ON DECIDING TYPING IN BIDIRECTIONAL MARTIN–LÖF TYPE THEORY

MAKING TYPE-CHECKERS COMPLETE BY CONSTRUCTION

Neel KRISHNASWAMI, Meven LENNON-BERTRAND

Types – June 12th 2023
WHY TYPE-CHECKING?

Decidable type-checking is crucial practically and philosophically. Elaboration, unification, etc. are nice, but fundamentally incomplete. Hundreds of papers on conversion... but not so much about decidability of type-checking.
Decidable type-checking is crucial *practically* and *philosophically*.
Decidable type-checking is crucial practically and philosophically.

Elaboration, unification, etc. are nice, but fundamentally incompleteness.
Decidable type-checking is crucial practically and philosophically.

Elaboration, unification, etc. are nice, but fundamentally incomplete.

Hundreds of papers on conversion... but not so much about decidability of type-checking.
Decidable type-checking is crucial practically and philosophically.

Elaboration, unification, etc. are nice, but fundamentally incomplete.

Hundreds of papers on conversion... but not so much about decidability of type-checking.

This talk!
Bad news: type-checking for general dependent types is undecidable (Dowek, 2001).
Bad news: type-checking for general dependent types is undecidable (Dowek, 2001). Good news: we know how to cope!

Bad news: type-checking for general dependent types is undecidable (Dowek, 2001). Good news: we know how to cope!

Solution 1: Annotations

\[ \lambda x : A. t \]

COQ, LEAN...

All terms infer
Bad news: type-checking for general dependent types is undecidable (Dowek, 2001). Good news: we know how to cope!

Solution 1:
Annotations

\( \lambda x: A. t \)

COQ, LEAN...

All terms infer

Solution 2:
Restricted terms

\( \lambda x. t \)

AGDA...

Neutrals infer
Normal forms check
Bad news: type-checking for general dependent types is undecidable (Dowek, 2001). Good news: we know how to cope!

<table>
<thead>
<tr>
<th>Solution 1: Annotations</th>
<th>Solution 2: Restricted terms</th>
<th>Solution 3: Free-standing annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda x : A. t$</td>
<td>$\lambda x. t$</td>
<td>$\lambda x. t$ and $t :: A$</td>
</tr>
<tr>
<td>COQ, LEAN...</td>
<td>AGDA...</td>
<td>McBride, RED* family...</td>
</tr>
<tr>
<td>All terms infer</td>
<td>Neutrals infer</td>
<td>Inferring terms</td>
</tr>
<tr>
<td></td>
<td>Normal forms check</td>
<td>Checking terms</td>
</tr>
</tbody>
</table>
Bad news: type-checking for general dependent types is undecidable (Dowek, 2001). Good news: we know how to cope!

<table>
<thead>
<tr>
<th>Solution 1: Annotations</th>
<th>Solution 2: Restricted terms</th>
<th>Solution 3: Free-standing annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda x: A. t )</td>
<td>( \lambda x. t )</td>
<td>( \lambda x. t ) and ( t :: A )</td>
</tr>
<tr>
<td>COQ, LEAN...</td>
<td>AGDA...</td>
<td>McBride, RED* family...</td>
</tr>
<tr>
<td>All terms infer</td>
<td>Neutrals infer</td>
<td>Inferring terms</td>
</tr>
<tr>
<td></td>
<td>Normal forms check</td>
<td>Checking terms</td>
</tr>
</tbody>
</table>

Our goal: unified setting & completeness proof, formalized.
Inferring and checkable terms, in the syntax

\[c, A, B ::= \_ | \square_k | \Pi x: A.B | \lambda x. c\]

\[i ::= c :: A | x | i c | \lambda x: A. i\]
INFERRING AND CHECKABLE TERMS, IN THE SYNTAX

\[ c, A, B ::= i \mid \square_k \mid \Pi x: A.B \mid \lambda x. c \]

\[ i ::= c :: A \mid x \mid i. c \mid \lambda x: A. i \]

Solution 1

\[ (\lambda x. c) u \text{ is not even valid syntax!} \]

\[ \Gamma \vdash A \prec \Gamma, x: A \vdash t \succ B \]

\[ \Gamma \vdash \lambda x: A. t \succ \Pi x: A.B \]

\[ \Gamma \vdash u \prec A \]

\[ \Gamma \vdash (\lambda x: A. t) u \succ B [u :: A] \]

Complete, by construction.
\[ c, A, B ::= i | □_k | Πx: A.B | λ x. c \]

\[ i ::= c :: A | x | i c | λ x: A. i \]
INFERRING AND CHECKABLE TERMS, IN THE SYNTAX

$c, A, B ::= i \mid \Box_k \mid \Pi x: A.B \mid \lambda x. c$

$i ::= c :: A \mid x \mid i \cdot c \mid \lambda x: A. i$

Solution 3

(\lambda x. c) \cdot u is not even valid syntax!

Γ ⊢ A ▷ Γ, x: A ⊢ t ▷ B

Γ ⊢ \lambda x: A. t ▷ \Pi x: A.B

Γ ⊢ u ▷ A

Γ ⊢ (\lambda x: A. t) \cdot u ▷ B [u :: A]

Complete, by construction.
$c, A, B ::= i \mid \Box^k \mid \Pi x: A. B \mid \lambda x. c$

$i ::= c :: A \mid x \mid i c \mid \lambda x: A. i$

$(\lambda x. c) u$ is not even valid syntax!
INFERRING AND CHECKABLE TERMS, IN THE SYNTAX

\[ c, A, B ::= \_ \mid □_k \mid \Pi x: A.B \mid \lambda x. c \]

\[ i ::= c :: A \mid x \mid i.c \mid \lambda x: A.i \]

\( (\lambda x. c) u \) is not even valid syntax!

\[
\begin{align*}
\Gamma \vdash A \triangleleft & \quad \Gamma, x: A \vdash t \triangleright B \\
\Gamma \vdash \lambda x: A. t \triangleright \Pi x: A.B & \quad \Gamma \vdash u \triangleleft A \\
\Gamma \vdash (\lambda x: A. t) u \triangleright B[u :: A]
\end{align*}
\]
Inferring and Checkable Terms, in the Syntax

\[
c, A, B ::= \_ | \Box_k | \Pi x: A.B | \lambda x. c
\]

\[
i ::= c :: A | x | i c | \lambda x: A. i
\]

\((\lambda x. c) u\) is not even valid syntax!

\[
\frac{\Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash t \triangleright B}{\Gamma \vdash \lambda x: A. t \triangleright \Pi x: A.B}
\]

\[
\frac{\Gamma \vdash \lambda x: A. t \triangleright \Pi x: A.B \quad \Gamma \vdash u \triangleleft A}{\Gamma \vdash (\lambda x: A. t) u \triangleright B[u :: A]}
\]

Complete, by construction.
INFERRING AND CHECKABLE TERMS, IN THE SYNTAX

\[ c, A, B ::= _i | □k | Πx: A.B | \lambda x. c \quad \mid \Sigma x: A.B | \langle c, c \rangle | W x: A.B | \text{sup}(c, c) \]

\[ i ::= c :: A \mid x \mid i \ c \mid \lambda x: A. i \quad \mid i_1 \mid i_2 \mid \langle i, c \rangle_{x.B} \mid \text{ind}_W(i; x.A; c) \mid \text{sup}_{x.B}(i, c) \]

\((\lambda x. c) u\) is not even valid syntax!

\[ \Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash t \triangleright B \]

\[ \Gamma \vdash \lambda x: A. t \triangleright \Pi x: A.B \quad \Gamma \vdash u \triangleleft A \]

\[ \Gamma \vdash (\lambda x: A. t) u \triangleright B[u :: A] \]

Complete, by construction.
Beware of substitutions!

\[(\lambda \, x : A. \, t) \, u \rightarrow t[u] \quad \times\]
Beware of substitutions!

$$(\lambda x: A. t) \ u \rightarrow t[u :: A] \quad \checkmark$$
Beware of substitutions!

$$\lambda x: A. t \ u \rightarrow t[u :: A]$$

Annotations reduce, type-directed (see observational equality, gradual cast calculus...):

$$(\lambda x. t) :: \Pi x: A. B) \ u \rightarrow (\lambda x: A. (t :: B)) \ u \rightarrow (t[u :: A]) :: B[u :: A]$$

(Stuck) annotations should be ignored (TT\ sub again):

$$\Gamma \vdash n \approx n' \triangleright A \Gamma \vdash n :: A' \approx n' \triangleright A'$$
Beware of substitutions!

\[(\lambda x: A. t) \ u \rightarrow t[u :: A] \quad \checkmark\]

Annotations reduce, type-directed (see observational equality, gradual cast calculus...):

\[((\lambda x. t) :: \Pi x: A.B) \ u \rightarrow (\lambda x: A. (t :: B)) \ u \rightarrow (t[u :: A]) :: B[u :: A]\]

Conversion, too, is bidirectional (Abel et al., 2018):

\[\Gamma \vdash A \cong A' \quad \text{and} \quad \Gamma \vdash c \cong c' \triangleleft A \quad \text{but} \quad \Gamma \vdash n \cong n' \triangleright A\]
Beware of substitutions!

\[ (\lambda x: A. t) \ u \rightarrow t[u :: A] \]  
✓

Annotations reduce, type-directed (see observational equality, gradual cast calculus...):

\[ (((\lambda x. t) :: \Pi x: A.B) \ u \rightarrow (\lambda x: A. (t :: B)) \ u \rightarrow (t[u :: A]) :: B[u :: A] \]

Conversion, too, is bidirectional (Abel et al., 2018):

\[ \Gamma \vdash A \equiv A' \quad \text{and} \quad \Gamma \vdash c \equiv c' \triangleleft A \quad \text{but} \quad \Gamma \vdash n \approx n' \triangleright A \]

(Stuck) annotations should be ignored (TT\textsuperscript{obs} again):

\[
\frac{
\Gamma \vdash n \approx n' \triangleright A
}{
\Gamma \vdash n :: A' \approx n' \triangleright A'
}\]
SHOULD WE COMPUTE ANNOTATIONS?

Annotation-free terms are **exactly** the normal/neutral forms:

\[ c, A, B ::= i \mid \Box_k \mid \Pi x: A.B \mid \lambda x. c \mid \Sigma x: A.B \mid \langle c, c \rangle \mid W x: A.B \mid \text{sup}(c, c) \]

\[ i ::= c :: A \mid x \mid i\ c \mid \lambda x: A.i \mid i.1 \mid i.2 \mid \langle i, c \rangle_x.B \mid \text{ind}_{W}(i; x.A; c) \mid \text{sup}_{x.B}(i, c) \]

Why?
Annotation-free terms are exactly the normal/neutral forms:

\[
c, A, B ::= i \mid \Box_k \mid \Pi x: A.B \mid \lambda x. c \mid \Sigma x: A.B \mid \langle c, c \rangle \mid W x: A.B \mid \sup(c, c)
\]

\[
i ::= c :: A \mid x \mid i c \mid \lambda x: A.i \mid i_1 \mid i_2 \mid \langle i, c \rangle_x.B \mid \text{ind}_W(i; x.A; c) \mid \sup_{x.B}(i, c)
\]

Why work hard to maintain annotations we know will disappear?
Annotation-free terms are exactly the normal/neutral forms:

\[
c, A, B ::= i \mid \Box_k \mid \Pi x: A. B \mid \lambda x. c \mid \Sigma x: A. B \mid \langle c, c \rangle \mid W x: A. B \mid \sup(c, c)
\]

\[
i ::= c :: A \mid x \mid i \cdot c \mid \lambda x: A. i \mid i_1 \mid i_2 \mid \langle i, c \rangle_{x. B} \mid \text{ind}_W(i; x. A; c) \mid \sup_{x. B}(i, c)
\]

Why work hard to maintain annotations we know will disappear?

- Typable intermediate computation steps are nice…
- but if we only care about fast comparison, we should not bother.
The plan is laid out... and the formalization is ongoing.
WHERE ARE WE AT?

The plan is laid out... and the formalization is ongoing.

Luckily, we already have formalized logical relations for ~Solution 1, in Coq. Listen to Kenji at 15:00!
THANK YOU!

\[ c, A, B ::= \_ | \Box_k | \Pi x: A.B | \lambda x. c | \Sigma x: A.B | \langle c, c \rangle | W x: A.B | \sup(c, c) \]

\[ i ::= c :: A | x | i c | \lambda x: A.i | i.1 | i.2 | \langle i, c \rangle_{x.B} | \ind_W(i; x.A; c) | \sup_{x.B}(i, c) \]