Local Contextual Type Inference

Xu Xue, Chen Cui, Shengyi Jiang, Bruno C. d. S. Oliveira

(The University of Hong Kong)

Local Algorithms

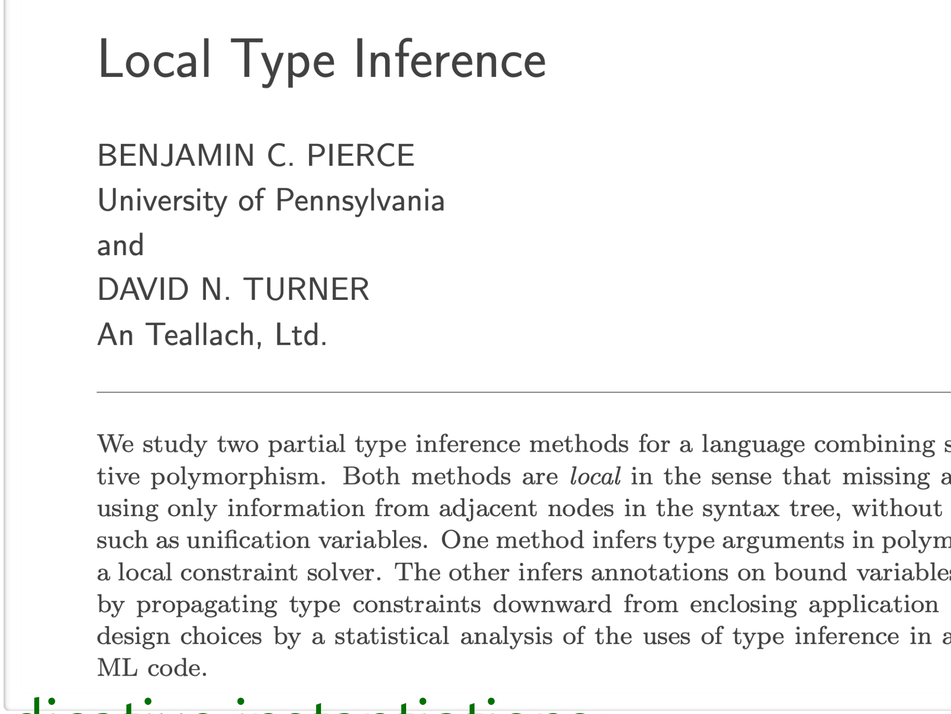
Contextual Typing

Local Contextual Type Inference

Local Type Inference

Local Type Inference

- Adopted by Java, Scala, C#, TypeScript etc.
- Considered as partial type inference methods;
- Scales to advanced features, including implicit impredicative instantiations;
- Consists of two key ideas:
  - Bidirectional Typing
  - Local synthesis of type arguments



Local Type Inference

BENJAMIN C. PIERCE
University of Pennsylvania
and
DAVID N. TURNER
An Teallach, Ltd.

We study two partial type inference methods for a language combining s
tive polymorphism. Both methods are *local* in the sense that missing a
using only information from adjacent nodes in the syntax tree, without
such as unification variables. One method infers type arguments in polym
a local constraint solver. The other infers annotations on bound variable
by propagating type constraints downward from enclosing application
design choices by a statistical analysis of the uses of type inference in a
ML code.

Local Type Inference: Specifications and Limitations

- Functions are default uncurried, whose types merged with polymorphic types;

- Four application rules: instantiation logic is monolithic;
  - Locality is limited to an argument list of an uncurried application;
    - cannot be propagated to, e.g., other argument lists;

- it requires all arguments in a list to be inferable;

$$\frac{\Gamma \vdash f \Rightarrow \forall \overline{\alpha}.\, \overline{B} \to C \qquad \Gamma \vdash \overline{e} \Leftarrow [\overline{A}/\overline{\alpha}]\overline{B}}{\Gamma \vdash f[\overline{A}](\overline{e}) \Rightarrow [\overline{A}/\overline{\alpha}]C}\ \text{S-App}$$

$$\frac{\Gamma \vdash f \Rightarrow \forall \overline{\alpha}.\, \overline{B} \to C \qquad \Gamma \vdash \overline{e} \Rightarrow \overline{D} \qquad |\overline{\alpha}| > 0 \quad \Gamma \vdash \overline{D} <: [\overline{A}/\overline{\alpha}]\overline{B} \qquad \forall \overline{F}.(\Gamma \vdash \overline{D} <: [\overline{F}/\overline{\alpha}]\overline{B} \text{ implies } \Gamma \vdash [\overline{A}/\overline{\alpha}]C <: [\overline{F}/\overline{\alpha}]C)}{\Gamma \vdash f(\overline{e}) \Rightarrow [\overline{A}/\overline{\alpha}]C}\ \text{S-App-InfSpec}$$

- Practical languages use instantiation information immediately, left to right

You're designing a bidirectional type system

For function application rules

Inferring the type of function, checking against arguments

Inferring the type of argument, "partial-inferring" the type of function

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

$$\frac{\Gamma \vdash e_1\ ?\ A \to B \qquad \Gamma \vdash e_2 \Rightarrow A}{\Gamma \vdash e_1\ e_2 \Rightarrow B}\ \textsc{App}$$

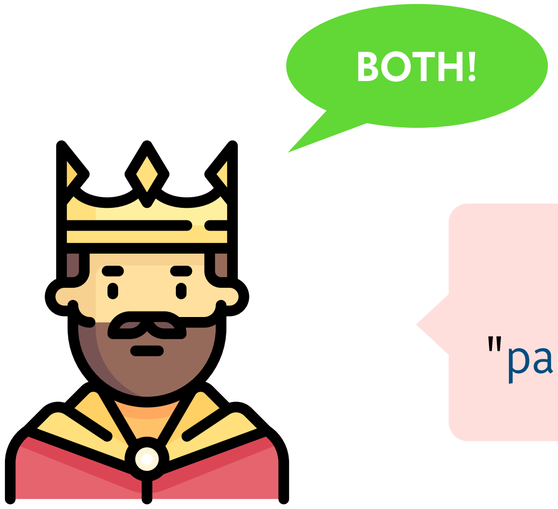
- Default choice
- Useful for higher-order application
- Arguments can be unannotated lambdas

- No suitable modes for functions
- Useful for polymorphic function instantiation
- Use argument information to assist function inference

You're designing a bidirectional type system

For function application rules

Inferring the type of function,
checking the type of arguments



Inferring the type of argument,
"partial-inferring" the type of function

Backtracking

Contextual Typing [Xue & Oliveira 2024]

① find a way to specify the "partial-inference"

Contextual Type Assignment Systems (CTAS):
Don't use modes, use **masks***.

$$\begin{array}{ll} \text{Atomic Masks} & a ::= \square \mid \blacksquare \\ \text{Masks} & m ::= a \mid a\ m \end{array}$$

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

** "Masks" is a re-interpretation of "counters" in the original work "Contextual Typing"*

Contextual Typing [Xue & Oliveira 2024]

① find a way to specify the "partial-inference"

Contextual Type Assignment Systems (CTAS):
Don't use modes, use **masks**.

Masks characterize what information we know from the context

No contextual information: `Γ ⊢■ e : A`

Full contextual information: `Γ ⊢□ e : A`

Partial contextual information: `Γ ⊢□■ e : A → B`

Contextual Typing [Xue & Oliveira 2024]

① find a way to specify the "partial-inference"

Contextual Type Assignment Systems (CTAS):
Don't use modes, use **masks**.

Application rules as mask collectors

$$\frac{\Gamma \vdash_{(\overline{a}\ m)}\ e_1 : A \to B \quad \Gamma \vdash_{a}\ e_2 : A}{\Gamma \vdash_{m}\ e_1\ e_2 : B}$$

Contextual Typing [Xue & Oliveira 2024]

② find a way to avoid backtracking to have both app rules

In algorithm, don't decide inference/checking of arguments in application, decide it when it's "consumed".

Contexts (Σ) are introduced to store deferred type-checking tasks of arguments

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

Contextual Typing [Xue & Oliveira 2024]

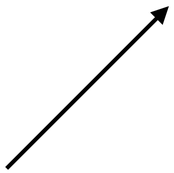
Two systems (specification & algorithm)
are proved to be equivalent.

Local Contextual Type Inference

- It's a refinement of LTI using contextual typing ideas;
- It gives rise to a modular rule design;
- It, just like LTI, advocates matching instead of unification;
- It provides clearer specification, making it closer to canonical systems (System F);
- It removes practical restrictions of LTI and offers better locality;
- It's rigorously studied (all are mechanized) Agda ROCQ
- We hope it stands as a principled and practical foundation for inference

Example (hypothetical): id 1

$$\frac{\Gamma \vdash C \qquad \Gamma \vdash [C/\alpha]A \leq B}{\Gamma \vdash \forall \alpha.\, A \leq B}$$



$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathsf{id} : \forall \alpha.\, \alpha \to \alpha}\,\mathrm{Var} \qquad \dfrac{}{\Gamma \vdash \forall \alpha.\, \alpha \to \alpha \leq \mathsf{Int} \to \mathsf{Int}}\,\forall L}{\Gamma \vdash \mathsf{id} : \mathsf{Int} \to \mathsf{Int}}\,\mathrm{Sub} \qquad \dfrac{}{\Gamma \vdash 1 : \mathsf{Int}}\,\mathrm{Lit}}{\Gamma \vdash \mathsf{id}\ 1 : \mathsf{Int}}\,\mathrm{App}$$

Unrestricted $\forall L$ is undecidable, people often compromise to a decidable fragment

Example (w/masks): id 1

$$\dfrac{\dfrac{}{\Gamma \vdash \mathrm{id} : \forall\alpha.\ \alpha \to \alpha}\ \mathrm{Var} \qquad \dfrac{}{\Gamma \vdash \forall\alpha.\ \alpha \to \alpha \le \mathrm{Int} \to \mathrm{Int}}\ \forall\mathrm{L}}{\Gamma \vdash \mathrm{id} : \mathrm{Int} \to \mathrm{Int}}\ \mathrm{Sub} \qquad \dfrac{}{\Gamma \vdash 1 : \mathrm{Int}}\ \mathrm{Lit}$$

$$\Gamma \vdash \mathrm{id}\ 1 : \mathrm{Int}\ \ \mathrm{App}$$

$$\dfrac{\dfrac{}{\Gamma \vdash_{\blacksquare} \mathrm{id} : \forall\alpha.\ \alpha \to \alpha}\ \mathrm{Var} \qquad \dfrac{}{\Gamma \vdash_{\square\blacksquare} \forall\alpha.\ \alpha \to \alpha \le \mathrm{Int} \to \mathrm{Int}}\ \forall\mathrm{L}}{\Gamma \vdash_{\square\blacksquare} \mathrm{id} : \mathrm{Int} \to \mathrm{Int}}\ \mathrm{Sub} \qquad \dfrac{}{\Gamma \vdash_{\blacksquare} 1 : \mathrm{Int}}\ \mathrm{Lit}$$

$$\Gamma \vdash_{\blacksquare} \mathrm{id}\ 1 : \mathrm{Int}\ \ \mathrm{App}$$

Key insight: Leverage masks to ensure the guess can be safely made.

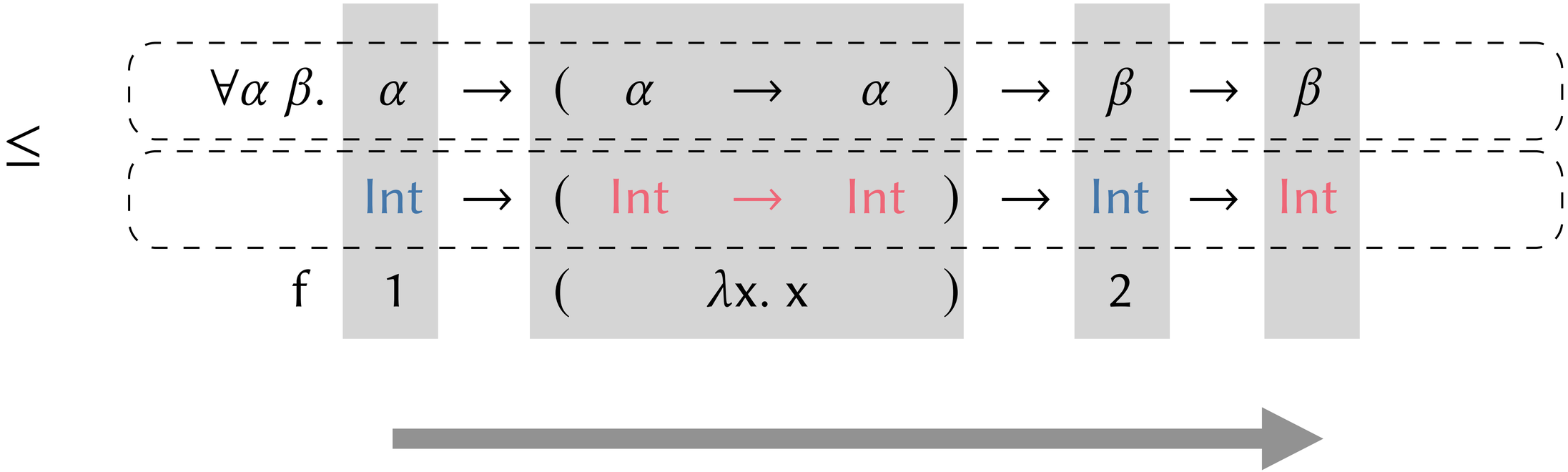
Restrict $\forall L$ rule with masks

$$\frac{\Gamma \vdash C \qquad \Gamma \vdash [C/\alpha]A \leq B}{\Gamma \vdash \forall \alpha.\, A \leq B}$$

$$\frac{\Gamma \vdash B \qquad A^{\alpha}_{(a\ m)} \qquad \Gamma \vdash_{(a\ m)} [B/\alpha]A \leq C \to D}{\Gamma \vdash_{(a\ m)} \forall \alpha.\, A \leq C \to D}$$

- Polymorphic types can only be instantiated to function types.
  - Implicit polymorphism is *application triggered only*.
- **Instantiability** guarantees us that we are able to make the guess;

Instantiability: when you can make the guess



- Left-to-right traversal;
- Interleaved inference and checking of arguments

Towards Algorithm: Matching Subtyping

- Matching variables: track solved instantiations;

$$\frac{\Gamma \vdash B \qquad A^{\alpha}_{(a\ m)} \qquad \Gamma \mid \Delta, \hat{\alpha} = B \vdash_{(a\ m)} A \leq^{+} C \to D}{\Gamma \mid \Delta \vdash_{(a\ m)} \forall \alpha.\ A \leq^{+} C \to D}$$

- Solution of matching variables cannot contain other matching variables
  - Different to unification: $\alpha = \beta \to \beta$ is disallowed in matching.
- **Invariant**: matching variables can only appear on one side.

$$\frac{\Gamma \vdash f \Rightarrow \forall \overline{\alpha}.\ \overline{B} \to C \quad \Gamma \vdash \overline{e} \Rightarrow \overline{D} \qquad |\overline{\alpha}| > 0 \quad \Gamma \vdash \overline{D} <: [\overline{A}/\overline{\alpha}]\overline{B} \qquad \forall \overline{F}.(\Gamma \vdash \overline{D} <: [\overline{F}/\overline{\alpha}]\overline{B} \text{ implies } \Gamma \vdash [\overline{A}/\overline{\alpha}]C <: [\overline{F}/\overline{\alpha}]C)}{\Gamma \vdash f(\overline{e}) \Rightarrow [\overline{A}/\overline{\alpha}]C}$$

implied by

Towards Algorithm: Input and Output Environments

1) unsolved matching variables $\hat{\alpha}$

$$\frac{\Gamma \mid \Delta, \hat{\alpha} \vdash A \leq^{+} \boxed{e} \rightsquigarrow \Sigma \dashv \Delta', \hat{\alpha} =^{?} C \rightsquigarrow\!\!\!\!\rightarrow B}{\Gamma \mid \Delta \vdash \forall \alpha.\, A \leq^{+} \boxed{e} \rightsquigarrow \Sigma \dashv \Delta' \rightsquigarrow\!\!\!\!\rightarrow B}$$

2) learn more information from the **inference** of arguments

$$\frac{\Gamma \mid \Delta \vdash_{\mathsf{open}} A \quad \Gamma \vdash \blacksquare \Rightarrow e \Rightarrow C \qquad \Gamma \mid \Delta \vdash C \leq^{-} A \dashv \Delta' \quad \Gamma \mid \Delta' \vdash B \leq^{+} \Sigma \dashv \Delta'' \rightsquigarrow\!\!\!\!\rightarrow D}{\Gamma \mid \Delta \vdash A \to B \leq^{+} \boxed{e} \rightsquigarrow \Sigma \dashv \Delta'' \rightsquigarrow\!\!\!\!\rightarrow C \to D}$$

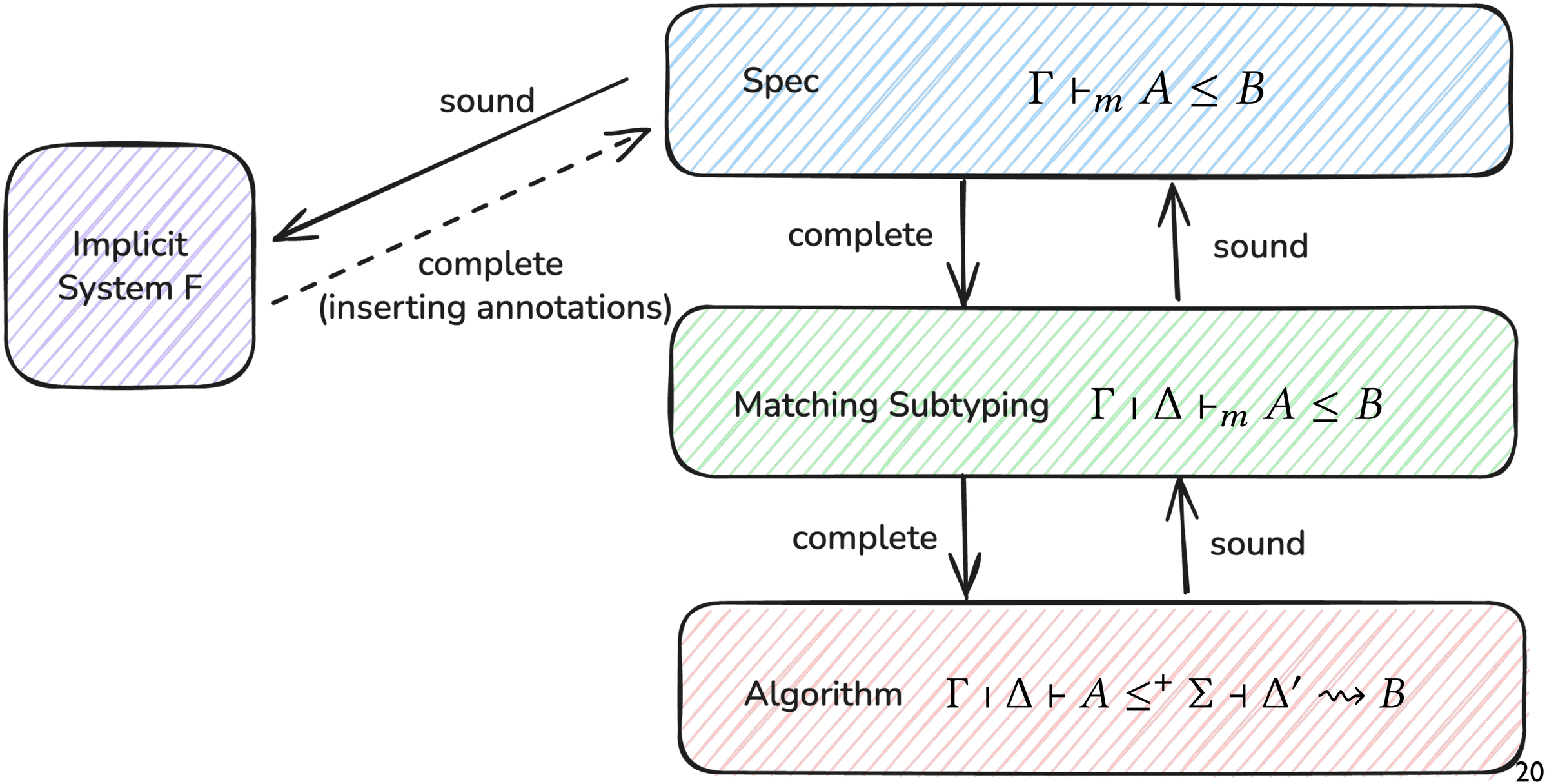
3) once information is enough, **check** arguments

$$\frac{\Gamma \mid \Delta \vdash_{\mathsf{closed}} A \quad \Gamma \vdash [\Delta]A \Rightarrow e \Rightarrow A' \qquad \Gamma \mid \Delta \vdash B \leq^{+} \Sigma \dashv \Delta' \rightsquigarrow\!\!\!\!\rightarrow D}{\Gamma \mid \Delta \vdash A \to B \leq^{+} \boxed{e} \rightsquigarrow \Sigma \dashv \Delta' \rightsquigarrow\!\!\!\!\rightarrow [\Delta]A \to D}$$

Towards Algorithm: Input and Output Environments

$$\dfrac{\dfrac{}{\Gamma \vdash \blacksquare \Rightarrow \mathrm{id} \Rightarrow \forall\alpha.\,\alpha \to \alpha}\ \textsf{Var} \qquad \dfrac{\dfrac{\cdots}{\Gamma, \hat{\alpha} \vdash \alpha \to \alpha \le \boxed{1} \rightsquigarrow \blacksquare \dashv \hat{\alpha} = \mathsf{Int} \leadsto \mathsf{Int} \to \mathsf{Int}}\ \forall\textsf{L}}{\Gamma \vdash \forall\alpha.\,\alpha \to \alpha \le \boxed{1} \rightsquigarrow \blacksquare \leadsto \mathsf{Int} \to \mathsf{Int}}}{\dfrac{\Gamma \vdash \boxed{1} \rightsquigarrow \blacksquare \Rightarrow \mathrm{id} \Rightarrow \mathsf{Int} \to \mathsf{Int}}{\Gamma \vdash \blacksquare \Rightarrow \mathrm{id}\ 1}\ \textsf{App}}\ \textsf{Sub}$$

Recap: Local Contextual Type Inference



Conclusion

- **LCTI**: offering a simple and modular specification of LTI
- **Matching Subtyping**: specify the use of matching in constraint solving
- **Contextual System F**: soundness & completeness across three systems
- Everything at https://github.com/juniorxxue/LCTI
  - Mechanized proofs, Haskell prototype and extended paper