Type Theory

Lecture 3: Dependent Type Theory

H. Geuvers

Radboud University
Nijmegen, NL

OPLSS 2026 Summer 2026

Outline

The idea of dependent types

Dependent Type Theory

Logical Framework

Type Checking Algorithm

Where we are in the course

propositional logic	$\leftrightarrow$	simple type theory $\lambda{\rightarrow}$
2nd order propositional logic	$\leftrightarrow$	polymorphic type theory $\lambda 2$
predicate logic	$\leftrightarrow$	type theory with dependent types λP
higher order predicate logic	$\leftrightarrow$	calculus of constructions λC
univalent foundations	$\leftrightarrow$	homotopy type theory

Extending formulas-as-types to predicate logic

- First order language: domain D, with variables $x, y, z : D$ and possibly functions over D, e.g. $f : D \to D$, $g : D \to D \to D$.
- There are two "kinds" of variables:
  - the first order variables (ranging over the domain D)
  - the "proof variables" (used as [local] assumptions of formulas).
- Formulas and domains are both types. What is the type of a predicate or relation?
- A predicate P is a map from D to the collection of types, $*$
- $P : D \to *$ for P a predicate and $R : D \to D \to *$ for R a binary relation on D.
- We will have to make this more precise ...

Idea of extending to $\forall$

We introduce term rules for the $\forall$-quantifier in predicate logic.

$$\frac{\Gamma \vdash M : \forall x{:}D.A}{\Gamma \vdash M\,t : A[x := t]} \text{ if } t : D \qquad \frac{\Gamma \vdash M : A}{\Gamma \vdash \lambda x{:}D.M : \forall x{:}D.A}\, x \text{ not free in } \Gamma$$

With the usual β-reduction rule

$$(\lambda x{:}D.M)t \to M[x := t]$$

.

This conforms with cut-elimination (or "detour elimination") on logical derivations.

Example

Deriving irreflexivity from anti-symmetry

$$\begin{aligned}\text{AntiSym}\, R &:= \forall x, y{:}D.(R\,x\,y)\to(R\,y\,x)\to\bot\\ \text{Irrefl}\, R &:= \forall x{:}D.(R\,x\,x)\to\bot\end{aligned}$$

Derivation in predicate logic:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall x, y{:}D.R\,x\,y\to R\,y\,x\to\bot}{\forall y{:}D.R\,x\,y\to R\,y\,x\to\bot}}{R\,x\,x\to R\,x\,x\to\bot} \qquad [R\,x\,x]^1}{R\,x\,x\to\bot} \qquad [R\,x\,x]^1}{\bot}}{R\,x\,x\to\bot}\,1}{\forall x{:}D.R\,x\,x\to\bot}$$

Example derivation in type theory, with terms

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{H : \forall x, y{:}D. R\,x\,y \to R\,y\,x \to \bot}{H\,x : \forall y{:}D. R\,x\,y \to R\,y\,x \to \bot}}{H\,x\,x : R\,x\,x \to R\,x\,x \to \bot} \qquad [H' : R\,x\,x]^1}{H\,x\,x\,H' : R\,x\,x \to \bot} \qquad [H' : R\,x\,x]^1}{H\,x\,x\,H'\,H' : \bot}\,1}{\lambda H'{:}(R\,x\,x). H\,x\,x\,H'\,H' : R\,x\,x \to \bot}}{\lambda x{:}A. \lambda H'{:}(R\,x\,x). H\,x\,x\,H'\,H' : \forall x{:}D. R\,x\,x \to \bot}
$$

Main difference between $\lambda\!\to$ and λP

$$A \to B$$

'type of functions from A to B'

$$\Pi x : A.\, B$$

'type of functions from A to B'

dependent product
dependent function type

The type of the function value B now can depend on the function argument x
arrow type becomes a special case.Examples of Π-types:

- $\forall x : \mathbb{N}.x \geq 0$. A proof of $\forall x : \mathbb{N}.x \geq 0$ is a term $p : \Pi x : \mathbb{N}.x \geq 0$, a function that for every $n : \mathbb{N}$ gives a term $p\, n : n \geq 0$.
- $\Pi x : \mathbb{N}.A^x$. A term $t : \Pi x : \mathbb{N}.A^x$ is a function that for every $n : \mathbb{N}$ produces a vectore over A of length n.

Dependent Type Theory

- We have seen informally “dependent types at work” in the predicate logic example.
- Now: the rules

With dependent types:

- everything depends on everything
- we can't first define the types, then the terms
- two universes: $*$ and $\Box$
- $*$ is the universe of types
- We can't have $* : *$, so we have another universe: $* : \Box$.

NB The Coq system uses “Set” and “Prop” for what I call $*$ and “Type” for what I call $\Box$.

First order Dependent Type theory, λP

Derive judgements of the form

$$\Gamma \vdash M : B$$

- Γ is a context

$$x_1 : B_1, x_2 : B_2, \ldots, x_n : B_n$$

- M and B are terms
  taken from the set of pseudoterms

$$\mathsf{T} ::= \mathsf{Var} \mid * \mid \Box \mid (\mathsf{T}\,\mathsf{T}) \mid (\lambda x{:}\mathsf{T}.\mathsf{T}) \mid \Pi x{:}\mathsf{T}.\mathsf{T}$$

Auxiliary judgement

$$\Gamma \vdash$$

denoting that Γ is a correct context.

Derivation rules of λP

$\mathbf{s}$ ranges over $\{*, \Box\}$.

(base) $\emptyset \vdash$ (ctxt) $\dfrac{\Gamma \vdash A : \mathbf{s}}{\Gamma, x{:}A \vdash}$ if x not in Γ (ax) $\dfrac{\Gamma \vdash}{\Gamma \vdash * : \Box}$

(proj) $\dfrac{\Gamma \vdash}{\Gamma \vdash x : A}$ if $x{:}A \in \Gamma$ (Π) $\dfrac{\Gamma \vdash A : * \quad \Gamma, x{:}A \vdash B : \mathbf{s}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{s}}$

(λ) $\dfrac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash \Pi x{:}A.B : \mathbf{s}}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$ (app) $\dfrac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$

(conv) $\dfrac{\Gamma \vdash M : B \quad \Gamma \vdash A : \mathbf{s}}{\Gamma \vdash M : A} A =_\beta B$

Notation: write $A \rightarrow B$ for $\Pi x{:}A.B$ if $x \notin \mathrm{FV}(B)$.

The use of the Π-type

- The Π rule allows to form two forms of function types.

$$(\Pi)\ \frac{\Gamma, x{:}A \vdash B : \mathbf{s} \quad \Gamma \vdash A : *}{\Gamma \vdash \Pi x{:}A.B : \mathbf{s}}$$

$$\Pi x{:}A.B \simeq \{f \mid \forall a : A(f\ a : B[x := a])\}$$

Write $A{\rightarrow}B$ if $x \notin \mathrm{FV}(B)$

- With $\mathbf{s} = *$, we can form $\mathrm{D}{\rightarrow}\mathrm{D}$ and $\Pi x{:}\mathrm{D}.x = x$, etc.
- With $\mathbf{s} = \Box$, we can form $\mathrm{D}{\rightarrow}\mathrm{D}{\rightarrow}*$ and $\mathrm{D}{\rightarrow}*$.

Representation of PRED into λP

PRED = minimal predicate logic, that is: intuitionistic predicate logic with just $\to$ and $\forall$.
Represent both the domains of the logic and the formulas as types.

$$\begin{array}{lcl} \mathsf{A} & : & *, \\ P & : & \mathsf{A}\to *, \\ R & : & \mathsf{A}\to\mathsf{A}\to *, \end{array}$$

Now implication is represented as $\to$ and $\forall$ is represented as Π:

$$\forall x{:}A.P\,x \mapsto \Pi x{:}A.P\,x$$

Intro and elim rules are just λ-abstraction and application

Example

$$\mathsf{A} : *, R : \mathsf{A}\to\mathsf{A}\to* \quad \vdash \quad \lambda z{:}A.\lambda h{:}(\Pi x, y{:}A.R\,x\,y).h\,z\,z$$
$$: \quad \Pi z{:}A.(\Pi x, y{:}A.R\,x\,y)\to R\,z\,z$$

This term is a proof of $\forall z{:}A.(\forall x, y{:}A.R(x, y))\to R(z, z)$

Exercise: Find terms of the following types (NB $\to$ binds strongest)

$$(\Pi x{:}A.P\,x\to Q\,x)\to(\Pi x{:}A.P\,x)\to\Pi x{:}A.Q\,x$$

and

$$(\Pi x{:}A.P\,x\to\Pi z.R\,z\,z)\to(\Pi x{:}A.P\,x)\to\Pi z{:}A.R\,z\,z).$$

Also write down the contexts in which these terms are typed.

Correctness of embedding PRED into λP

For Σ a first order signature, let Γ_Σ denote the associated λP-context.
For φ a PRED-formula, let $\overline{\varphi}$ denote the associated λP-type, and similarly $\overline{\Gamma}$ be a declaration context in λP for a set of formulas Γ.

THEOREM.
For Σ is a first order signature, and Γ a finite set of formulas and φ a formula, we have

$$\Gamma \vdash \varphi \quad \Longleftrightarrow \quad \exists M(\Gamma_\Sigma, \overline{\Gamma} \vdash M : \overline{\varphi})$$

PROOF.
$\Rightarrow$ by induction on the derivation we can construct M as the proof-term that represents the derivation of $\Gamma \vdash \varphi$
$\Leftarrow$ is more complicated. There are terms M that do not "come from a proof" in PRED; this is far from an isomorphism. $\square$

Direct embedding of logic in type theory

For $\lambda\!\rightarrow$ and λP we have seen

Direct representations of logic in type theory.

- Connectives each have a counterpart in the type theory:

implication	$\sim$	$\rightarrow$-type
universal quantification	$\sim$	$\forall$-type

- Logical rules have their direct counterpart in type theory

$\rightarrow$-introduction	$\sim$	λ-abstraction
$\rightarrow$- elimination	$\sim$	application
$\forall$-introduction	$\sim$	λ-abstraction
$\forall$-elimination	$\sim$	application

- Context declares signature, local varibales and assumptions.

LF embedding of logic in type theory

Second way of interpreting logic in type theory De Bruijn; Harper, Honsell, Plotkin:

Logical framework encoding or higher order abstract syntax embedding of logic in type theory.

- Type theory used as a meta system for encoding ones own logic.
- Choose an appropriate context Γ_L, in which the logic L (including its proof rules) is declared.
- Context used as a signature for the logic.
- Use the type system as the 'meta' calculus for dealing with substitution and binding.

Direct and LF embedding

	proof	formula
direct embedding	$\lambda x{:}A.x$	$A \to A$
LF embedding	$\mathrm{imp_intr}\, A\, A\, \lambda x{:}T\, A.x$	$T(A \Rightarrow A)$

- Direct representation:
  One type system : One logic,
  Logical rules $\sim$ type theoretic rules
- LF encoding: One type system : Many logics,
  Logical rules $\sim$ context declarations

Examples of the LF embedding

The encoding of logics in a logical framework is shown by three examples:

1. Minimal proposition logic
2. Minimal predicate logic (just $\{\Rightarrow, \forall\}$)
3. Untyped λ-calculus

Minimal propositional logic

Fix the signature (context) of minimal propositional logic.

$$\begin{array}{rcl} \textbf{prop} & : & * \\ \text{imp} & : & \textbf{prop} \to \textbf{prop} \to \textbf{prop} \end{array}$$

Notation:

$$A \Rightarrow B \text{ for } \text{imp}\, A\, B$$

The type **prop** is the type of 'names' of propositions.
NB : A term of type **prop** can not be inhabited (proved), as it is not a type.
We 'lift' a name p : **prop** to the type of its proofs by declaring the following map:

$$\mathsf{T} \;:\; \textbf{prop} \to * \,.$$

Intended meaning of $\mathsf{T}p$ is 'the type of proofs of p'.
We interpret 'p is valid' by '$\mathsf{T}p$ is inhabited'.

Encoding of derivations

To derive $\mathsf{T}p$ we also encode the logical derivation rules

$$\begin{aligned} \mathsf{imp_intr} &: \Pi p, q : \mathsf{prop}.(\mathsf{T}p \to \mathsf{T}q) \to \mathsf{T}(p \Rightarrow q), \\ \mathsf{imp_el} &: \Pi p, q : \mathsf{prop}.\mathsf{T}(p \Rightarrow q) \to \mathsf{T}p \to \mathsf{T}q. \end{aligned}$$

imp_intr takes two (names of) propositions p and q and a term $f : \mathsf{T}\,p \to \mathsf{T}\,q$ and returns a term of type $\mathsf{T}(p \Rightarrow q)$
Indeed $A \Rightarrow A$, becomes valid:

$$\mathsf{imp_intr} A\, A (\lambda x{:}\mathsf{T}\, A.x) : \mathsf{T}(A \Rightarrow A)$$

Exercise: Construct a term of type $\mathsf{T}(A \Rightarrow (B \Rightarrow A))$

Signature of PROP in LF

To encode proposition logic in LF we need a context (signature) Σ_{PROP}:

$$\begin{array}{rcl} \textbf{prop} & : & * \\ \Rightarrow & : & \textbf{prop} \rightarrow \textbf{prop} \rightarrow \textbf{prop} \\ \mathsf{T} & : & \textbf{prop} \rightarrow * \\ \mathsf{imp_intr} & : & (A, B : \textbf{prop})(\mathsf{T}\, A \rightarrow \mathsf{T}\, B) \rightarrow \mathsf{T}(A \Rightarrow B) \\ \mathsf{imp_el} & : & (A, B : \textbf{prop})\mathsf{T}(A \Rightarrow B) \rightarrow \mathsf{T}\, A \rightarrow \mathsf{T}\, B. \end{array}$$

Desired propertiy of the encoding: Adequacy, in particular

- Soundness of the encoding:

  $$\vdash_{\text{PROP}} A \quad \Longrightarrow \quad \Sigma_{\text{PROP}}, \vec{a}{:}\textbf{prop} \vdash p : \mathsf{T}\, A \text{ for some } p.$$

  $\vec{a}$ is the sequence of proposition variables in A.
- Faithfulness (or completeness) is the converse. It also holds, but more involved to prove.

Minimal predicate logic over one domain A

$$
\begin{array}{rcl}
\textbf{prop} & : & *, \\
\mathsf{A} & : & *, \\
\mathsf{T} & : & \textbf{prop} \rightarrow * \\
\mathsf{f} & : & \mathsf{A} \rightarrow \mathsf{A}, \\
\mathsf{R} & : & \mathsf{A} \rightarrow \mathsf{A} \rightarrow \textbf{prop}, \\
\Rightarrow & : & \textbf{prop} \rightarrow \textbf{prop} \rightarrow \textbf{prop}, \\
\mathsf{imp_intr} & : & \Pi p, q : \textbf{prop}.(\mathsf{T}p \rightarrow \mathsf{T}q) \rightarrow \mathsf{T}(p \Rightarrow q), \\
\mathsf{imp_el} & : & \Pi p, q : \textbf{prop}.\mathsf{T}(p \Rightarrow q) \rightarrow \mathsf{T}p \rightarrow \mathsf{T}q.
\end{array}
$$

Now encode $\forall$:

$\forall$ takes a $P : \mathsf{A} \rightarrow \textbf{prop}$ and returns a proposition, so we add:

$$\forall : (\mathsf{A} \rightarrow \textbf{prop}) \rightarrow \textbf{prop}$$

Universal quantification is translated as follows.

$$\forall x{:}A.(Px) \mapsto \forall(\lambda x{:}A.(Px))$$

Intro and Elim rules for ∀

$$
\begin{array}{rcl}
\forall & : & (\mathrm{A}\to\mathbf{prop})\to\mathbf{prop}, \\
\forall_\mathrm{intr} & : & \Pi P{:}\mathrm{A}\to\mathrm{prop}.(\Pi x{:}\mathrm{A}.\mathrm{T}(Px))\to\mathrm{T}(\forall P), \\
\forall_\mathrm{elim} & : & \Pi P{:}\mathrm{A}\to\mathrm{prop}.\mathrm{T}(\forall P)\to\Pi x{:}\mathrm{A}.\mathrm{T}(Px).
\end{array}
$$

The proof of

$$\forall z{:}A(\forall x, y{:}A.Rxy) \Rightarrow Rzz$$

is now mirrored by the proof-term

$$
\begin{array}{ll}
\forall_\mathrm{intr}[_](& \lambda z{:}\mathrm{A}.\mathrm{imp_intr}[_][_](\lambda h{:}\mathrm{T}(\forall x, y{:}A.Rxy). \\
 & \forall_\mathrm{elim}[_](\forall_\mathrm{elim}[_]hz)z)\ )
\end{array}
$$

We have replaced the instantiations of the Π-type by [_].
This term is of type

$$\mathrm{T}(\forall(\lambda z{:}\mathrm{A}.\mathrm{imp}(\forall(\lambda x{:}\mathrm{A}.(\forall(\lambda y{:}\mathrm{A}.\mathrm{R}xy))))(\mathrm{R}zz)))$$

Intro and Elim rules for $\forall$

$$\begin{aligned}
\forall &: (\mathrm{A}\to\mathbf{prop})\to\mathbf{prop},\\
\forall_\mathrm{intr} &: \Pi P{:}\mathrm{A}\to\mathrm{prop}.(\Pi x{:}\mathrm{A}.\mathrm{T}(P\,x))\to\mathrm{T}(\forall P),\\
\forall_\mathrm{elim} &: \Pi P{:}\mathrm{A}\to\mathrm{prop}.\mathrm{T}(\forall P)\to\Pi x{:}\mathrm{A}.\mathrm{T}(P\,x).
\end{aligned}$$

The proof of

$$\forall z{:}A\,(\forall x,y{:}A.R\,x\,y)\Rightarrow R\,z\,z$$

is now mirrored by the proof-term

$$\begin{aligned}
\forall_\mathrm{intr}[_](\quad & \lambda z{:}\mathrm{A}.\mathrm{imp_intr}[_][_](\lambda h{:}\mathrm{T}(\forall x,y{:}A.Rxy).\\
& \forall_\mathrm{elim}[_](\forall_\mathrm{elim}[_]hz)z)\ )
\end{aligned}$$

Exercise: Construct a proof-term that mirrors the (obvious) proof of

$$\forall x(P\,x\Rightarrow Q\,x)\Rightarrow\forall x.P\,x\Rightarrow\forall x.Q\,x$$

Untyped λ-calculus

$$\text{Signature } \Sigma_{\mathsf{lambda}} : \quad \begin{array}{rcl} \mathsf{D} & : & *; \\ \mathsf{app} & : & \mathsf{D}\to(\mathsf{D}\to\mathsf{D}); \\ \mathsf{abs} & : & (\mathsf{D}\to\mathsf{D})\to\mathsf{D}. \end{array}$$

- A variable x in λ-calculus becomes $x : \mathsf{D}$ in the type system.
- The translation $[-] : \Lambda \to \mathsf{Term}(\mathsf{D})$ is defined as follows.

$$\begin{array}{rcl} [x] & = & x; \\ [PQ] & = & \mathsf{app}\,[P]\,[Q]; \\ [\lambda x.P] & = & \mathsf{abs}\,(\lambda x{:}\mathsf{D}.[P]). \end{array}$$

Examples: $[\lambda x.x\,x] := \mathsf{abs}(\lambda x{:}\mathsf{D}.\mathsf{app}\,x\,x)$
$[(\lambda x.x\,x)\,(\lambda y.y)] := \mathsf{app}\,(\mathsf{abs}\,(\lambda x{:}\mathsf{D}.\mathsf{app}\,x\,x))\,(\mathsf{abs}\,(\lambda y{:}\mathsf{D}.y))$.

Introducing β-equality

$$\mathsf{eq}{:}\mathsf{D}\to\mathsf{D}\to *\,.$$

Notation $P = Q$ for eq P Q.
Rules for proving equalities.

$$\begin{array}{rcl}
\mathsf{refl} & : & \Pi x{:}\mathsf{D}.x = x, \\
\mathsf{sym} & : & \Pi x, y{:}\mathsf{D}.x = y \to y = x, \\
\mathsf{trans} & : & \Pi x, y, z{:}\mathsf{D}.x = y \to y = z \to x = z, \\
\mathsf{mon} & : & \Pi x, x', z, z'{:}\mathsf{D}.x = x' \to z = z' \to \mathsf{app}\ z\ x = \mathsf{app}\ z'\ x', \\
\mathsf{xi} & : & \Pi f, g{:}\mathsf{D}\to\mathsf{D}.(\Pi x{:}\mathsf{D}.f\,x = g\,x) \to \mathsf{abs}\ f = \mathsf{abs}\ g, \\
\mathsf{beta} & : & \Pi f{:}\mathsf{D}\to\mathsf{D}.\Pi x{:}\mathsf{D}.\mathsf{app}\ (\mathsf{abs}\ f)\,x = f\,x.
\end{array}$$

Properties of λP

- Uniqueness of types
  If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_\beta B$.
- Subject Reduction
  If $\Gamma \vdash M : A$ and $M \rightarrow_\beta N$, then $\Gamma \vdash N : A$.
- Strong Normalization
  If $\Gamma \vdash M : A$, then all β-reductions from M terminate.

Proof of SN is by defining a reduction preserving map from λP to $\lambda\!\rightarrow$.

Decidability Questions

$\Gamma \vdash M : A?$	TCP
$\Gamma \vdash M : ?$	TSP
$\Gamma \vdash ? : A$	TIP

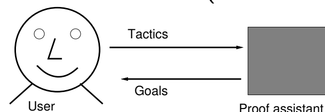
For λP:

- TIP is undecidable
- TCP/TSP: simultaneously with Context checking

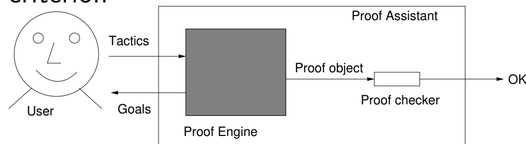
Why would we believe a proof assistant?

Separating the simple proof checker from the powerful proof engine

Proof Assistant (Interactive Theorem Prover)



Proof Assistant with a small kernel that satisfies the De Bruijn criterion



Type Checking (= proof checking)

Define algorithms $\mathrm{Ok}(-)$ and $\mathsf{TC}_{_}(-)$ simultaneously:

- $\mathrm{Ok}(-)$ takes a context and returns 'true' or 'false'
- $\mathsf{TC}_{_}(-)$ takes a context and a term and returns a term or 'false'.

DEFINITION The type synthesis algorithm $\mathsf{TC}_{_}(-)$ is sound if (for all Γ and M)

$$\mathsf{TC}_{\Gamma}(M) = A \implies \Gamma \vdash M : A$$

The type synthesis algorithm $\mathsf{TC}_{_}(-)$ is complete if (for all Γ, M and A)

$$\Gamma \vdash M : A \implies \mathsf{TC}_{\Gamma}(M) =_{\beta} A$$

- A proof assistant like Rocq is based on a type checking algorithm.
- The type checking algorithm is the trusted kernel of Rocq

$$
\begin{array}{rcl}
\mathrm{Ok}(<>) & = & \text{'true'} \\[2ex]
\mathrm{Ok}(\Gamma, x{:}A) & = & \mathsf{TC}_\Gamma(A) \in \{*, \Box\}, \\[2ex]
\mathsf{TC}_\Gamma(x) & = & \text{if } \mathrm{Ok}(\Gamma) \text{ and } x{:}A \in \Gamma \text{ then } A \text{ else 'false'}, \\[2ex]
\mathsf{TC}_\Gamma(*) & = & \text{if } \mathrm{Ok}(\Gamma)\text{then } \Box \text{ else 'false'}, \\[2ex]
\mathsf{TC}_\Gamma(MN) & = & \text{if } \mathsf{TC}_\Gamma(M) = C \text{ and } \mathsf{TC}_\Gamma(N) = D \\
 & & \quad \text{then} \quad \text{if } C \twoheadrightarrow_\beta \Pi x{:}A.B \text{ and } A =_\beta D \\
 & & \quad \qquad\qquad \text{then } B[x := N] \text{ else 'false'} \\
 & & \quad \text{else} \quad\ \text{'false'},
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{TC}_\Gamma(\lambda x{:}A.M) & = & \text{if } \mathsf{TC}_{\Gamma,x:A}(M) = B \\
& & \text{then} \quad \begin{array}{l} \text{if } \mathsf{TC}_\Gamma(\Pi x{:}A.B) \in \{*, \Box\} \\ \text{then } \Pi x{:}A.B \text{ else `false'} \end{array} \\
& & \text{else `false'}, \\
\mathsf{TC}_\Gamma(\Pi x{:}A.B) & = & \text{if } \mathsf{TC}_\Gamma(A) = * \text{ and } \mathsf{TC}_{\Gamma,x:A}(B) = s \\
& & \text{then } s \text{ else `false'}
\end{array}
$$

Soundness and Completeness

Soundness

$$\mathsf{TC}_\Gamma(M) = A \quad\Longrightarrow\quad \Gamma \vdash M : A$$

Completeness

$$\Gamma \vdash M : A \quad\Longrightarrow\quad \mathsf{TC}_\Gamma(M) =_\beta A$$

As a consequence:

$$\mathsf{TC}_\Gamma(M) = \text{‘false’} \quad\Longrightarrow\quad M \text{ is not typable in } \Gamma$$

NB 1. Completeness implies that TC terminates on all well-typed terms. We want that TC terminates on all pseudo terms.
NB 2. Completeness only makes sense if we have uniqueness of types
(Otherwise: let $\mathsf{TC}_{_}(-)$ generate a set of possible types)

Termination

We want $\mathsf{TC}_{_}(-)$ to terminate on all inputs.
Interesting cases: λ-abstraction and application:

$$\begin{array}{rcl}\mathsf{TC}_\Gamma(\lambda x{:}A.M) & = & \text{if } \mathsf{TC}_{\Gamma,x:A}(M) = B \\ & & \quad \text{then} \qquad \text{if } \mathsf{TC}_\Gamma(\Pi x{:}A.B) \in \{*, \Box\} \\ & & \qquad\qquad\qquad \text{then } \Pi x{:}A.B \text{ else `false'} \\ & & \quad \text{else `false'},\end{array}$$

! Recursive call is not on a smaller term!
Replace the side condition

$$\text{if } \mathsf{TC}_\Gamma(\Pi x{:}A.B) \in \{*, \Box\}$$

by

$$\text{if } \mathsf{TC}_\Gamma(A) \in \{*\}$$

Termination

We want $\mathsf{TC}_{_}(-)$ to terminate on all inputs.
Interesting cases: λ-abstraction and application:

$$
\begin{array}{rcl}
\mathsf{TC}_\Gamma(MN) & = & \text{if } \mathsf{TC}_\Gamma(M) = C \text{ and } \mathsf{TC}_\Gamma(N) = D \\
 & & \quad \text{then} \quad \text{if } C \twoheadrightarrow_\beta \Pi x{:}A.B \text{ and } A =_\beta D \\
 & & \quad \phantom{\text{then}} \quad \text{then } B[x := N] \text{ else `false'} \\
 & & \quad \text{else} \quad \text{`false'},
\end{array}
$$

! Need to decide β-reduction and β-equality!
For this case, termination follows from:

- Soundness of TC and
- Decidability of equality on well-typed terms.

This decidability of equality follows from SN (strong normalization) and CR (Church-Rosser property).

Questions?