Implementing Type Checkers for Dependent Type Theories

Zoe Stafford

Oxford Uni/Aarhus Uni

LogSem Seminar 2025-09-15

Background

What is a type checker?

What makes type checking hard?

How type check?

Motivation

- Check your proofs

$$0 = \infty - \infty = (1 + \infty) - \infty = 1 + (\infty - \infty) = 1$$

Motivation

- Check your proofs

$$0 = \infty - \infty = (1 + \infty) - \infty = 1 + (\infty - \infty) = 1$$

- Verify code

```
let days = [ "Mon", "Tue", "Wed", "Thu"
           , "Fri", "Sat", "Sun" ];
let today = days[8]; // Oh no!
```

Our Type Theory

- Martin Löf type theory
- Based on lambda calculus
- Underlying type theory of Idris, Agda, Rocq

Our Type Theory

- Dependent product: $(x : A) \to B$
- Dependent sum: $\sum(x : A)B$

```
record Sigma (A : Set) (B : A → Set) : Set where
  constructor _,_
  field
    fst : A
    snd : B fst
```

- Bool with eliminator $\mathsf{If}(y.M; T; F; b)$
- Intensional equality types: $\mathsf{Id}_A(x, y)$
- Universe U

Our Type Theory

Example

$$\mathsf{not} \ : \ \mathsf{Bool} \to \mathsf{Bool}$$
$$\mathsf{not} = \lambda(x : \mathsf{Bool}).\, \mathsf{If}(y.\, \mathsf{Bool}; \mathsf{false}; \mathsf{true}; x)$$

$$\mathsf{not\_not} \ : \ (x : \mathsf{Bool}) \to \mathsf{Id}_{\mathsf{Bool}}(\mathsf{not}(\mathsf{not}(x)), x)$$
$$\mathsf{not\_not} = \lambda(x : \mathsf{Bool}).\, \mathsf{If}(y.\, \mathsf{Id}_{\mathsf{Bool}}(\mathsf{not}(\mathsf{not}(y)), y); \mathsf{refl}(\mathsf{true}); \mathsf{refl}(\mathsf{false}); x)$$

Our Type Theory

Example

$$\mathsf{not} \;:\; \mathsf{Bool} \to \mathsf{Bool}$$
$$\mathsf{not} = \lambda(x : \mathsf{Bool}).\, \mathsf{If}(y.\, \mathsf{Bool}; \mathsf{false}; \mathsf{true}; x)$$

$$\mathsf{not\_not} \;:\; (x : \mathsf{Bool}) \to \mathsf{Id}_{\mathsf{Bool}}(\mathsf{not}(\mathsf{not}(x)), x)$$
$$\mathsf{not\_not} = \lambda(x : \mathsf{Bool}).\, \mathsf{If}(y.\, \mathsf{Id}_{\mathsf{Bool}}(\mathsf{not}(\mathsf{not}(y)), y); \mathsf{refl}(\mathsf{true}); \mathsf{refl}(\mathsf{false}); x)$$

$$(\mathsf{Id}_{\mathsf{Bool}}(\mathsf{not}(\mathsf{not}(y)), y))[\mathsf{true} \,/\, y] = \mathsf{Id}_{\mathsf{Bool}}(\mathsf{true}, \mathsf{true})$$

Definitional Equality

α	renaming	$\lambda x.x = \lambda y.y$
β	computation	$(\lambda x.N)(M) = N[M/x]$
δ	unfolding definitions	$\mathsf{const}(x) = (\lambda y\, z.y)(x)$
η	extensionality	$f = \lambda x.f(x)$

- Formally: need definitionally equal terms to be identical
- Implementations use $\beta\delta$-normal η-long representatives

Formal presentation

Contexts	Γ, Δ
Substitutions	$\Gamma \vdash \gamma : \Delta$
Types	$\Gamma \vdash A \,\mathsf{type}$
Terms	$\Gamma \vdash M : A$

$$\textsc{Substitution Action}\quad \frac{\Gamma \vdash \gamma : \Delta \qquad \Delta \vdash M : A}{\Gamma \vdash M[\gamma] : A[\gamma]}$$

$$\textsc{Conversion}\quad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B \,\mathsf{type}}{\Gamma \vdash M : B}$$

Formal presentation

Dependent product

$$\textsc{Type Former}\quad \frac{\Gamma \vdash A\,\mathsf{type} \qquad \Gamma.A \vdash B\,\mathsf{type}}{\Gamma \vdash \prod(A,B)\,\mathsf{type}} \qquad \textsc{Intro}\quad \frac{\Gamma.A \vdash N : B}{\Gamma \vdash \lambda_A N : \prod(A,B)}$$

$$\textsc{Elim}\quad \frac{\Gamma \vdash N : \prod(A,B) \qquad \Gamma \vdash M : A}{\Gamma \vdash N(M) : B[1.M]}$$

$$\textsc{Beta}\quad \frac{\Gamma.A \vdash N : B \qquad \Gamma \vdash M : A}{\Gamma \vdash (\lambda_A N)M = N[1.M] : B[1.M]}$$

$$\textsc{Eta}\quad \frac{\Gamma \vdash M : \prod(A,B)}{\Gamma \vdash M = \lambda_A((M[\mathsf{p}])(\mathsf{v}_0)) : \prod(A,B)}$$

Less formal syntax

- We write informal notation, computer fill in details
- Named variables
- Fewer type annotations
- Type checker fills in ‘obvious’ terms (holes)

$$\begin{aligned} &\mathsf{and} \;:\; \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} \\ &\mathsf{and} = \lambda x\, y.\, \mathsf{If}(z.\ \_; y; \mathsf{false}; x) \end{aligned}$$

What is a type checker?

- Input: human readable terms *(preterms)*
- Output: fully explicit terms or a **helpful** error
- Bonus: help write proofs

Preterms

Variables: x, y, z

Preterms: $M, N ::= x$
$\quad | \ (x : M) \to N \mid \lambda(x : M).N \mid \lambda x.N \mid M(N)$
$\quad | \ \sum(x : M)N \mid (M, N) \mid \mathsf{fst}(M) \mid \mathsf{snd}(M)$
$\quad | \ \ldots$

$\mathsf{swap} : (M : \mathsf{U}) \to (N : \mathsf{U}) \to \Sigma(x : M)N \to \Sigma(x : N)M$

$\mathsf{swap} = \lambda M\ N\ p.(\mathsf{snd}(p), \mathsf{fst}(p))$

What makes type checking hard?

$$\mathsf{foo} : \mathsf{Id}(N, M)$$
$$\mathsf{foo} = \mathsf{refl}(N)$$

- foo is allowed iff N and M are definitionally equal
- Type checking requires *equality* checking

What makes type checking hard?

- Type checker must contain an interpreter
- Don't evaluate the entire program
- Keeping track of contexts

Bidirectional type checking

Example

$$\text{Check} \qquad x : \mathsf{Bool} \vdash \mathsf{id}(\mathsf{Bool})(x) : \mathsf{Bool}$$

- Infer id has type $(A : \mathsf{U}) \to A \to A$
- Check Bool has type U
- Infer x has type Bool - check inferred type is Bool
- Infer id(Bool)(x) has type Bool - check inferred type is Bool

Bidirectional type checking

Example

$$\text{Check} \qquad x : \text{Bool} \vdash \text{id}(\text{Bool})(x) : \text{Bool}$$

- Infer id has type $(A : \text{U}) \to A \to A$
- Check Bool has type U
- Infer x has type Bool - check inferred type is Bool
- Infer id(Bool)(x) has type Bool - check inferred type is Bool

Bidirectional type checking

Example

$$\text{Check} \qquad x : \mathsf{Bool} \vdash \mathsf{id}(\mathsf{Bool})(x) : \mathsf{Bool}$$

- Infer id has type $(A : \mathsf{U}) \to A \to A$
- Check Bool has type U
- Infer x has type Bool - check inferred type is Bool
- Infer id(Bool)(x) has type Bool - check inferred type is Bool

Bidirectional type checking

Example

$$\text{Check} \qquad x : \text{Bool} \vdash \text{id}(\text{Bool})(x) : \text{Bool}$$

- Infer id has type $(A : \text{U}) \to A \to A$
- Check Bool has type U
- Infer x has type Bool - check inferred type is Bool
- Infer $\text{id}(\text{Bool})(x)$ has type Bool - check inferred type is Bool

Bidirectional type checking

Example

Check $x : \mathsf{Bool} \vdash \mathsf{id}(\mathsf{Bool})(x) : \mathsf{Bool}$

- Infer id has type $(A : \mathsf{U}) \to A \to A$
- Check Bool has type U
- Infer x has type Bool - check inferred type is Bool
- Infer $\mathsf{id}(\mathsf{Bool})(x)$ has type Bool - check inferred type is Bool

Bidirectional type checking

Example

Check $\quad x : \mathsf{Bool} \vdash \mathsf{id}(\mathsf{Bool})(x) : \mathsf{Bool}$

- Infer id has type $(A : \mathsf{U}) \to A \to A$
- Check Bool has type U
- Infer x has type Bool - check inferred type is Bool
- Infer id(Bool)(x) has type Bool - check inferred type is Bool

Bidirectional type checking

Example

$$\text{Check} \qquad x : \mathsf{Bool} \vdash \mathsf{id}(\mathsf{Bool})(x) : \mathsf{Bool}$$

- Infer id has type $(A : \mathsf{U}) \to A \to A$
- Check Bool has type U
- Infer x has type Bool - check inferred type is Bool
- Infer $\mathsf{id}(\mathsf{Bool})(x)$ has type Bool - check inferred type is Bool

How type check?

- Bidirectional type checking
- Normalisation by Evaluation
- Metavariables and elaboration

Bidirectional type checking

Type checking $\quad \Gamma \vdash M \Leftarrow A \rightsquigarrow N$

Type inference $\quad \Gamma \vdash M \Rightarrow A \rightsquigarrow N$

Bidirectional type checking

$$\frac{\Gamma, x : A \vdash M \Leftarrow B \rightsquigarrow N}{\Gamma \vdash \lambda x.M \Leftarrow \prod(A, B) \rightsquigarrow \lambda_A N}$$

$$\frac{\Gamma \vdash M \Rightarrow \prod(A, B) \rightsquigarrow M' \qquad \Gamma \vdash N \Leftarrow A \rightsquigarrow N'}{\Gamma \vdash M(N) \Rightarrow B[1.N'] \rightsquigarrow M'(N')}$$

$$\frac{x \notin \mathsf{Subjects}(\Gamma_1)}{\Gamma_0, x : A, \Gamma_1 \vdash x \Rightarrow A \rightsquigarrow \mathsf{v}_{|\Gamma_1|}}$$

Bidirectional type checking

$$\frac{\Gamma, x : A \vdash M \Leftarrow B \rightsquigarrow N}{\Gamma \vdash \lambda x.M \Leftarrow \prod(A, B) \rightsquigarrow \lambda_A N}$$

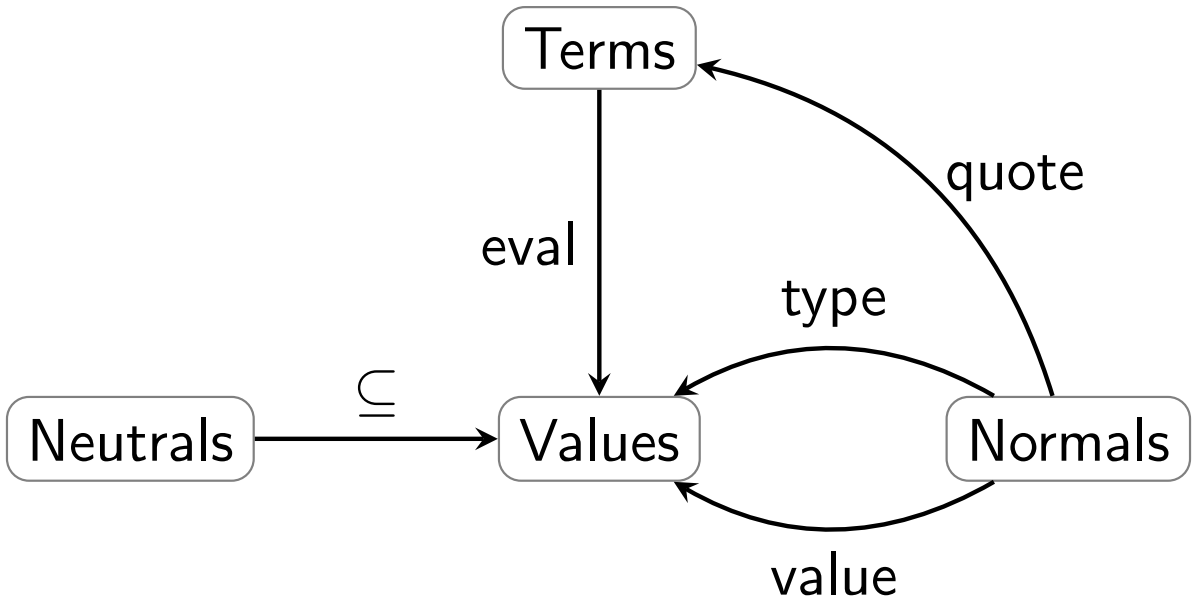
$$\frac{\Gamma \vdash M \Rightarrow \prod(A, B) \rightsquigarrow M' \qquad \Gamma \vdash N \Leftarrow A \rightsquigarrow N'}{\Gamma \vdash M(N) \Rightarrow B[1.N'] \rightsquigarrow M'(N')}$$

$$\frac{x \notin \mathsf{Subjects}(\Gamma_1)}{\Gamma_0, x : A, \Gamma_1 \vdash x \Rightarrow A \rightsquigarrow \mathsf{v}_{|\Gamma_1|}}$$

$$\frac{\Gamma \vdash M \Rightarrow A \rightsquigarrow M' \qquad \Gamma \vdash A \stackrel{?}{=} B\,\mathsf{type}}{\Gamma \vdash M \Leftarrow B \rightsquigarrow M'}$$

Normalisation by Evaluation

- eval produces $\beta\delta$-normal forms
- Type directed quoting gives η-long forms



Equality checking

- $\mathsf{norm}(M) = \mathsf{quote}(\mathsf{eval}(M))$
- M and N are definitionally equal iff $\mathsf{norm}(M)$ and $\mathsf{norm}(N)$ are structurally equal

$$\frac{\mathsf{norm}(M) = \mathsf{norm}(N)}{\Gamma \vdash M \stackrel{?}{=} N : A}$$

Elaboration

- This still sucks :(

$$\mathsf{map}(\mathsf{Bool} \to \mathsf{Bool})(\mathsf{compose}(\mathsf{Bool})(\mathsf{Bool})(\mathsf{Bool})(\mathsf{not}))$$
$$: \mathsf{List}(\mathsf{Bool} \to \mathsf{Bool}) \to \mathsf{List}(\mathsf{Bool} \to \mathsf{Bool})$$

Elaboration

- Add *holes* (_) to preterms and *metavariables* to terms
- Elaboration fills in holes
- Metacontext (Θ) — telescope of metavariables with:
  - Context Γ
  - Type $\Gamma \vdash A$ type
  - Optional solution $\Gamma \vdash N : A$

$$\textsc{Meta/Intro}\quad \frac{(\Delta \vdash \alpha : A) \in \Theta \qquad \gamma : \Gamma \to \Delta}{\Theta | \Gamma \vdash \alpha[\gamma] : A[\gamma]}$$

Tracking metacontexts

- Formally with metasubstitutions
- Informally with mutable global metacontext
  - freshMeta(Γ, A) — create a fresh metavariable
  - solveMeta(α, M) — solve α as M

Unification

- Find metasubstitution θ to unify M and N (ie $M[\theta] = N[\theta]$)
- Informally: solve metavariables to make $M = N$
- Subsumes equality checking

Solving metavariables

- Want most general unifier
- Undecidable in general
- *Pattern fragment* is decidable

$$\Gamma \vdash \alpha[x_1, \ldots, x_n] \stackrel{?}{=} N : A$$

1. $x_1, \ldots, x_n$ are distinct bound variables
2. Free variables of N are in the $\{x_1, \ldots, x_n\}$
3. α doesn't occur in N

$$\alpha := N[\mathsf{v}_0 \mapsto x_n, \ldots, \mathsf{v}_{n-1} \mapsto x_1]$$

Elaborating holes

$$\frac{\alpha := \mathsf{freshMeta}(\Gamma, A)}{\Gamma \vdash \_ \Leftarrow A \rightsquigarrow \alpha}$$

$$\frac{\alpha := \mathsf{freshMeta}(\Gamma, \mathsf{U}) \qquad \beta := \mathsf{freshMeta}(\Gamma, \alpha)}{\Gamma \vdash \_ \Rightarrow \alpha \rightsquigarrow \beta}$$

$$\frac{\begin{array}{c}\alpha[x_1, \ldots, x_n] \stackrel{?}{=} N \text{ satisfies pattern conditions} \\ \mathsf{solveMeta}(\alpha, N[\mathsf{v}_0 \mapsto x_n, \ldots, \mathsf{v}_{n-1} \mapsto x_1])\end{array}}{\Gamma \vdash \alpha[x_1, \ldots, x_n] \stackrel{?}{=} N : A}$$

Elaborating holes

Example

$$\cdot \vdash \mathsf{id}(\_)(\mathsf{true}) \Rightarrow \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$$

- Infer $\mathsf{id} : (A : \mathsf{U}) \to A \to A$
- Check $\_ : \mathsf{U} \rightsquigarrow \alpha[1]$
- Infer $\mathsf{true} : \mathsf{Bool}$
- Unify $\alpha[1]$ and Bool — solve $\alpha = \mathsf{Bool}$
- Infer $\mathsf{id}(\_)(\mathsf{true}) : \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$

Elaborating holes

Example

$$\cdot \vdash \mathsf{id}(\_)(\mathsf{true}) \Rightarrow \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$$

- Infer $\mathsf{id} : (A : \mathsf{U}) \to A \to A$
- Check $\_ : \mathsf{U} \rightsquigarrow \alpha[1]$
- Infer $\mathsf{true} : \mathsf{Bool}$
- Unify $\alpha[1]$ and Bool — solve $\alpha = \mathsf{Bool}$
- Infer $\mathsf{id}(\_)(\mathsf{true}) : \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$

Elaborating holes

Example

$$\cdot \vdash \mathsf{id}(\_)(\mathsf{true}) \Rightarrow \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$$

- Infer $\mathsf{id} : (A : \mathsf{U}) \to A \to A$
- Check $\_ : \mathsf{U} \rightsquigarrow \alpha[1]$
- Infer $\mathsf{true} : \mathsf{Bool}$
- Unify $\alpha[1]$ and Bool — solve $\alpha = \mathsf{Bool}$
- Infer $\mathsf{id}(\_)(\mathsf{true}) : \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$

Elaborating holes

Example

$$\cdot \vdash \mathsf{id}(\_)(\mathsf{true}) \Rightarrow \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$$

- Infer $\mathsf{id} : (A : \mathsf{U}) \to A \to A$
- Check $\_ : \mathsf{U} \rightsquigarrow \alpha[1]$
- Infer $\mathsf{true} : \mathsf{Bool}$
- Unify $\alpha[1]$ and Bool — solve $\alpha = \mathsf{Bool}$
- Infer $\mathsf{id}(\_)(\mathsf{true}) : \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$

Elaborating holes

Example

$$\cdot \vdash \mathsf{id}(\_)(\mathsf{true}) \Rightarrow \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$$

- Infer $\mathsf{id} : (A : \mathsf{U}) \to A \to A$
- Check $\_ : \mathsf{U} \rightsquigarrow \alpha[1]$
- Infer $\mathsf{true} : \mathsf{Bool}$
- Unify $\alpha[1]$ and Bool — solve $\alpha = \mathsf{Bool}$
- Infer $\mathsf{id}(\_)(\mathsf{true}) : \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$

Elaborating holes

Example

$$\cdot \vdash \mathsf{id}(\_)(\mathsf{true}) \Rightarrow \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$$

- Infer $\mathsf{id} : (A : \mathsf{U}) \to A \to A$
- Check $\_ : \mathsf{U} \rightsquigarrow \alpha[1]$
- Infer $\mathsf{true} : \mathsf{Bool}$
- Unify $\alpha[1]$ and Bool — solve $\alpha = \mathsf{Bool}$
- Infer $\mathsf{id}(\_)(\mathsf{true}) : \mathsf{Bool} \rightsquigarrow \mathsf{id}(\mathsf{Bool})(\mathsf{true})$

Further improvements

- Implicit arguments

$$\begin{array}{l}\textbf{let}\ \mathsf{id} : \{A : \_\} \to A \to A := \lambda x.x \\ \textbf{in}\ \mathsf{id}(\mathsf{true})\end{array}$$

- Pruning

$$\alpha[x, x, y] \stackrel{?}{=} y \quad \Longrightarrow \quad \begin{cases} \alpha[w, x, y] := \beta[y] \\ \beta[y] \stackrel{?}{=} y \end{cases}$$

Conclusion

- Type checking fills in type annotations
- Elaboration fills in holes/implicit arguments
- NbE reduces definitional equality to structural equality
- Unification solves metavariables to fill in holes

Further reading

- András Kovács' *Elaboration Zoo*

References

- Theirry Coquand *An Algorithm for Type-Checking Dependent Types, 1996*
- Dale Miller *A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification, 1991*