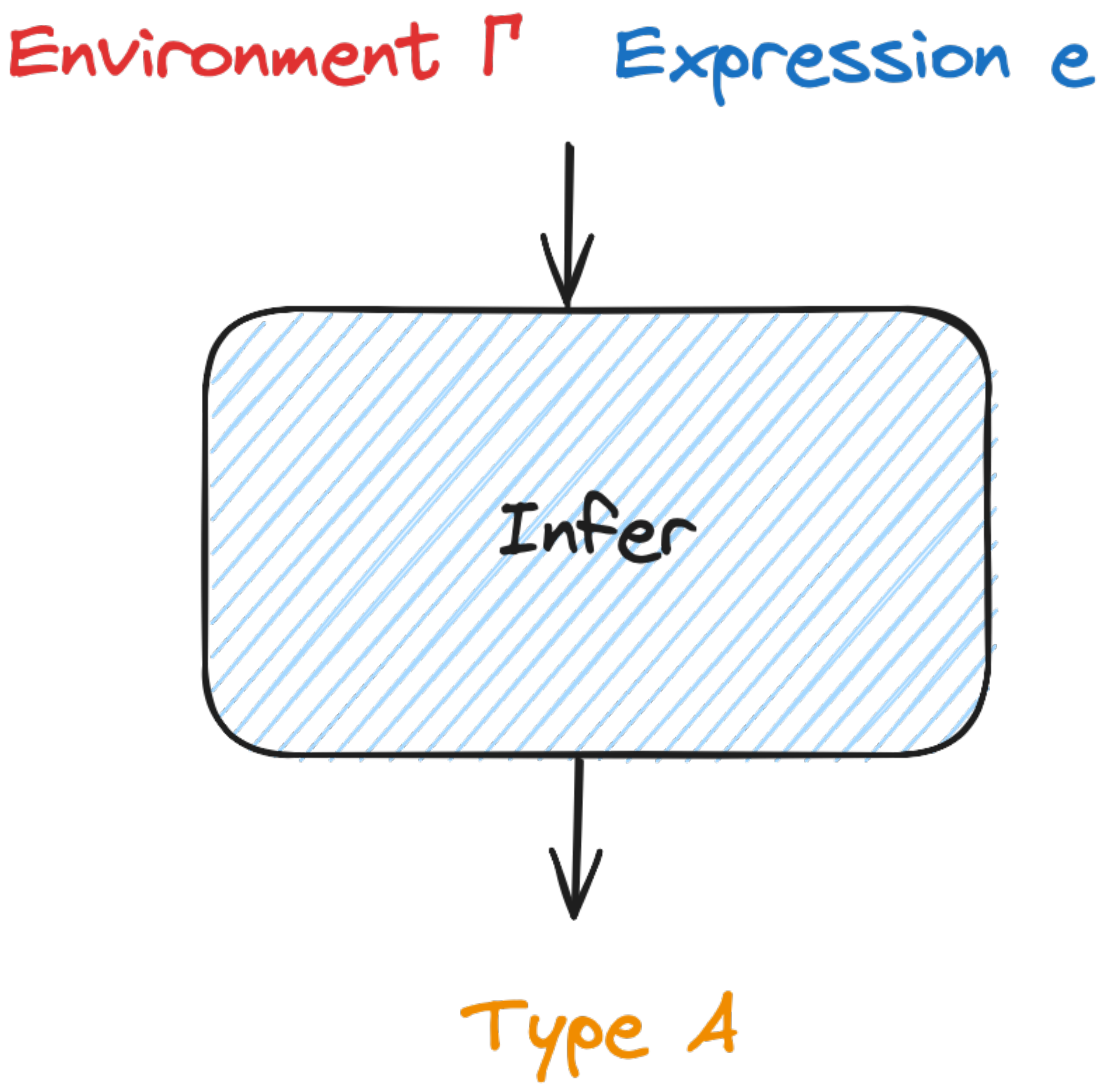
Contextual Typing

Xu Xue and *Bruno C. d. S. Oliveira*

The University of Hong Kong

2024-09-05 @ ICFP

Type Inference techniques and what we believe ...



→ Scalability is necessary; better be lightweight;

→ Reasonable and meaningful annotations are ok;

→ Having guidelines for language implementors and programmers is good;

→ Implementation can be easily derived.

Bidirectional Typing

Why not?

- Merge type inference and type checking by two modes;
- Types are propagated to neighbouring terms;

Inference mode: $\Gamma \vdash e \Rightarrow A$

Checking mode: $\Gamma \vdash e \Leftarrow A$

Bidirectional Typing: Limited Expressive Power or Backtracking

The usual application rule:

$$\frac{\Gamma \vdash e_1 \Rightarrow A \to B \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash e_1\ e_2 \Rightarrow B}\ \textsf{App}$$

The usual subsumption rule:

$$\frac{\Gamma \vdash e \Rightarrow A \qquad A = B}{\Gamma \vdash \boxed{e \Leftarrow B}}\ \textsf{Sub}$$

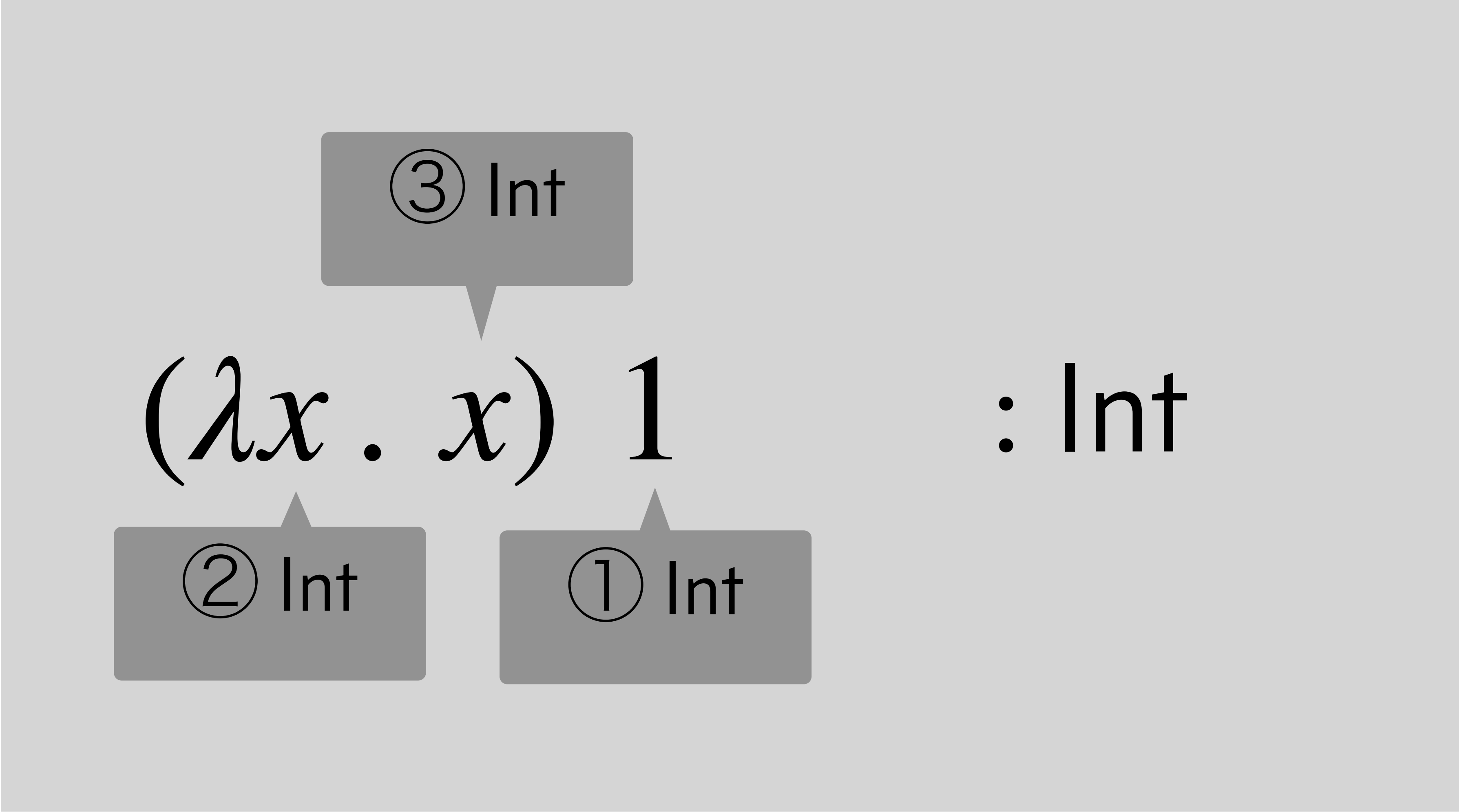
overlapping

The "unusual" application rule:

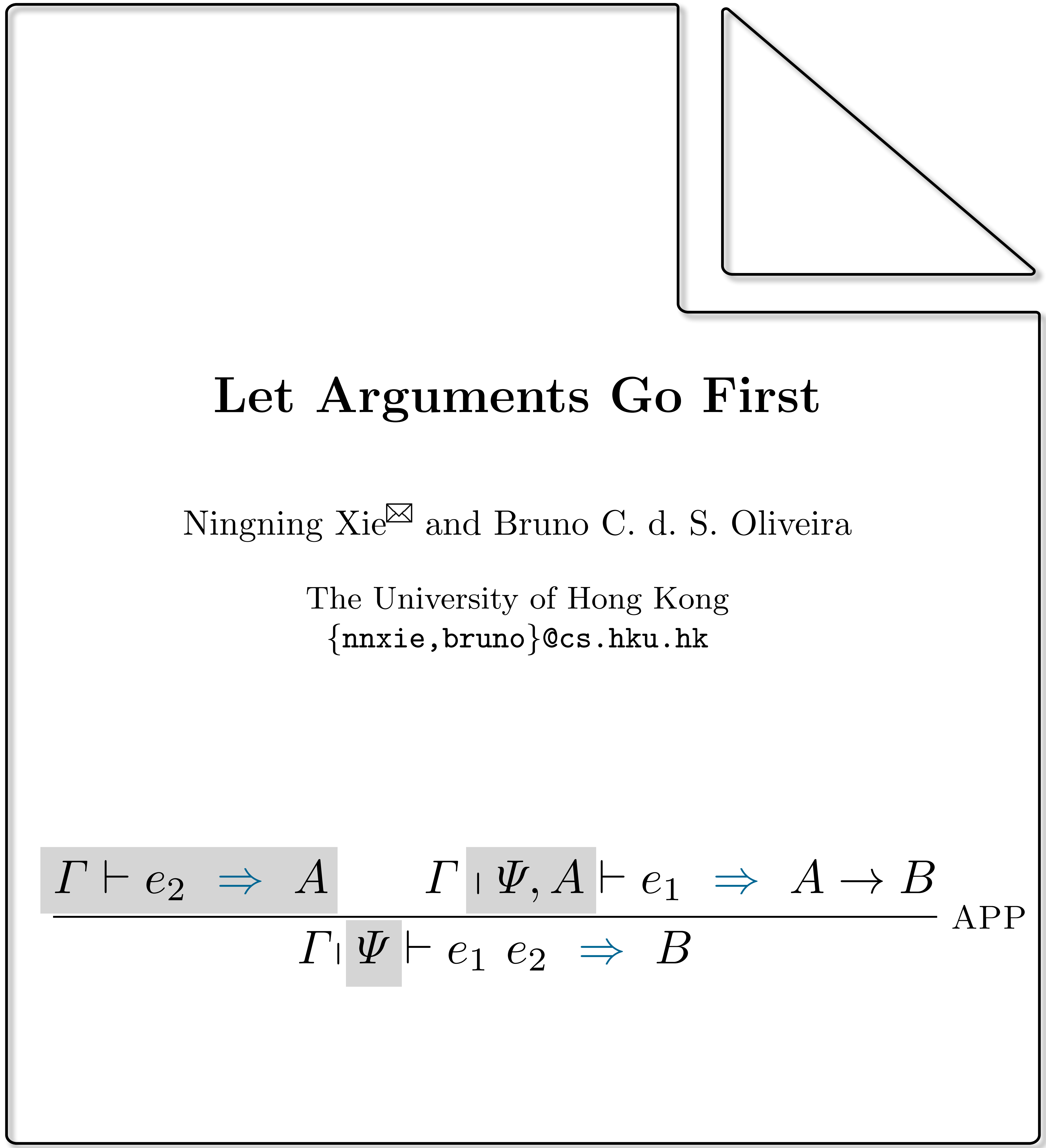
$$\frac{\Gamma \vdash e_2 \Rightarrow A \qquad \Gamma \vdash e_1 \Leftarrow A \to B}{\Gamma \vdash \boxed{e_1\ e_2 \Leftarrow B}}\ \textsf{App2}$$

Example: $\Gamma, z : \texttt{Int} \vdash ((\lambda x.\ x)\ z) : \texttt{Int}$

Bidirectional Typing: Still not enough



causes more overlapping



Let Arguments Go First

Ningning Xie and Bruno C. d. S. Oliveira

The University of Hong Kong
{nnxie,bruno}@cs.hku.hk

$$\frac{\Gamma \vdash e_2 \Rightarrow A \qquad \Gamma \mid \Psi, A \vdash e_1 \Rightarrow A \to B}{\Gamma \mid \Psi \vdash e_1\ e_2 \Rightarrow B}\ \text{APP}$$

Bidirectional Typing: Annotatability and Subsumption

- Informal and unclear annotatability (How to annotate a program);
- Inexpressive subsumption;

$$\frac{\Gamma \vdash e_1 \Rightarrow A \qquad \boxed{A \triangleright B \to C} \qquad \Gamma \vdash e_2 \Leftarrow B}{\Gamma \vdash e_1\ e_2 \Rightarrow C}\ \textsf{App}$$

Intersection Types

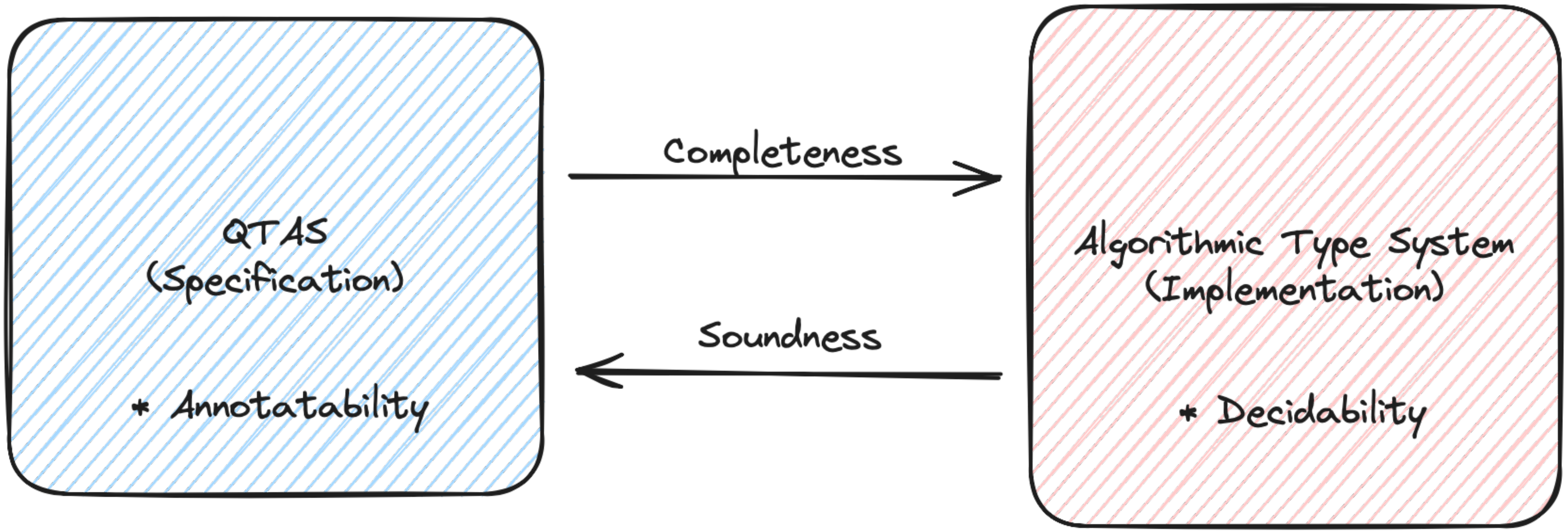
$$\frac{\Psi \vdash e_1 \Rightarrow A \qquad \boxed{\Psi \vdash A \bullet e_2 \Rrightarrow C}}{\Psi \vdash e_1\ e_2 \Rightarrow C}\ \textsf{Decl}{\to}\textsf{E}$$

Polymorphic Types

Our Solution: Contextual Typing

- It's an improvement over bidirectional typing; and it offers
- more expressive power without backtracking;
- easier annotatability guidelines;
- more expressive subsumption.

Our Solution: Contextual Typing



$$\Gamma \vdash_n e : A$$

Counter

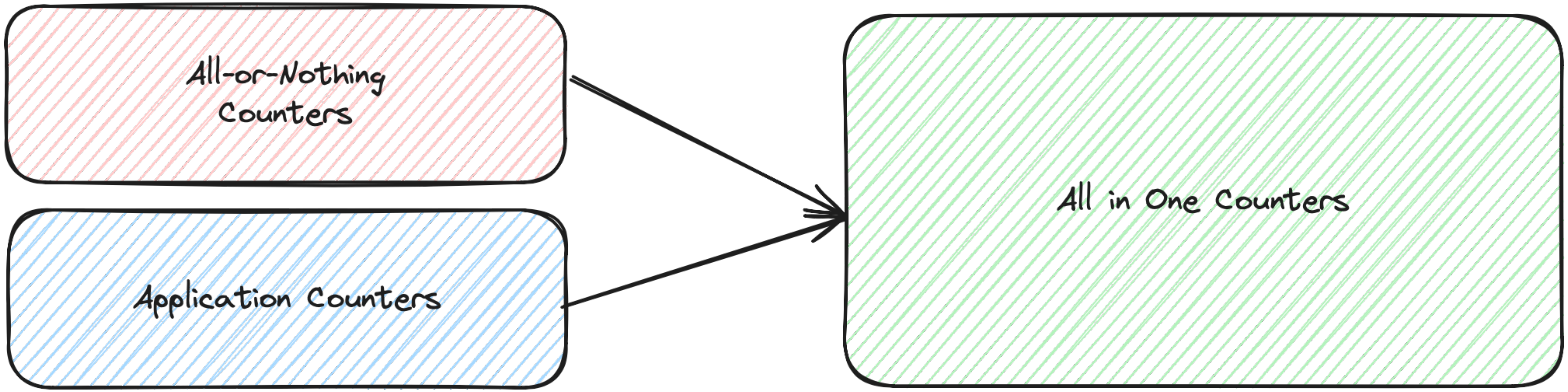
$$\Gamma \vdash \Sigma \Rightarrow e \Rightarrow A$$

Context

Quantitative Type Assignment Systems (QTASs)

$$\boxed{\Gamma \vdash_n e : A}$$

- A variant of Type Assignment Systems (TASs);
- Typing is parametrised by counters;
- Counters quantify how much information we know from the context;
- Counters can vary in different systems;



QTAS: All-or-nothing counters and application counters

- All-or-nothing counters are zero (0) and infinity (∞);
  - models modes in bidirectional typing;

No contextual information: $\Gamma \vdash_0 e : A$

Full contextual information: $\Gamma \vdash_\infty e : A$

- Application counters have successors (S n);
  - quantify how many input types we know from the context.

Partial contextual information: $\Gamma \vdash_{S\ 0} e : A \to B$

Partial contextual information: $\Gamma \vdash_{S\ S\ 0} e : A \to B \to C$

Quantitative Type Assignment Systems (QTASs)

DLam

$$\frac{\Gamma, x : A \vdash_{n-} e : B}{\Gamma \vdash_n \lambda x.\, e : \boxed{A} \rightarrow B}$$

non-zero counter

DApp2

$$\frac{\Gamma \vdash_{(S\ n)} e_1 : A \rightarrow B \qquad \Gamma \vdash_0 e_2 : \boxed{A}}{\Gamma \vdash_n e_1\ e_2 : B}$$

Towards a Non-Backtracking Algorithm

DApp1

$$\frac{\Gamma \vdash_0 e_1 : A \to B \qquad \Gamma \vdash_\infty e_2 : A}{\Gamma \vdash_0 e_1\ e_2 : B}$$

DApp2

$$\frac{\Gamma \vdash_{(S\ n)} e_1 : A \to B \qquad \Gamma \vdash_0 e_2 : A}{\Gamma \vdash_n e_1\ e_2 : B}$$

What's the root cause?

Consider two simple cases:

$(\lambda x\,.\ e)\ e_2$ → $$\frac{\Gamma \vdash_{(S\ n)} \lambda x.\ e : A \to B \qquad \Gamma \vdash_0 e_2 : A}{\Gamma \vdash_n (\lambda x.\ e)\ e_2 : B}\ \text{DApp2}$$

$x\ e_2$ → $$\frac{\Gamma \vdash_0 x : A \to B \qquad \Gamma \vdash_n e_2 : A}{\Gamma \vdash_0 x\ e_2 : B}\ \text{DApp1}$$

$(e_1\ e_2)\ e_3$

Application Consumer

- it is either a variable, a lambda or an annotated term;
- it is the term that will eventually consume the contextual information for the argument;

$$(\lambda x.\ x)\ e_2$$

DApp2

$$((\lambda x.\ y)\ 1)\ (\lambda z.\ z)$$

DApp1

DApp2

Application Consumer

$(\lambda x.\ x + 1)$ `true` ✗ DApp2

Finding the application consumer tells us the best rule to apply;

but does not tell us whether the typing will be successful or not!

Bring arguments and application consumer together

Teleporting Typing Judgements

Teleportation: transport the argument to its application consumer

$$\dfrac{\dfrac{\dfrac{\dfrac{\dots \qquad \boxed{\Gamma, x : I, y : I \vdash 2 : I}}{\Gamma, x : I \vdash \lambda y.\, x + y : I \to I}\,\textsf{Lam} \qquad \boxed{\Gamma, x : I \vdash 1 : I}}{\Gamma \vdash \lambda x.\, \lambda y.\, x + y : I \to I \to I}\,\textsf{Lam} \qquad \boxed{\Gamma \vdash 1 : I}}{\Gamma \vdash (\lambda x.\, \lambda y.\, x + y)\ 1 : I \to I}\,\textsf{App} \qquad \boxed{\Gamma \vdash 2 : I}}{\Gamma \vdash ((\lambda x.\, \lambda y.\, x + y)\ 1)\ 2 : I}\,\textsf{App}$$

Syntax-directed Algorithmic Type System

$$\boxed{\Gamma \vdash \Sigma \Rightarrow e \Rightarrow A}$$

inputs

- Typing is parametrised by surrounding contexts (Σ);
- Surrounding contexts capture the information that is in context for the terms;
- A surrounding context can be empty, full type, or a sequence of terms;

$$\Sigma ::= \square \mid A \mid \boxed{e} \mapsto \Sigma$$

Syntax-directed Algorithmic Type System

- Terms in contexts are deferred type checking tasks of applications;

$$\frac{\Gamma \vdash \boxed{e_2} \mapsto \Sigma \Rightarrow e_1 \Rightarrow A \rightarrow B}{\Gamma \vdash \Sigma \Rightarrow e_1\ e_2 \Rightarrow B}\ \texttt{AApp}$$

- Terms in contexts will be carried out to the *application consumer*: inferred or checked

$$\frac{\Gamma \vdash \Box \Rightarrow e_2 \Rightarrow A \qquad \Gamma, x : A \vdash \Sigma \Rightarrow e \Rightarrow B}{\Gamma \vdash \boxed{e_2} \mapsto \Sigma \Rightarrow \lambda x.\ e \Rightarrow A \rightarrow B}\ \texttt{ALam2}$$

$$\frac{\Gamma \vdash \Box \Rightarrow g \Rightarrow A \qquad \Sigma \neq \Box \qquad A \approx \Sigma}{\Gamma \vdash \Sigma \Rightarrow g \Rightarrow A}\ \texttt{ASub}$$

annotated term; variable

$$\frac{\Gamma \vdash A \Rightarrow e \Rightarrow C \qquad \Gamma \vdash B \approx \Sigma}{\Gamma \vdash A \rightarrow B \approx \boxed{e} \mapsto \Sigma}\ \texttt{SubTerm}$$

Metatheory

Soundness (Corollaries):

If $\Gamma \vdash \square \Rightarrow e \Rightarrow A$, then $\Gamma \vdash_0 e : A$

If $\Gamma \vdash A \Rightarrow e \Rightarrow A$, then $\Gamma \vdash_\infty e : A$

Completeness (Corollaries):

If $\Gamma \vdash_0 e : A$, then $\Gamma \vdash \square \Rightarrow e \Rightarrow A$.

If $\Gamma \vdash_\infty e : A$, then $\Gamma \vdash A \Rightarrow e \Rightarrow A$.

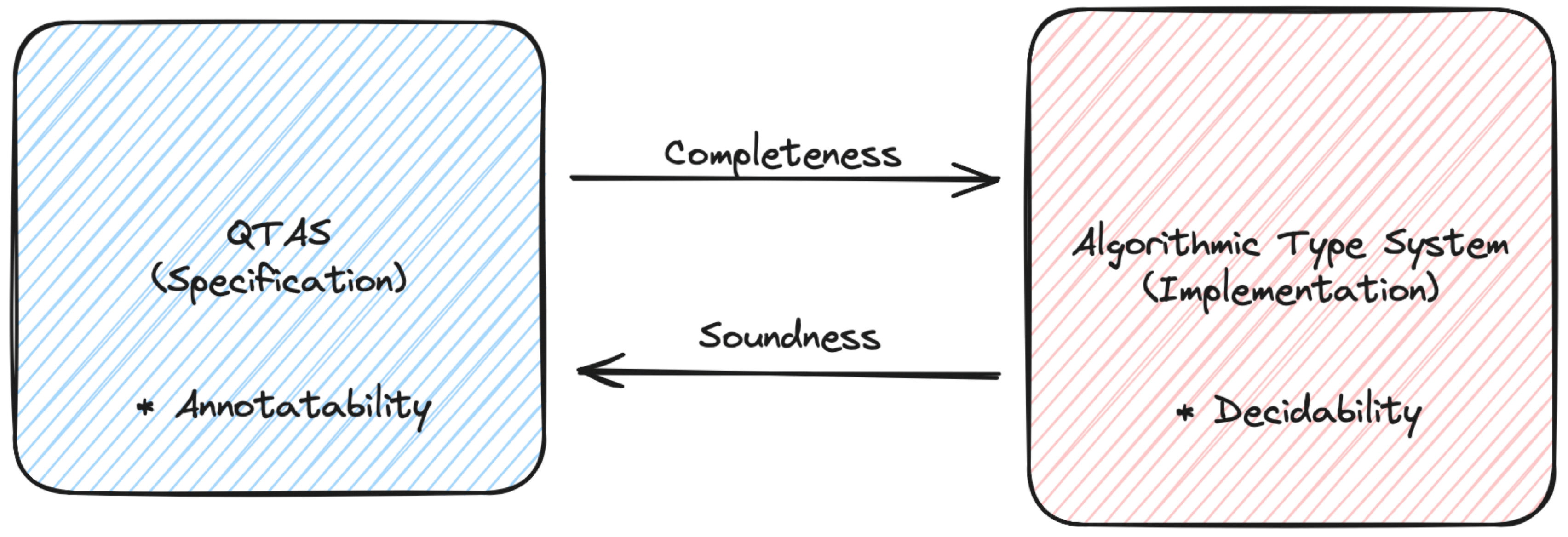
A Calculus with Intersection Types, Overloading and Records

- Introduces subtyping, expressive subsumption;
- Models features such as function overloading, record projection;
- More counters: check counters and projection counters;
- More context: labels can be appended to the context;
- All the properties: annotatability, decidability, soundness, completeness

Check Our Paper!

Future Work?

Our Solution: Contextual Typing



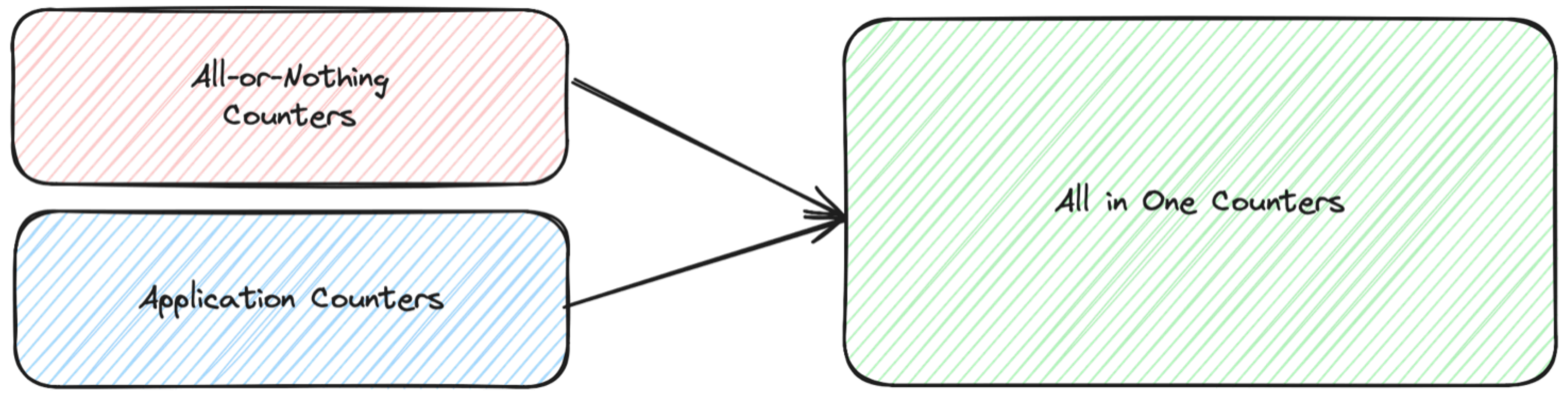
$\Gamma \vdash_n e : A$ — Counter

$\Gamma \vdash \Sigma \Rightarrow e \Rightarrow A$ — Context

Quantitative Type Assignment Systems (QTASs)

$\boxed{\Gamma \vdash_n e : A}$

- A variant of Type Assignment Systems (TASs);
- Typing is parametrised by counters;
- Counters quantify how much information we know from the context;
- Counters can vary in different systems;



https://github.com/juniorxxue/contextual-typing

Formalised in Agda

Application Consumer

- it is either a variable, a lambda or an annotated term;
- it is the term that will eventually consume the contextual information for the argument;

$(\lambda x.\, x)\ e_2$ — DApp2

$((\lambda x.\, y)\ 1)\ (\lambda z.\, z)$ — DApp1, DApp2

Syntax-directed Algorithmic Type System

- Terms in contexts are deferred type checking tasks of applications;

$$\frac{\Gamma \vdash \boxed{e_2} \mapsto \Sigma \Rightarrow e_1 \Rightarrow A \to B}{\Gamma \vdash \Sigma \Rightarrow e_1\ e_2 \Rightarrow B}\ \text{AApp}$$

- Terms in contexts will be dealt with by *application consumer*: inferred or checked

$$\frac{\Gamma \vdash \square \Rightarrow e_2 \Rightarrow A \qquad \Gamma, x : A \vdash \Sigma \Rightarrow e \Rightarrow B}{\Gamma \vdash \boxed{e_2} \mapsto \Sigma \Rightarrow \lambda x.\, e \Rightarrow A \to B}\ \text{ALam2}$$

$$\frac{\Gamma \vdash \square \Rightarrow g \Rightarrow A \qquad \Sigma \neq \square \qquad A \approx \Sigma}{\Gamma \vdash \Sigma \Rightarrow g \Rightarrow A}\ \text{ASub}$$

annotated term; variable

$$\frac{\Gamma \vdash A \Rightarrow e \Rightarrow C \qquad \Gamma \vdash B \approx \Sigma}{\Gamma \vdash A \to B \approx \boxed{e} \mapsto \Sigma}\ \text{SubTerm}$$

QTAS: Annotatability (How to annotate a program)

doesn't tell us where to put annotations

Weak Annotatability: If $\Gamma \vdash e : A$, then $\exists e', \Gamma \vdash_0 e' : A$ and e is the (type) erasure of e'.

Strong Annotatability:

1) If $\Gamma \vdash e : A \rightsquigarrow e'$, then $\Gamma \vdash_{(\mathtt{need}\ e)} e' : A$. *2) If $\Gamma \vdash e : A \rightsquigarrow e'$, then $\Gamma \vdash_0 (e' : A) : A$.*