Spine-local Type Inference

Christopher Jenkins and Aaron Stump

Computer Science
University of Iowa

IFL '18

Outline

1. Background and Motivation
   - Local Type Inference
   - Spine-local Type Inference
2. The Specificational System
   - Terms and Terminology
   - Type Inference
3. Discussion
   - Specificational System Properties
   - Algorithmic System Properties
   - Future Work

Outline

What is "Local Type Inference"?

- Introduced by Pierce and Turner in '98
- Extended by Odersky et al. in '01
- Uses two main techniques
  - *Bidirectional typing rules:*
  - *Local type-argument inference:*

What is "Local Type Inference"?

- Introduced by Pierce and Turner in '98
- Extended by Odersky et al. in '01
- Uses two main techniques
  - *Bidirectional typing rules:*

Synthesis mode:	$\lambda\, x : Nat .\, x$	$\Uparrow$	$Nat \rightarrow Nat$
Checking mode:	$\lambda\, x .\, x$	$\Downarrow$	$Nat \rightarrow Nat$

  - *Local type-argument inference:*

What is "Local Type Inference"?

- Introduced by Pierce and Turner in '98
- Extended by Odersky et al. in '01
- Uses two main techniques
  - *Bidirectional typing rules:*

Synthesis mode:	$\lambda\, x : Nat.\, x$	$\Uparrow$	$Nat \to Nat$
Checking mode:	$\lambda\, x.\, x$	$\Downarrow$	$Nat \to Nat$

  - *Local type-argument inference:*

Let	$id : \forall\, X.\, X \to X$	
Type	$\boxed{id\ 0}$	$\Uparrow Nat$
Infer	$X = Nat$	from 0

$\boxed{\text{Local}}$ and Synthetic

Why use local type inference?

- It is a method of *partial* type inference
  - *Complete* type inference: no annotations **ever** (e.g. *Damas-Hindley-Milner* and ML)
  - Undecidable for System F and beyond

Why use local type inference?

- It is a method of *partial* type inference
  - *Complete* type inference: no annotations **ever** (e.g. *Damas-Hindley-Milner* and ML)
  - Undecidable for System F and beyond
- It is user-friendly
  - Infers many type annotations
  - Predictable annotation requirements
  - Better-quality error messages

Why use local type inference?

- It is a method of *partial* type inference
  - *Complete* type inference: no annotations **ever** (e.g. *Damas-Hindley-Milner* and ML)
  - Undecidable for System F and beyond
- It is user-friendly
  - Infers many type annotations
  - Predictable annotation requirements
  - Better-quality error messages
- It is implementer-friendly
  - Relatively simple implementation
  - *Extensible*: new features added without threatening decidability

Limitations

Local type inference in its published form can sometimes still require "silly" type annotations, i.e. those for which there *should be* enough contextual information to omit

Let	$pair : \forall X, Y.\, X \to Y \to X \times Y$
Type	$pair\ (\lambda x.\, x)\ 0$

Limitations

Local type inference in its published form can sometimes still require "silly" type annotations, i.e. those for which there *should be* enough contextual information to omit

Let	$pair : \forall X, Y. X \rightarrow Y \rightarrow X \times Y$	
Type	$pair\ (\lambda x. x)\ 0$	$\Uparrow$???

- We do not expect to locally synthesize a type

Limitations

Local type inference in its published form can sometimes still require "silly" type annotations, i.e. those for which there *should be* enough contextual information to omit

Let	$pair : \forall X, Y. X \to Y \to X \times Y$	
Type	$pair\ (\lambda x. x)\ 0$	$\Uparrow$???

- We do not expect to locally synthesize a type

Limitations

Local type inference in its published form can sometimes still require "silly" type annotations, i.e. those for which there *should be* enough contextual information to omit

Let	$pair : \forall X, Y. X \rightarrow Y \rightarrow X \times Y$		
Type	$pair\ (\lambda x. x)\ 0$	$\Uparrow$	???
Type	$pair\ (\lambda x. x)\ 0$	$\Downarrow$	$Nat \rightarrow Nat \times Nat$

- We do not expect to locally synthesize a type
- ... but we would expect to check it against a type

Limitations

Local type inference in its published form can sometimes still require "silly" type annotations, i.e. those for which there *should be* enough contextual information to omit

Let	$pair : \forall X, Y. X \to Y \to X \times Y$		
Type	$pair\ (\lambda x. x)\ 0$	$\Uparrow$	???
Type	$pair\ (\lambda x. x)\ 0$	$\Downarrow$	$Nat \to Nat \times Nat$

- We do not expect to locally synthesize a type
- ... but we would expect to check it against a type
  - We could call this "contextual" type-argument inference.

Limitations

Local type inference in its published form can sometimes still require "silly" type annotations, i.e. those for which there *should be* enough contextual information to omit

Let	$pair : \forall X, Y.\, X \to Y \to X \times Y$		
Type	$pair\ (\lambda\, x.\, x)\ 0$	$\Uparrow$	???
Type	$pair\ (\lambda\, x.\, x)\ 0$	$\Downarrow$	$Nat \to Nat \times Nat$

- We do not expect to locally synthesize a type
- … but we would expect to check it against a type
  - We could call this "contextual" type-argument inference.
- Unfortunately, this is not done in the two major published systems
  - Popular "unofficial" extension (used in e.g. Scala, Rust)

Limitations (cont.)

- Usually uses “fully-uncurried” function applications

$$f(t_1, .., t_n)$$

  - Maximize available info at a single application

Limitations (cont.)

- Usually uses "fully-uncurried" function applications

$$f(t_1, .., t_n)$$

  - Maximize available info at a single application

- Usually without partial type application ("all-or-nothing")

$$f[T_1, ..., T_m](t_1, ..., t_n)$$

Our Contributions

- Type inference for some expressions not typed by other variants of local type inference, by using *contextual* type-argument inference

- Precise, specificational account of this technique
- Better support function currying and partial type applications by being "spine-local."

Our Contributions

- Type inference for some expressions not typed by other variants of local type inference, by using *contextual* type-argument inference

$$f \;\; t_1^{\Uparrow} \;\; t_2^{\Uparrow} \;\; t_3^{\Downarrow}$$

- Precise, specificational account of this technique
- Better support function currying and partial type applications by being "spine-local."

Our Contributions

- Type inference for some expressions not typed by other variants of local type inference, by using *contextual* type-argument inference

$$f\ \ t_1^{\Uparrow}\ \ t_2^{\Downarrow}\ \ t_3^{\Downarrow}\ \Downarrow\ T$$

- Precise, specificational account of this technique
- Better support function currying and partial type applications by being "spine-local."

Our Contributions

- Type inference for some expressions not typed by other variants of local type inference, by using *contextual* type-argument inference

$$f\ t_1^{\Uparrow}\ t_2^{\Downarrow}\ t_3^{\Downarrow} \Downarrow T$$

- Precise, specificational account of this technique
- Better support function currying and partial type applications by being "spine-local."

$$\boxed{f[S, T, V](t_1, t_2, t_3)}$$

Our Contributions

- Type inference for some expressions not typed by other variants of local type inference, by using *contextual* type-argument inference

$$f \;\; t_1^{\Uparrow} \;\; t_2^{\Downarrow} \;\; t_3^{\Downarrow} \Downarrow T$$

- Precise, specificational account of this technique
- Better support function currying and partial type applications by being "spine-local."

$$\boxed{f[S][T][V] \;\; t_1 \;\; t_2 \;\; t_3}$$

Our Contributions

- Type inference for some expressions not typed by other variants of local type inference, by using *contextual* type-argument inference

$$f \ \ t_1^{\Uparrow} \ \ t_2^{\Downarrow} \ \ t_3^{\Downarrow} \Downarrow T$$

- Precise, specificational account of this technique
- Better support function currying and partial type applications by being “spine-local.”

$$\boxed{f[S] \qquad\qquad t_1 \ \ t_2 \quad}$$

Our type system(s)

- Two type systems: one *specificational* and one *algorithmic*
- Spec. system abstracts contextual type-argument inference
  - Non-deterministic
- Sanity checks for spec. system, annotation requirements
- Equivalence of the two systems

Outline

Our Setting

- The setting for our type inference system is (impredicative) System F
- Internal and external term languages:
  - *internal*: all type annotations and arguments are provided
  - *external*: some of these can be elided
- Type inference viewed as relation between these two langauges
  - *Elaborate* external $\rightsquigarrow$ internal terms

Language Syntax

Types	$S, T, U, V ::= X, Y, Z \mid S \to T \mid \forall X.\, T \mid S \times T$
Contexts	$\Gamma ::= \cdot \mid \Gamma, X \mid \Gamma, x : T$
Internal Terms	$e, p ::= x \mid \lambda\, x : T.\, e \mid \Lambda\, X.\, e \mid e\ e' \mid e[T]$
External Terms	$t ::= x \mid \lambda\, x : T.\, t \mid \Lambda\, X.\, t \mid t\ t' \mid t[T]$

Language Syntax

Types	$S, T, U, V ::= X, Y, Z \mid S \to T \mid \forall X.\, T \mid S \times T$
Contexts	$\Gamma ::= \cdot \mid \Gamma, X \mid \Gamma, x : T$
Internal Terms	$e, p ::= x \mid \lambda\, x : T.\, e \mid \Lambda\, X.\, e \mid e\ e' \mid e[T]$
External Terms	$t ::= x \mid \lambda\, x : T.\, t \mid \Lambda\, X.\, t \mid t\ t' \mid t[T]$

- Pair types for illustration

Language Syntax

Types	$S, T, U, V ::= X, Y, Z \mid S \to T \mid \forall X.\, T \mid S \times T$
Contexts	$\Gamma ::= \cdot \mid \Gamma, X \mid \Gamma, x : T$
Internal Terms	$e, p ::= x \mid \lambda x : T.\, e \mid \Lambda X.\, e \mid e\ e' \mid e[T]$
External Terms	$t ::= x \mid \lambda x : T.\, t \mid \Lambda X.\, t \mid t\ t' \mid t[T]$
	$\mid \lambda x.\, t$

- Pair types for illustration
- Will try to infer binder annotations and type arguments in external language

Language Syntax

$$
\begin{array}{rrl}
\textbf{Types} & S, T, U, V & ::= X, Y, Z \mid S \to T \mid \forall X.\, T \mid S \times T \\
\textbf{Contexts} & \Gamma & ::= \cdot \mid \Gamma, X \mid \Gamma, x : T \\
\textbf{Internal Terms} & e, p & ::= x \mid \lambda x : T.\, e \mid \Lambda X.\, e \mid e\ e' \mid e[T] \\
\textbf{External Terms} & t & ::= x \mid \lambda x : T.\, t \mid \Lambda X.\, t \mid t\ t' \mid t[T] \\
 & & \mid \lambda x.\, t
\end{array}
$$

- Pair types for illustration
- Will try to infer binder annotations **and type arguments** in external language

Terminology

- *Application head*: variable or abstraction

$$x, \quad \Lambda X.\, t, \quad \lambda x.\, t$$

Terminology

- *Application head*: variable or abstraction

$$x, \quad \Lambda X.\, t, \quad \lambda x.\, t$$

- *Application spine*: head followed by seq. of term, type arguments

$$\boxed{x \quad}\boxed{t_1 \;\; t_2 \;\; t_3} \text{ vs } (((x \;\; t_1) \;\; t_2) \;\; t_3)$$

Terminology

- *Application head*: variable or abstraction

$$x, \quad \Lambda X.\, t, \quad \lambda x.\, t$$

- *Application spine*: head followed by seq. of term, type arguments

$$\boxed{x \quad}\boxed{t_1 \;\; t_2 \;\; t_3} \text{ vs } (((x \;\, t_1) \;\, t_2) \;\, t_3)$$

- *Applicand:* Term in the function position of an application

$$t_1 \text{ in } t_1 \;\, t_2$$

Terminology

- *Application head*: variable or abstraction

$$x, \quad \Lambda X.\, t, \quad \lambda x.\, t$$

- *Application spine*: head followed by seq. of term, type arguments

$$\boxed{x \quad}\boxed{t_1 \;\; t_2 \;\; t_3} \text{ vs } (((x \;\; t_1) \;\; t_2) \;\; t_3)$$

- *Applicand:* Term in the function position of an application

$$t_1 \text{ in } t_1 \;\; t_2$$

- *Maximal application:* spine that is not an applicand

Not max	$\underline{x \;\; t_1 \;\; t_2} \;\; t_3$
Max	$\underline{x \;\; t_1 \;\; t_2 \;\; t_3}$

Example – High Level Goals

Example from the intro: $\Gamma \vdash_{\Downarrow} \mathit{pair}\ (\lambda\, x.\, x)\ 0 : (\mathit{Nat} \rightarrow \mathit{Nat}) \times \mathit{Nat}$

- "Under context Γ, the expression checks against the given type"
  (Where *pair* and 0 are suitably defined)

Example – High Level Goals

Example from the intro: $\Gamma \vdash_{\Downarrow} pair\ (\lambda\, x.\, x)\ 0 : (Nat \to Nat) \times Nat$

- "Under context Γ, the expression checks against the given type" (Where *pair* and 0 are suitably defined)
- System will elaborate to $pair[Nat \to Nat][Nat]\ (\lambda\, x\!:\!Nat.\, x)\ 0$ For illustration, example shows synthetic and contextual type-arg. inference

Example – High Level Goals

Example from the intro: $\Gamma \vdash_{\Downarrow} \mathit{pair}\ (\lambda\, x.\, x)\ 0 : (\mathit{Nat} \to \mathit{Nat}) \times \mathit{Nat}$

- "Under context Γ, the expression checks against the given type" (Where *pair* and 0 are suitably defined)
- System will elaborate to $\mathit{pair}[\mathit{Nat} \to \mathit{Nat}][\mathit{Nat}]\ (\lambda\, x : \mathit{Nat}.\, x)\ 0$
  For illustration, example shows synthetic and contextual type-arg. inference

Example – High Level Goals

Example from the intro: $\Gamma \vdash_{\Downarrow} \mathit{pair}\ (\lambda x.\, x)\ 0 : (\mathit{Nat} \to \mathit{Nat}) \times \mathit{Nat}$

- "Under context Γ, the expression checks against the given type" (Where *pair* and 0 are suitably defined)
- System will elaborate to $\mathit{pair}[\mathit{Nat} \to \mathit{Nat}][\mathit{Nat}]\ (\lambda x{:}\mathit{Nat}.\, x)\ 0$
  For illustration, example shows synthetic and contextual type-arg. inference
- ... however, elaboration clutters the rules, so omitted for the example

Spine Judgment

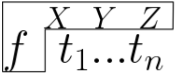
$$\boxed{\Gamma \vdash^{\mathsf{P}} t : T \rightsquigarrow \sigma}$$

- "Spine t partially synthesizes type T with contextual type-args. σ"
- Big idea: enforce locality, contextuality **at maximal applications**

Spine Judgment

$$\boxed{\Gamma \vdash^{\mathsf{P}} t : T \leadsto \sigma}$$

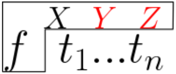
- "Spine t partially synthesizes type T with contextual type-args. σ"
- Big idea: enforce locality, contextuality **at maximal applications**
  - $\boxed{\text{cage meta-variables}}$ to just the spine with spine judgment (*locality*)



Spine Judgment

$$\boxed{\Gamma \vdash^{\mathsf{P}} t : T \leadsto \sigma}$$

- "Spine t partially synthesizes type T with contextual type-args. σ"
- Big idea: enforce locality, contextuality **at maximal applications**
  - cage meta-variables to just the spine with spine judgment (*locality*)
  - require meta-variable "guesses" justified *contextuality*



Spine Judgment (Ex.)

$$\Gamma \vdash^{\mathsf{P}} \mathit{pair}\ (\lambda\, x.\, x)\ 0 : X \times \mathit{Nat} \leadsto [\mathit{Nat} \to \mathit{Nat} / X]$$

Spine Judgment (Ex.)

$$\boxed{\Gamma \vdash^{P} \textit{pair}\ (\lambda\, x.\, x)\ 0 : X \times \textit{Nat} \rightsquigarrow [\textit{Nat} \rightarrow \textit{Nat}/X]}$$

Base case: synthesize type for head

$\Gamma \vdash_{\Uparrow} \textit{pair} : \forall\, X, Y.\, X \rightarrow Y \rightarrow X \times Y$

Spine Judgment (Ex.)

$$\boxed{\Gamma \vdash^{P} \mathit{pair}\ (\lambda\, x.\, x)\ 0 : X \times \mathit{Nat} \leadsto [\mathit{Nat} \to \mathit{Nat}/X]}$$

Begin walking up spine

$\Gamma \vdash^{P} \mathit{pair} : \forall X, Y.\, X \to Y \to X \times Y \leadsto \sigma_{id}$ (σ_{id} is identity subst.)

Spine Judgment (Ex.)

$$\boxed{\Gamma \vdash^{P} \textit{pair}\ (\lambda\, x.\, x)\ 0 : X \times \textit{Nat} \rightsquigarrow [\textit{Nat} \to \textit{Nat}/X]}$$

Encounter term app. with missing type arg.

$\Gamma \vdash^{P} \textit{pair} : \forall\, X, Y.\, X \to Y \to X \times Y \rightsquigarrow \sigma_{id}$ (σ_{id} is identity subst.)

Spine Judgment (Ex.)

$$\boxed{\Gamma \vdash^{P} \color{purple}{pair\ (\lambda\, x.\, x)}\ 0 : X \times Nat \rightsquigarrow [Nat \to Nat / X]}$$

Defer to last judgment form: *application* judgment

$$\Gamma \vdash^{\cdot} (\forall\, X, Y.\, X \to Y \to X \times Y, \sigma_{id}) \cdot (\lambda\, x.\, x) : Y \to X \times Y \rightsquigarrow [Nat \to Nat / X]$$

Application Judgment

$$\boxed{\Gamma \vdash^{\cdot} (T, \sigma) \cdot t : T' \rightsquigarrow \sigma'}$$

- "An applicand of type T with ctxt. solutions σ can be applied to argument t, producing result type T' and result ctxt. solutions σ'"
- Infer missing type-args in term apps., synthetically *and* contextually
- Type application when arrow revealed

Application Judgment

$$\boxed{\Gamma \vdash^{\cdot} (T, \sigma) \cdot t : T' \leadsto \sigma'}$$

- "An applicand of type T with ctxt. solutions σ can be applied to argument t, producing result type T' and result ctxt. solutions σ'"
- Infer missing type-args in term apps., synthetically *and* contextually
  - the *whether* and *what* of contextual inference is non-deterministic
- Type application when arrow revealed

Application Judgment (Ctx.)

$$\boxed{\Gamma \vdash^{\cdot} (\forall X{,}Y.\, X \to Y \to X \times Y, \sigma_{id}) \cdot (\lambda\, x.\, x) : Y \to X \times Y \leadsto [\mathit{Nat} \to \mathit{Nat}/X]}$$

Application Judgment (Ctx.)

$$\boxed{\Gamma \vdash^{\cdot} (\forall {\color{red}X}, Y. X \to Y \to X \times Y, \sigma_{id}) \cdot (\lambda x. x) : Y \to X \times Y \rightsquigarrow [Nat \to Nat/X]}$$

Make a contextual guess for X, $Nat \to Nat$

Application Judgment (Ctx.)

$$\boxed{\Gamma \vdash^{\cdot} (\forall\, Y.\, X \to Y \to X \times Y, [Nat \to Nat/X]) \cdot (\lambda\, x.\, x) : Y \to X \times Y \leadsto [Nat \to Nat/X]}$$

Application Judgment (Ctx.)

$$\boxed{\Gamma \vdash^{\cdot} (\forall\, Y.\, X \to Y \to X \times Y, [Nat \to Nat/X]) \cdot (\lambda\, x.\, x) : Y \to X \times Y \rightsquigarrow [Nat \to Nat/X]}$$

Non-deterministically choose to instantiate Y synthetically

Application Judgment (Ctx.)

$$\boxed{\Gamma \vdash^{\cdot} (X \rightarrow Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot (\lambda\, x.\, x) : Y \rightarrow X \times Y \leadsto [Nat \rightarrow \; Nat/X]}$$

Application Judgment (Ctx.)

$$\boxed{\Gamma \vdash^{\cdot} (X \to Y \to X \times Y, [Nat \to Nat/X]) \cdot (\lambda\, x.\, x) : Y \to X \times Y \rightsquigarrow [Nat \to Nat/X]}$$

Reveal an arrow in applicand type

Application Judgment (Arrow)

Two cases arise when we reveal an arrow.

Application Judgment (Arrow)

Two cases arise when we reveal an arrow.

- Expected type of arg. is *fully known* (from spine head, contextual type, previous arguments)
  Use checking mode for arg.

Application Judgment (Arrow)

Two cases arise when we reveal an arrow.

- Expected type of arg. is *fully known* (from spine head, contextual type, previous arguments)
  Use checking mode for arg.
- Expected type has unsolved meta-vars
  Use synthesis mode for arg. to learn instantiations

Application Judgment (Ctx)

$$\boxed{\Gamma \vdash^{\cdot} (X \rightarrow Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot (\lambda\, x.\, x) : Y \rightarrow X \times Y \rightsquigarrow [Nat \rightarrow\ Nat/X]}$$

Application Judgment (Ctx)

$$\boxed{\Gamma \vdash^{\cdot} (X \rightarrow Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot (\lambda\, x.\, x) : Y \rightarrow X \times Y \rightsquigarrow [Nat \rightarrow Nat/X]}$$

Type is fully known: $\Gamma \vdash_{\Downarrow} \lambda\, x.\, x : Nat \rightarrow Nat$

Application Judgment (Ctx)

$$\boxed{\Gamma \vdash^{\cdot} (X \to Y \to X \times Y, [Nat \to Nat/X]) \cdot (\lambda\, x.\, x) : Y \to X \times Y \leadsto [Nat \to Nat/X]}$$

Produced result type of the app, with ctxt. solution

Application Judgment (Syn)

- Last part of the spine judgment is typing *pair* $(\lambda x. x)$ to 0
- We defer again to application judgment
- Y will be inferred synthetically from 0

Application Judgment (Syn)

$$\boxed{\Gamma \vdash^{\cdot} (Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot 0 : X \times Nat \rightsquigarrow [Nat \rightarrow Nat/X]}$$

Application Judgment (Syn)

$$\boxed{\Gamma \vdash^{\cdot} (Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot 0 : X \times Nat \rightsquigarrow [Nat \rightarrow Nat/X]}$$

Arrow revealed

Application Judgment (Syn)

$$\boxed{\Gamma \vdash^{\cdot} (Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot 0 : X \times Nat \rightsquigarrow [Nat \rightarrow Nat/X]}$$

Incomplete info. for expected arg. type Y

Application Judgment (Syn)

$$\boxed{\Gamma \vdash^{\cdot} (Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot 0 : X \times Nat \rightsquigarrow [Nat \rightarrow Nat/X]}$$

Synthesize type for arg. (note Y not passed down!)

$\Gamma \vdash_{\Uparrow} 0 : Nat$

Application Judgment (Syn)

$$\boxed{\Gamma \vdash^{\cdot} (Y \rightarrow X \times Y, [Nat \rightarrow Nat/X]) \cdot 0 : X \times Nat \rightsquigarrow [Nat \rightarrow Nat/X]}$$

Must match expectation Y, provide instantiation $[Nat/Y]$

$\Gamma \vdash_{\Uparrow} 0 : [Nat/Y]Y$

Application Judgment (Syn)

$$\boxed{\Gamma \vdash^{\cdot} (Y \to X \times Y, [Nat \to Nat/X]) \cdot 0 : X \times [Nat/Y]\ Y \leadsto [Nat \to Nat/X]}$$

Use syn. type-arg in result type of app

Enforcement at Maximal Application

$$\Gamma \vdash^{P} \mathit{pair}\ (\lambda x.\, x)\ 0 : X \times \mathit{Nat} \rightsquigarrow [\mathit{Nat} \to \mathit{Nat}/X]$$

Earlier I said "enforce locality, contextuality..." how?

Enforcement at Maximal Application

$$\dfrac{\Gamma \vdash^{P} \mathit{pair}\ (\lambda\, x.\, x)\ 0 : X \times \mathit{Nat} \leadsto [\mathit{Nat} \to \mathit{Nat}/X] \qquad \mathit{dom}([\mathit{Nat} \to \mathit{Nat}/X]) = X = \mathit{MV}(\Gamma, X \times \mathit{Nat})}{}$$

Earlier I said "enforce locality, contextuality..." how?

- All remaining meta-variables are solved by σ
  $MV(\Gamma, T)$: meta-vars of T wrt declared variables of Γ

Enforcement at Maximal Application

$$\frac{\Gamma \vdash^{\mathsf{P}} \mathit{pair}\ (\lambda\, x.\, x)\ 0 : X \times \mathit{Nat} \rightsquigarrow [\mathit{Nat} \to \mathit{Nat}/X] \qquad \mathit{dom}([\mathit{Nat} \to \mathit{Nat}/X]) = X = \mathit{MV}(\Gamma, X \times \mathit{Nat}) \qquad [\mathit{Nat} \to \mathit{Nat}/X]\ (X \times \mathit{Nat}) = {\color{red}(\mathit{Nat} \to \mathit{Nat}) \times \mathit{Nat}}}{}$$

Earlier I said "enforce locality, contextuality..." how?

- All remaining meta-variables are solved by σ
  $MV(\Gamma, T)$: meta-vars of T wrt declared variables of Γ
- Contextual solutions *really are* contextual

Enforcement at Maximal Application

$$\frac{\Gamma \vdash^{\mathsf{P}} \mathit{pair}\ (\lambda x.\, x)\ 0 : X \times \mathit{Nat} \rightsquigarrow [\mathit{Nat} \to \mathit{Nat}/X] \qquad \mathit{dom}([\mathit{Nat} \to \mathit{Nat}/X]) = X = \mathit{MV}(\Gamma, X \times \mathit{Nat}) \qquad [\mathit{Nat} \to \mathit{Nat}/X]\ (X \times \mathit{Nat}) = (\mathit{Nat} \to \mathit{Nat}) \times \mathit{Nat}}{\Gamma \vdash_{\Downarrow} \mathit{pair}\ (\lambda x.\, x)\ 0 : (\mathit{Nat} \to \mathit{Nat}) \times \mathit{Nat}}$$

Earlier I said "enforce locality, contextuality..." how?

- All remaining meta-variables are solved by σ
  $MV(\Gamma, T)$: meta-vars of T wrt declared variables of Γ
- Contextual solutions *really are* contextual
- We clear these conditions and can type the expression

Outline

Specificational System Properties

Sanity check wrt. internal language (System F; $\Gamma \vdash t : T$)

Specificational System Properties

Sanity check wrt. internal language (System F; $\Gamma \vdash t : T$)

- Soundness:

$$\Gamma \vdash_{\delta} t : T \leadsto e \text{ implies } \Gamma \vdash e : T$$

- Trivial completeness:

$$\Gamma \vdash e : T \text{ implies } \Gamma \vdash_{\Uparrow} e : T \leadsto e$$

Specificational System Properties (cont.)

- Typeability of the *external* language (i.e. type annotation requirements)
- Assume $\Gamma \vdash e : T$. Erase binder, type args to get external term t.
- $\Gamma \vdash_{\Uparrow} t : T \leadsto e$ when given

Specificational System Properties (cont.)

- Typeability of the *external* language (i.e. type annotation requirements)
- Assume $\Gamma \vdash e : T$. Erase binder, type args to get external term t.
- $\Gamma \vdash_{\Uparrow} t : T \leadsto e$ when given
  - Binder annotations to λs when its context or *spine-context* lack this info
  - Instantiations for "phantom" type-arguments
    $\forall X, Y. X \to X$
  - Enough info to "see" a term or type application
    e.g. applicand of type X given $[S]$ or t

Algorithmic system

- "Prototypes" track expected result type, num args to spine head

$$? \to ? \to Nat$$

Algorithmic system

- "Prototypes" track expected result type, num args to spine head

$$? \to ? \to Nat$$

- Matched against head type, produces a "decorated" function type

$$\forall X = Nat. \forall Y = Y. X \to Y \to X$$

Prototype Matching (Ex. 1)

Check *pair* $(\lambda x.\, x)$ 0 against $(Nat \rightarrow Nat) \times Nat$

Prototype Matching (Ex. 1)

Check *pair* $(\lambda x. x)$ 0 against $(Nat \rightarrow Nat) \times Nat$

Prototype:			$? \rightarrow$	$? \rightarrow$	$(Nat \rightarrow Nat) \times$	Nat
Head type:	$\forall X.$	$\forall Y.$	$X \rightarrow$	$Y \rightarrow$	$X \times$	Y

Prototype Matching (Ex. 1)

Check *pair* $(\lambda x.\, x)$ 0 against $(Nat \rightarrow Nat) \times Nat$

Prototype:			$? \rightarrow$	$? \rightarrow$	$(Nat \rightarrow Nat) \times$	Nat
Head type:	$\forall X.$	$\forall Y.$	$X \rightarrow$	$Y \rightarrow$	$X \times$	Y
Decoration:	$\forall X = Nat \rightarrow Nat.$	$\forall Y = Nat.$	$X \rightarrow$	$Y \rightarrow$	$X \times$	Y

Prototype Matching (Ex. 1)

Check *pair* $(\lambda x.\, x)$ 0 against $(Nat \to Nat) \times Nat$

Prototype:			$? \to$	$? \to$	$(Nat \to Nat) \times$	Nat
Head type:	$\forall X.$	$\forall Y.$	$X \to$	$Y \to$	$X \times$	Y
Decoration:	$\forall X = Nat \to Nat.$	$\forall Y = Nat.$	$X \to$	$Y \to$	$X \times$	Y

No "guessing" for contextual type-args.

Prototype Matching (Ex. 2)

Careful handling needed when prototype arity exceeds the spine head's

Check *id suc* 0 against type *Nat*

Prototype Matching (Ex. 2)

Careful handling needed when prototype arity exceeds the spine head's

Check *id suc* 0 against type *Nat*

Prototype:		$? \rightarrow$		$? \rightarrow$	$\textcolor{red}{Nat}$
Head type:	$\forall X$	$. X \rightarrow$	X		

Prototype Matching (Ex. 2)

Careful handling needed when prototype arity exceeds the spine head's

Check *id suc* 0 against type *Nat*

Prototype:		$? \rightarrow$		$? \rightarrow Nat$	
Head type:	$\forall X$	$. X \rightarrow$	X		
Decoration:	$\forall X = X$	$. X \rightarrow$	$(X,$	$? \rightarrow Nat)$	

Prototype Matching (Ex. 2)

Careful handling needed when prototype arity exceeds the spine head's

Check *id suc* 0 against type *Nat*

Prototype:		$? \rightarrow$		$? \rightarrow Nat$
Head type:	$\forall X$	$. X \rightarrow$	X	
Decoration:	$\forall X = X$	$. X \rightarrow$	$(X,$	$? \rightarrow Nat)$

- Don't know how to instantiate X, save for later

Prototype Matching (Ex. 2)

Careful handling needed when prototype arity exceeds the spine head's

Check *id* *suc* 0 against type *Nat*

Prototype:		$? \to$		$? \to Nat$
Head type:	$\forall X$	$. X \to$	X	
Decoration:	$\forall X = X$	$. X \to$	$(X,$	$? \to Nat)$

- Don't know how to instantiate X, save for later
- From synthesis instantiate X, then compare

  Match $Nat \to Nat$ with $? \to Nat$ once we reach first arg. *suc*

Algorithmic Systems Properties

$$\boxed{\Gamma \Vdash_\delta t : T \leadsto e}$$

- *Algorithmic:*
  The system is given as a set of syntax-directed inference rules
- *Equivalent to Specification:*
  - Soundness:

    $$\Gamma \Vdash_\delta t : T \leadsto e \text{ implies } \Gamma \vdash_\delta t : T \leadsto e$$

  - Completeness:

    $$\Gamma \vdash_\delta t : T \leadsto e \text{ implies } \Gamma \Vdash_\delta t : T \leadsto e$$

Algorithmic Systems Properties

$$\boxed{\Gamma \Vdash_\delta t : T \leadsto e}$$

- *Algorithmic:*
  The system is given as a set of syntax-directed inference rules
- *Equivalent to Specification:*
  - Soundness:

    $$\Gamma \Vdash_\delta t : T \leadsto e \text{ implies } \Gamma \vdash_\delta t : T \leadsto e$$

  - Completeness:

    $$\Gamma \vdash_\delta t : T \leadsto e \text{ implies } \Gamma \Vdash_\delta t : T \leadsto e$$

- ... even though we never mentioned prototype matching or "stuck" decorations in the spec!

Algorithmic System Properties

Spec. system		Alg. system
$\Gamma \vdash^{\mathsf{P}} t : T \rightsquigarrow \sigma$ $\Gamma \vdash^{\cdot} (T, \sigma) \cdot t : T' \rightsquigarrow \sigma'$	$\equiv$	$\Gamma; P \Vdash^{?} t : W \rightsquigarrow \sigma$ $\Gamma \Vdash^{\cdot} (W, \sigma) \cdot t : W' \rightsquigarrow \sigma'$ $\overline{X} \Vdash^{:=} T := P \Rightarrow (W, \sigma)$

Algorithmic System Properties

$$\boxed{\begin{array}{c} \Gamma \vdash^{\mathsf{P}} t : T \rightsquigarrow \sigma \\ \Gamma \vdash^{\cdot} (T, \sigma) \cdot t : T' \rightsquigarrow \sigma' \end{array}} \quad \equiv \quad \boxed{\begin{array}{c} \Gamma ; P \Vdash^{?} t : W \rightsquigarrow \sigma \\ \Gamma \Vdash^{\cdot} (W, \sigma) \cdot t : W' \rightsquigarrow \sigma' \\ \overline{X} \Vdash^{:=} T := P \Rightarrow (W, \sigma) \end{array}}$$

Spec. system

Alg. system

Future Work

Type inference algorithm is implemented in Cedille, a language with impredicativity, dependent types, and dependent intersections. A local type inference system will be a good foundation for considering the following extensions:

Future Work

Type inference algorithm is implemented in Cedille, a language with impredicativity, dependent types, and dependent intersections. A local type inference system will be a good foundation for considering the following extensions:

- *partial* type propagation a la "Colored Local Type Inference"
- higher-order type inference using matching
- inference for *erased* term arguments (Cedille feature)
- subsumption based on some form of "type containment"

Thanks!

Questions?