Type Checking for Propositional Type Theory in Linear Time.

Daniël Otten

Introduction.

What is Propositional Type Theory?

Equality

Intensional type theory (ITT) has two notions of equality:

propositional ($=$)	internal	proofs	undecidable
definitional ($\equiv$)	external	reductions	decidable

So, definitional eq forms a decidable fragment of propositional eq.
This is needed to make type checking decidable.

Why this particular decidable fragment?

There are some trade-offs for the amount of definitional eq:

- more makes it easier to work inside the system,
- less makes is easier to find models for the system.

Extremes

We consider the two extremes:

- Propositional type theory (PTT), without any definitional eq.
- Extensional type theory (ETT), where every eq is definitional.

These form the min and max of a poset with ITT in the middle:

$$\left(\begin{tikzcd} \mathsf{ETT} \\ \mathsf{ITT} \arrow[u, hook] \\ \mathsf{PTT} \arrow[u, hook] \end{tikzcd}\right)$$

Rules for =-types in ITT

Formation and Introduction:

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash a =_A a' : \mathsf{Type} \qquad \Gamma \vdash \mathsf{refl}_a : a =_A a.}$$

Elimination and Computation:

$$\frac{\Gamma, x : A, x' : A, w : x =_A x' \vdash C[x, x', w] : \mathsf{Type} \qquad \Gamma, x : A \vdash c[x] : C[x, x, \mathsf{refl}_x] \qquad \Gamma \vdash p : a =_A a'}{\Gamma \vdash \mathsf{ind}^{=}_{A, C[x,x',w], c[x], a, a', p} : C[a, a', p] \qquad \Gamma \vdash \mathsf{ind}^{=}_{A, C[x,x',w], c[x], a, a, \mathsf{refl}_a} \equiv c[x].}$$

Rules for =-types in PTT

Formation and Introduction:

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash a =_A a' : \mathsf{Type} \qquad \Gamma \vdash \mathsf{refl}_a : a =_A a.}$$

Elimination and Computation:

$$\frac{\Gamma, x : A, x' : A, w : x =_A x' \vdash C[x, x', w] : \mathsf{Type} \qquad \Gamma, x : A \vdash c[x] : C[x, x, \mathsf{refl}_x] \qquad \Gamma \vdash p : a =_A a'}{\Gamma \vdash \mathsf{ind}^{=}_{A, C[x,x',w], c[x], a, a', p} : C[a, a', p] \qquad \Gamma \vdash \beta^{=}_{A, C[x,x',w], c[x], a} : \mathsf{ind}^{=}_{A, C[x,x',w], c[x], a, a, \mathsf{refl}_a} = c[x].}$$

Comparisons

The amount of definitional eq has little influence on the provability.
Théo Winterhalter showed that ETT is conservative over PTT plus:

- binder extensionality (bindext),
  (more general than function extensionality, equivalent in ITT);
- uniqueness of identity proofs (uip).

However, definitional eq has a large influence on type checking:

- ETT: undecidable;
- ITT: decidable but not in elementary time, so not in $\mathcal{O}(2^n) \cup \mathcal{O}(2^{2^n}) \cup \mathcal{O}(2^{2^{2^n}}) \cup \cdots$;
- PTT: decidable in linear time (as we will show here).

This builds on work of Benno van den Berg and Martijn den Besten who showed that type checking for PTT is quadratic time.

Type Checking in Linear Time.

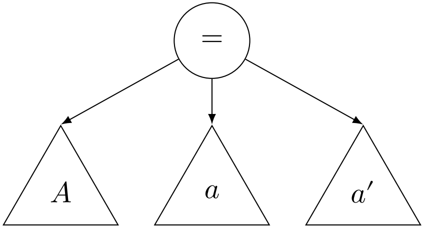
Theorem.

We can decide whether PTT derives $\Gamma \vdash a : A$ in linear time.

Representation

The input is represented as a syntax tree using De Bruijn indices.

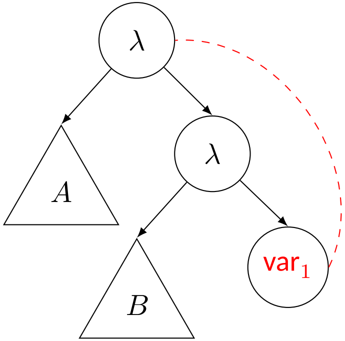
For example, $a =_A a'$ is:

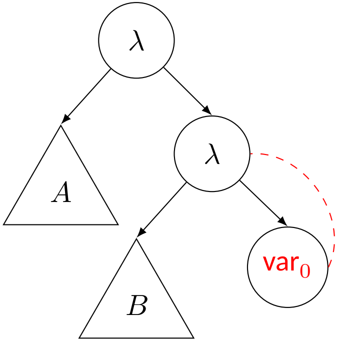


Representation

The input is represented as a syntax tree using De Bruijn indices.

For example, $\lambda x : A.\, \lambda y : B.\, x$ and $\lambda x : A.\, \lambda y : B.\, y$ are:





Checking the Context Efficiently

To derive $\Gamma \vdash a : A$ we check that Γ is well-formed multiple times.

So, for efficiency, we solve the following problem instead:

$\Gamma \vDash^{*} a : A$, where we decide $\Gamma \vdash a : A$ assuming $\Gamma \vdash A$: Type.

Lemma.

If $\Gamma \vDash^{*} a : A$ is $\mathcal{O}(|a| + |A|)$, then $\Gamma \vdash a : A$ is linear time.

Proof. We can decide $\Gamma \vdash a : A$ as follows:

- if $\Gamma = (x_0 : A_0, \dots, x_{n-1} : A_{n-1})$, then for every $i < n$:
  check $x_0 : A_0, \dots, x_{i-1} : A_{i-1} \vDash^{*} A_i$: Type in $\mathcal{O}(|A_i|)$;
- check $\Gamma \vDash^{*} A$: Type in $\mathcal{O}(|A|)$;
- check $\Gamma \vDash^{*} a : A$ in $\mathcal{O}(|a| + |A|)$. $\square$

Problems

The standard algorithm checks $\Gamma \overset{*}{\vdash} a : A$ with recursion on a.
Such an algorithm is not efficient enough.

We cannot use induction on a to show $\Gamma \overset{*}{\vdash} a : A$ is $\mathcal{O}(|a| + |A|)$:

- Problematic Case: deciding $\Gamma \overset{*}{\vdash} c =_C c' :$ Type.
  We need to check $\Gamma \overset{*}{\vdash} C :$ Type, $\Gamma \overset{*}{\vdash} c : C$, and $\Gamma \overset{*}{\vdash} c' : C$.
  C appears once in the input but we recursively use it trice.

Similarly, we cannot use induction on a to show $\Gamma \vdash a : A$ is $\mathcal{O}(|a|)$:

- Problematic Case: deciding $\Gamma \overset{*}{\vdash} x_i : A$.
  We need to check that x_i is in Γ and that the type of x_i is A.
  This cannot be done in constant time.

The last natural option, showing $\Gamma \vdash a : A$ is $\mathcal{O}(|A|)$-time, is ruled out in the same way as $\mathcal{O}(|a| + |A|)$-time.

Solution: Dividing the Work

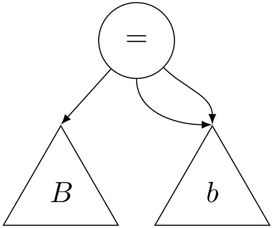
Instead, we will decide $\Gamma \overset{*}{\vdash} a : A$ in two steps:

- An $\mathcal{O}(|a|)$ algorithm inspired by type inference.

  We generate a type A' such that the following holds:

  $A \equiv A'$ iff $\Gamma \overset{*}{\vdash} a : A$.

  We may represent A' using sharing. For example, $b =_B b$ can be:



- An $\mathcal{O}(|A|)$ syntactic check. We check $A \equiv A'$.

  Using an algorithm of Condoluci, Accattoli, and Sacerdoti Coen.

The Algorithm.

We generate A' for $\Gamma \overset{*}{\vdash} a : A$ with recursion on a:

- For $\Gamma \vdash x_i : A$, we output $A' :\equiv A_i$. Because we allow sharing, we just save the root node instead of copying the syntax tree. Hence, this can indeed be done in $\mathcal{O}(1)$-time.
- For $\Gamma \vdash b =_B b' : A$, we first recursively run the algorithm on $\Gamma \overset{*}{\vdash} b : B$ and $\Gamma \overset{*}{\vdash} b' : B$ to get B' and B'' in $\mathcal{O}(|b| + |b'|)$-time. We check that $B \equiv B' \equiv B''$ in $\mathcal{O}(|B|)$-time.
  After this, we simply output $A' :\equiv$ Type.
- For $\Gamma \vdash \text{refl}_b : A$, we check that we have $A \equiv (b' =_B b'')$ for some b', b'', B. We check $\Gamma \overset{*}{\vdash} b : B$ in $\mathcal{O}(|b| + |B|)$-time and output $A' := (b =_B b)$ (using sharing).

The Algorithm.

We generate A' for $\Gamma \overset{*}{\vdash} a : A$ with recursion on a:

- For $\mathsf{ind}^{=}_{B,C[x,x',w],c[x],b,b',p} : A$, we use a similar approach to check that the subterms have the correct types in linear time. We output $A' := C[b,b',p]$, which we can construct by copying $C[x,x',w]$, traversing the tree, and replacing the leaves corresponding to x, x', w with b, b', p in $\mathcal{O}(C[x,x',w])$-time.
- For $\beta^{=}_{B,C[x,x',w],c[x],b} : A$, we check the subterms and output $A' := (\mathsf{ind}^{=}_{B,C[x,x',w],c[x],b,b,\mathsf{refl}_b} =_{C[b,b,\mathsf{refl}_b]} c[b])$.

The Algorithm.

We have covered the cases for $=$-types.

These cases contain the main ideas needed for this algorithm:

- using sharing so that we don't have to copy as often;
- using the fact that the recursive calls only depend on the term.

The cases for $\mathbb{0}, \mathbb{1}, \ldots, \mathbb{N}, \Sigma, \Pi$-types are similar.

This concludes the proof: type checking is linear time.

Conclusion,

and future work.

Future Work: Bidirectional Type Checking.

Currently, the algorithm is geared for unidirectional type checking.

So, we must be able to infer the types of subterms directly.

Therefore, we require application to be annotated:

$$\frac{\Gamma \vdash f : \Pi(x : A)\, B[x] \qquad \Gamma \vdash a : A}{\Gamma \vdash f \mathbin{@}_{A,B[x]} a : B[a]}$$

Bidirectional type checking does not need these annotations.

It combines type checking with type inference:

$$\frac{\Gamma \vdash f \Rightarrow \Pi(x : A)\, B[x] \qquad \Gamma \vdash a \Leftarrow A}{\Gamma \vdash f\, a \Rightarrow B[a]}$$

Our algorithm is inspired by type inference.

This might help to extend it to bidirectional type checking.

Conclusion.

For other terms, the algorithm seems robust and works for any amount of annotations: it works both for $\lambda x.\, b[x]$ and $\lambda x : A.\, b[x]$, and similarly for it works for each of refl$_{A,a}$, refl$_A$, refl.

This also suggests that it might be possible to extend it to bidirectional type checking.

References:

- Benno van den Berg, Martijn den Besten (2021):
  Quadratic type checking for objective type theory.
- Andrea Condoluci, Beniamino Accattoli, Claudio Sacerdoti Coen (2019): Sharing Equality is Linear.
- Théo Winterhalter, Matthieu Sozeau, Nicolas Tabareau (2019): Eliminating Reflection from Type Theory.