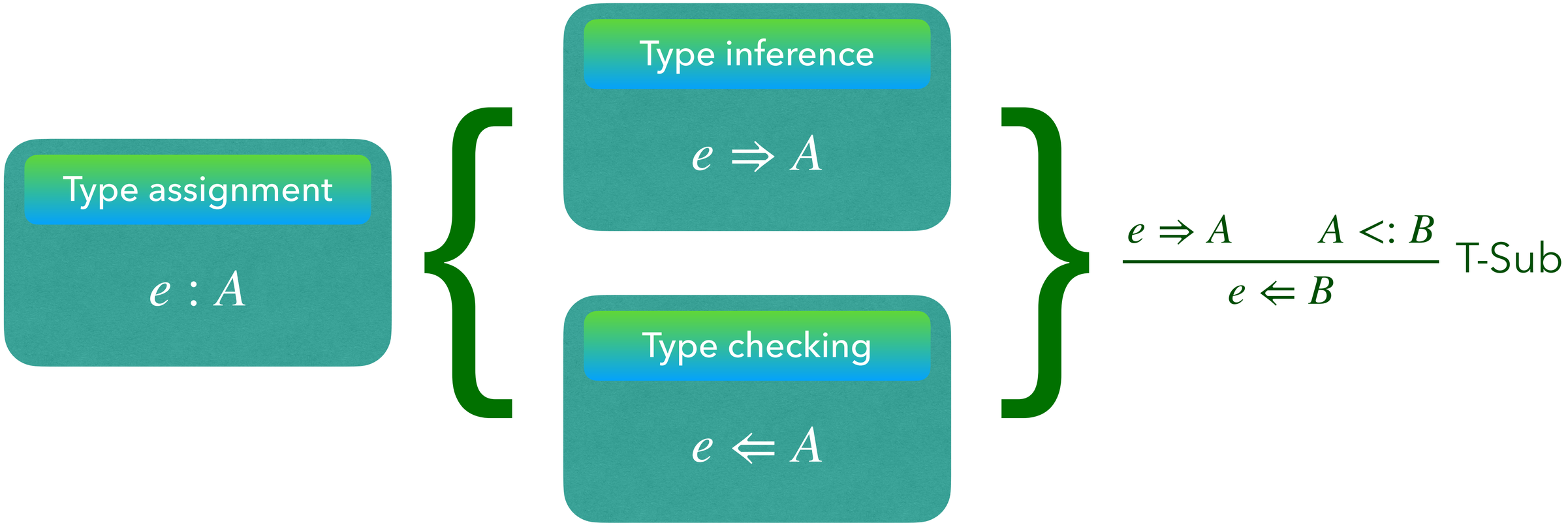
Bidirectional Typing to the Rescue



(usually)
declarative

(usually)
algorithmic

Bidirectional Typing to the Rescue



The bridge between the two typing modes: ***subsumption rule***.

Bidirectional Typing with Numbers

Type assignment (non-bidirectional)

$$\frac{}{1 : \text{Pos}}\ \text{T-Pos} \qquad \frac{1 : \text{Pos} \qquad \text{Pos} <: \text{Nat}}{1 : \text{Nat}}\ \text{T-Sub}$$

Bidirectional typing

$$\frac{}{1 \Rightarrow \text{Pos}}\ \text{T-Pos} \qquad \frac{1 \Rightarrow \text{Pos} \qquad \text{Pos} <: \text{Nat}}{1 \Leftarrow \text{Nat}}\ \text{T-Sub}$$

1 : Pos
✔ infer
✔ check

1 : Nat
✘ infer
✔ check

Bidirectional Typing with References (1)

Type assignment (non-bidirectional)

$$\frac{e : A}{\text{ref } e : \text{Ref } A}\ \text{T-Ref} \qquad \frac{e_1 : \text{Ref } A \quad e_2 \Leftarrow A}{e_1 := e_2 : ()}\ \text{T-Ass} \qquad \frac{e : \text{Ref } A}{!e : A}\ \text{T-Deref}$$

Davies and Pfenning [2000]

$$\frac{e \Leftarrow A}{\text{ref } e \Leftarrow \text{Ref } A}\ \text{T-Ref} \qquad \frac{B \uparrow \text{Ref } A \quad e_1 \Rightarrow B \quad e_2 \Leftarrow A}{e_1 := e_2 : ()}\ \text{T-Ass} \qquad \frac{B \uparrow \text{Ref } A \quad e \Rightarrow B}{!e : A}\ \text{T-Deref}$$

Our approach

$$\frac{e \Rightarrow A}{\text{ref } e \Rightarrow \text{Ref } A}\ \text{T-Ref} \qquad \frac{e_1 \Rightarrow \text{Ref } A \quad e_2 \Leftarrow A}{e_1 := e_2 \Rightarrow ()}\ \text{T-Ass} \qquad \frac{e \Rightarrow \text{Ref } A}{!e \Rightarrow A}\ \text{T-Deref}$$

Bidirectional Typing with References (2)

Our approach

$$\frac{e \Rightarrow A}{\text{ref } e \Rightarrow \text{Ref } A}\ \text{T-Ref}$$

ref 1 : Ref Pos
✔ infer
✔ check

$$\frac{1 \Rightarrow \text{Pos}}{\text{ref } 1 \Rightarrow \text{Ref Pos}}\ \text{T-Ref} \qquad \frac{\text{ref } 1 \Rightarrow \text{Ref Pos} \qquad \text{Ref Pos} <: \text{Ref Pos}}{\text{ref } 1 \Leftarrow \text{Ref Pos}}\ \text{T-Sub}$$

ref 1 : Ref Nat
✘ infer
✘ check

$$\frac{1 \Leftarrow \text{Nat}}{\cancel{\text{ref } 1 \Leftrightarrow \text{Ref Nat}}}\ \text{T-Ref} \qquad \frac{\text{ref } 1 \Rightarrow \text{Ref Pos} \qquad \cancel{\text{Ref Pos} <: \text{Ref Nat}}}{\cancel{\text{ref } 1 \Leftarrow \text{Ref Nat}}}\ \text{T-Sub}$$

Back to the Counterexamples

$$\frac{\cancel{\text{ref } 1 \Leftarrow \text{Ref Nat}} \qquad \text{ref } 1 \Leftarrow \text{Ref Pos}}{\cancel{\text{ref } 1 \Leftarrow \text{Ref Nat} \,\&\, \text{Ref Pos}}}\ \text{T-Intro}$$

$$\frac{\dfrac{\cancel{\text{ref } 1 \Leftarrow \text{Ref Nat}}}{\cancel{\lambda x.\ \text{ref } 1 \Leftarrow () \to \text{Ref Nat}}} \qquad \dfrac{\text{ref } 1 \Leftarrow \text{Ref Pos}}{\lambda x.\ \text{ref } 1 \Leftarrow () \to \text{Ref Pos}}}{\cancel{\lambda x.\ \text{ref } 1 \Leftarrow (() \to \text{Ref Nat}) \,\&\, (() \to \text{Ref Pos})}}\ \text{T-Intro}$$

```
let x = ref 1 : Ref Nat & Ref Pos;
x := 0;
!x : Pos
```

Type sound!

[TYPE ERROR]
"ref 1" does not have type "Ref Nat"

```
let x = (λx. ref 1) () : Ref Nat & Ref Pos;
x := 0;
!x : Pos
```

Type sound!

Wanna have "Ref Nat" instead of "Ref Pos"?

```
let n = 1 : Nat;
let x = ref n : Ref Nat;
.....
```

Advantages over Davies and Pfenning [2000]

①

$$\frac{e \Leftarrow A \qquad e \Leftarrow B}{e \Leftarrow A \,\&\, B}\ \text{T-Intro}$$

Intersection introduction is standard (w/o value restriction).

②

$$\frac{}{(A \to B) \,\&\, (A \to C) <: A \to B \,\&\, C}\ \text{S-Dist}$$

Distributive subtyping is supported.

Why distributivity matters?

Imperative Compositional Programming (ICP)

- Logical implication is left-distributive over conjunction:

$$A \to B \wedge C \quad \Leftrightarrow \quad (A \to B) \wedge (A \to C)$$

- Distributivity of traits in ICP (encoded as functions of records)

  ↳ nested trait composition [Bi et al. 2018]

  ↳ family polymorphism [Ernst 2001]

  ↳ solution to the expression problem [Wadler 1998]

More in the Paper

- ✦ More introduction to ICP and its type-theoretic foundations.
- ✦ Complete code of modular data-flow analysis in ICP.

- ❖ Formalization of λ_{im}, a calculus similar to that by Davies and Pfenning [2000] but employing our solution based on bidirectional typing.
- ❖ Formalization of F^{+}_{im}, the core calculus for ICP.
- ❖ Mechanized proofs of type soundness for both calculi.

Q&A