A modular formalization of type theory in Coq

Andrej Bauer (University of Ljubljana)
Philipp Haselwarter (University of Ljubljana)
Théo Winterhalter (ENS Cachan)

TYPES 2017 – Budapest – Hungary

We are reporting on a library of formalized type theory in Coq. This is joint work with Philipp Haselwarter and Théo Winterhalter. Allow me to tell you what we formalized first, and we can discuss alternatives later.

Context	Γ context	$\Gamma \equiv \Delta$
Substitution	$\sigma : \Gamma \to \Delta$	$\sigma \equiv \theta : \Gamma \to \Delta$
Type	$\Gamma \vdash A$ type	$\Gamma \vdash \Gamma \equiv \Delta$
Term	$\Gamma \vdash u : A$	$\Gamma \vdash u \equiv v : A$

- We formalized Martin-Löf type theory with contexts, explicit substitutions, types, and terms.
- Each of these has its own pre-syntax and two judgment forms, expressing well-formedness and equality.
- The calculus is variable free, with binding a la de Bruijn.

Context	Γ context	$\Gamma \equiv \Delta$
Substitution	$\sigma : \Gamma \to \Delta$	$\sigma \equiv \theta : \Gamma \to \Delta$
Type	$\Gamma \vdash A$ type	$\Gamma \vdash \Gamma \equiv \Delta$
Term	$\Gamma \vdash u : A$	$\Gamma \vdash u \equiv v : A$

(variable free pre-syntax)

- We formalized Martin-Löf type theory with contexts, explicit substitutions, types, and terms.
- Each of these has its own pre-syntax and two judgment forms, expressing well-formedness and equality.
- The calculus is variable free, with binding a la de Bruijn.

Types:

$\prod$ `Id` × `Univ`$_i$ `Empty Unit Bool`

Configurable principles:

- equality reflection
- J-rule for `Id`
- η-rule for $\prod$
- impredicative universe `Prop`

- The formalization has products, identity types, a hierarchy of universes, and base types.
- Apart from the standard rules there are also some principles that we might no want in every situation: equality reflection, J-rule, η-rule for products, and an impredicative universe of propositions.
- So we made the principles *configurable*: depending on what you're doing, you may turn each one on or off, or be ambivalent about it.
- In fact, the types are also configurable. (We could have made products and identity types configurable as well.)
- It is fairly easy to add more types and principles, for example one would like to add Σ and inductive types.

Types:

$\prod$ `Id` `×` `Univ`$_i$ `Empty Unit Bool`

configurable

Configurable principles:

- equality reflection
- J-rule for `Id`
- η-rule for $\prod$
- impredicative universe `Prop`

- The formalization has products, identity types, a hierarchy of universes, and base types.
- Apart from the standard rules there are also some principles that we might no want in every situation: equality reflection, J-rule, η-rule for products, and an impredicative universe of propositions.
- So we made the principles *configurable*: depending on what you're doing, you may turn each one on or off, or be ambivalent about it.
- In fact, the types are also configurable. (We could have made products and identity types configurable as well.)
- It is fairly easy to add more types and principles, for example one would like to add Σ and inductive types.

```
Inductive context      : Type := …
with      type         : Type := …
with      term         : Type := …
with      substitution : Type := …
```

- The embedding into Coq is “deep”.
- Pre-syntax is defined by four mutually recursive inductive types.
- The pre-syntax is *not* configurable. Even if you turn off universes, for example, the presyntax for them still exists. But you will never see it in a proof because tactics will dispose of the cases involving features that are turned off.

```
isctx   : context → Type
issubst : substitution → context → context → Type
istype  : context → type → Type
isterm  : context → term → type → Type
eqctx   : context → context → Type
eqsubst : substitution → substitution →
                         context → context → Type
eqtype  : context → type → type → Type
eqterm  : context → term → term → type → Type
```

- The rules for judgment forms are given by eight mutually inductive types.
- There are 154 clauses, which means that type theory has these many rules. We also wrote a little script that converts them to LaTeX. It is a *complete* list of rules of type theory – no shortcuts.
- These judgments are configurable through a type class mechanism.

```
| TermProj1 :
  simpleproduct rule
    parameters: {Γ A B p},
    precond: isctx Γ
    precond: istype Γ A
    precond: istype Γ B
    premise: isterm Γ p (SimProd A B)
    conclusion:
      isterm Γ (proj1 A B p) A
  endrule
```

- Here is what a rule looks like. It is an elimination rule for simple products.
- We are using custom notation to make it readable.
- The notation unfolds to the following standard syntax.

```
| TermProj1 :
  ∏ {_ : simpleproductsFlag},
  ∏ {Γ A B p},
    (precondFlag → isctx Γ) →
    (precondFlag → istype Γ A) →
    (precondFlag → istype Γ B) →
    isterm Γ p (SimProd A B) →
    isterm Γ (proj1 A B p) A
```

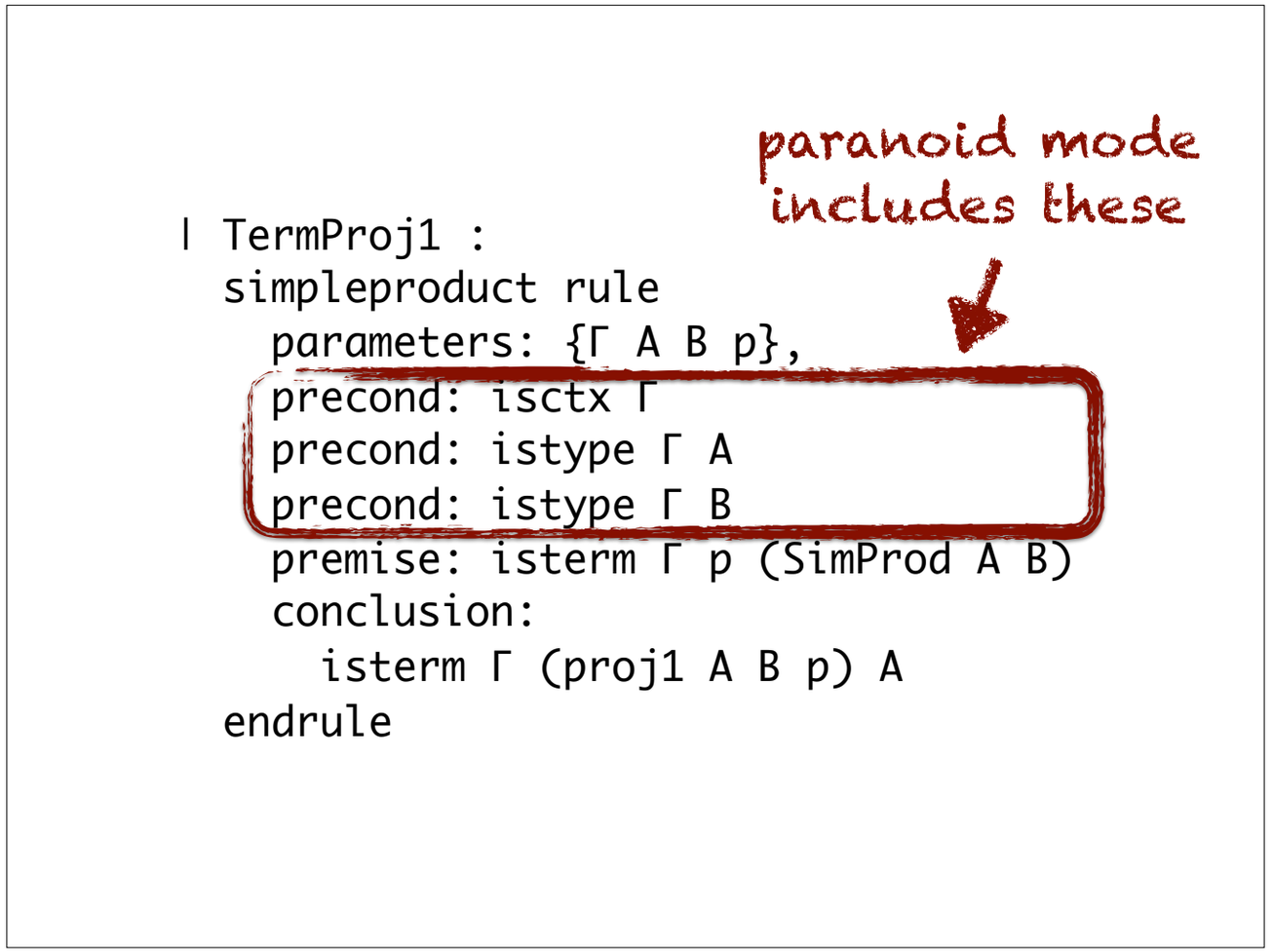
- This is more familiar, but having the other notation is quite helpful.
- You see that the entire rule is prefixed with a configuration flag, and so are the preconditions.
- Through the type class mechanism Coq will find these flag to be either “off” or “on” or “ambivalent”.
- Let us go back to the humane notation.

```
| TermProj1 :
  simpleproduct rule
    parameters: {Γ A B p},
    precond: isctx Γ
    precond: istype Γ A
    precond: istype Γ B
    premise: isterm Γ p (SimProd A B)
    conclusion:
      isterm Γ (proj1 A B p) A
  endrule
```

You will notice that we distinguish between *preconditions* and *premises*.
There are two ways to formulate such a rule:

1. In the *paranoid* mode we include both the preconditions and the premises.
2. In the *economic* mode we include only the premises.

This is a configurable option in the library. One may need either the paranoid or the economic version of type theory, depending on what one is doing. (And we proved meta-theorems that allows us to transfer between them.)



You will notice that we distinguish between *preconditions* and *premises*.
There are two ways to formulate such a rule:

1. In the *paranoid* mode we include both the preconditions and the premises.
2. In the *economic* mode we include only the premises.

This is a configurable option in the library. One may need either the paranoid or the economic version of type theory, depending on what one is doing. (And we proved meta-theorems that allows us to transfer between them.)

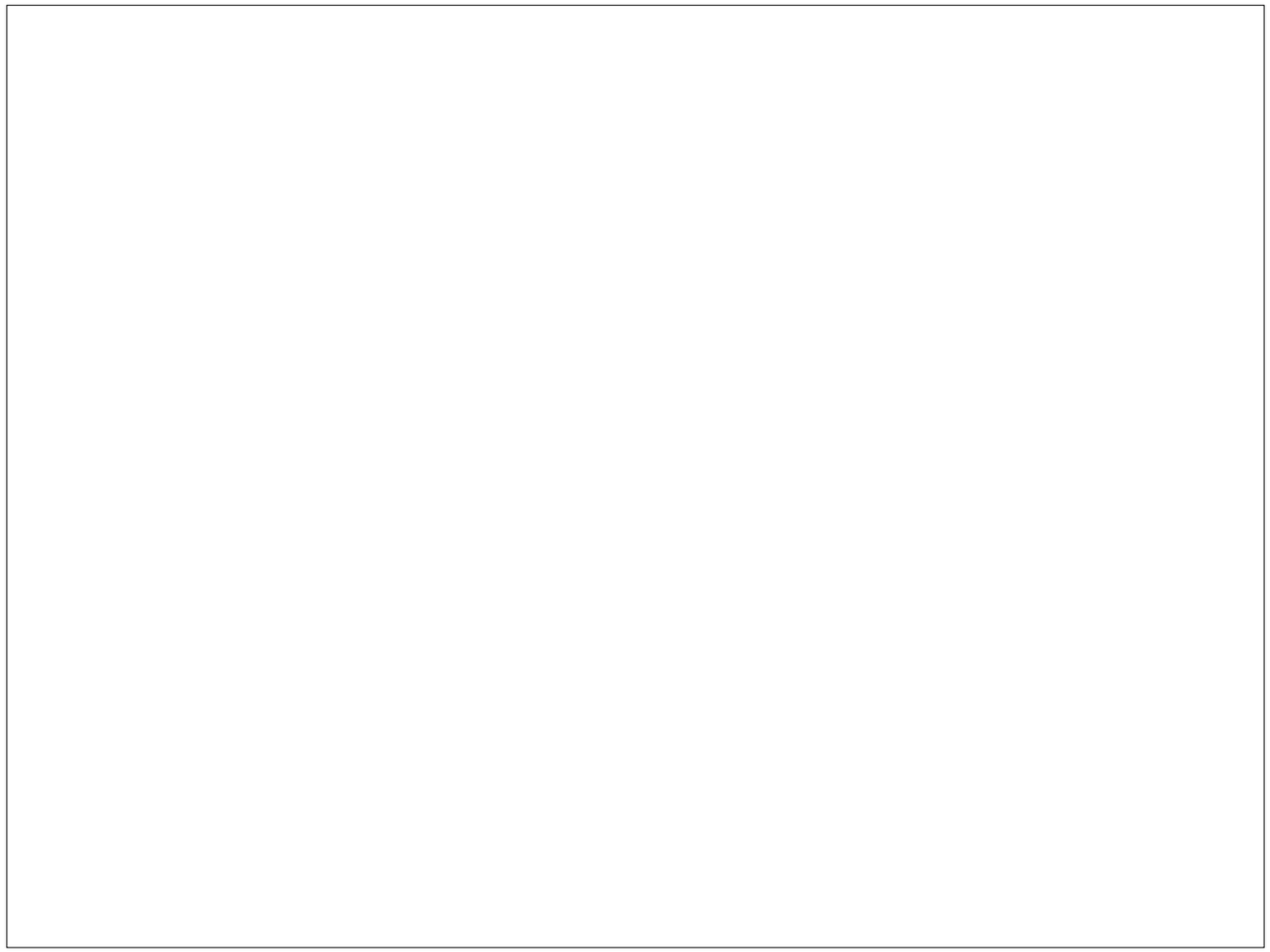
```
| TermProj1 :
  simpleproduct rule
    parameters: {Γ A B p},
    precond: isctx Γ
    precond: istype Γ A
    precond: istype Γ B
    premise: isterm Γ p (SimProd A B)
    conclusion:
      isterm Γ (proj1 A B p) A
  endrule
```

economic mode:
this is enough

You will notice that we distinguish between *preconditions* and *premises*.
There are two ways to formulate such a rule:

1. In the *paranoid* mode we include both the preconditions and the premises.
2. In the *economic* mode we include only the premises.

This is a configurable option in the library. One may need either the paranoid or the economic version of type theory, depending on what one is doing. (And we proved meta-theorems that allows us to transfer between them.)



With more than a 100 rules it is in fact easy to get them wrong (we did). So we proved several meta-theorems which guarantee that the rules fit together and nothing is missing. These are:

1. A useful theorem says that the economic and paranoid rules prove the same judgments. We may thus switch between them as desired.
2. Sanity theorems say things like: if you can show that u is a term of type A then A is a type.
3. Uniqueness of typing, which says that a term has at most one type.
4. There are also several useful (and sometimes tricky) lemmas about inversion and admissibility.

One that is missing but we would like to have is that explicit substitutions can always be computed away. We are open to ideas about how to prove this in the best way.

Paranoia is needless

$\Gamma \vdash_{\text{paranoid}} \mathcal{J}$ if and only if $\Gamma \vdash_{\text{economic}} \mathcal{J}$.

With more than a 100 rules it is in fact easy to get them wrong (we did). So we proved several meta-theorems which guarantee that the rules fit together and nothing is missing. These are:

1. A useful theorem says that the economic and paranoid rules prove the same judgments. We may thus switch between them as desired.
2. Sanity theorems say things like: if you can show that u is a term of type A then A is a type.
3. Uniqueness of typing, which says that a term has at most one type.
4. There are also several useful (and sometimes tricky) lemmas about inversion and admissibility.

One that is missing but we would like to have is that explicit substitutions can always be computed away. We are open to ideas about how to prove this in the best way.

Paranoia is needless

$\Gamma \vdash_{\text{paranoid}} \mathcal{J}$ if and only if $\Gamma \vdash_{\text{economic}} \mathcal{J}$.

Sanity theorem

If $\Gamma \vdash \mathtt{u} : \mathtt{A}$ then Γ `context` and $\Gamma \vdash \mathtt{A}$ `type`.

With more than a 100 rules it is in fact easy to get them wrong (we did). So we proved several meta-theorems which guarantee that the rules fit together and nothing is missing. These are:

1. A useful theorem says that the economic and paranoid rules prove the same judgments. We may thus switch between them as desired.
2. Sanity theorems say things like: if you can show that u is a term of type A then A is a type.
3. Uniqueness of typing, which says that a term has at most one type.
4. There are also several useful (and sometimes tricky) lemmas about inversion and admissibility.

One that is missing but we would like to have is that explicit substitutions can always be computed away. We are open to ideas about how to prove this in the best way.

Paranoia is needless

$\Gamma \vdash_{\text{paranoid}} \mathcal{J}$ if and only if $\Gamma \vdash_{\text{economic}} \mathcal{J}$.

Sanity theorem

If $\Gamma \vdash \mathtt{u} : \mathtt{A}$ then Γ `context` and $\Gamma \vdash \mathtt{A}$ `type`.

Uniqueness of typing

If $\Gamma \vdash \mathtt{u} : \mathtt{A}$ and $\Gamma \vdash \mathtt{u} : \mathtt{B}$ then $\Gamma \vdash \mathtt{A} \equiv \mathtt{B}$.

With more than a 100 rules it is in fact easy to get them wrong (we did). So we proved several meta-theorems which guarantee that the rules fit together and nothing is missing. These are:

1. A useful theorem says that the economic and paranoid rules prove the same judgments. We may thus switch between them as desired.
2. Sanity theorems say things like: if you can show that u is a term of type A then A is a type.
3. Uniqueness of typing, which says that a term has at most one type.
4. There are also several useful (and sometimes tricky) lemmas about inversion and admissibility.

One that is missing but we would like to have is that explicit substitutions can always be computed away. We are open to ideas about how to prove this in the best way.

Paranoia is needless

$\Gamma \vdash_{\text{paranoid}} \mathcal{J}$ if and only if $\Gamma \vdash_{\text{economic}} \mathcal{J}$.

Sanity theorem

If $\Gamma \vdash \mathtt{u} : \mathtt{A}$ then Γ `context` and $\Gamma \vdash \mathtt{A}$ `type`.

Uniqueness of typing

If $\Gamma \vdash \mathtt{u} : \mathtt{A}$ and $\Gamma \vdash \mathtt{u} : \mathtt{B}$ then $\Gamma \vdash \mathtt{A} \equiv \mathtt{B}$.

… and inversion principles and admissibility lemmas.

With more than a 100 rules it is in fact easy to get them wrong (we did). So we proved several meta-theorems which guarantee that the rules fit together and nothing is missing. These are:

1. A useful theorem says that the economic and paranoid rules prove the same judgments. We may thus switch between them as desired.
2. Sanity theorems say things like: if you can show that u is a term of type A then A is a type.
3. Uniqueness of typing, which says that a term has at most one type.
4. There are also several useful (and sometimes tricky) lemmas about inversion and admissibility.

One that is missing but we would like to have is that explicit substitutions can always be computed away. We are open to ideas about how to prove this in the best way.

Theorem:
Function extensionality is not derivable in MLTT.

Test case: a proof that function extensionality is not derivable (following Boulier, Pédrot and Tabareau). The actual proof is a very boring translation of type theory into type theory, with few interesting cases involving products. A perfect job for a proof assistant.

Theorem:
Function extensionality is not derivable in MLTT.

Proof idea (Boulier, Pédrot & Tabareau):
reinterpret `A → B` as `(A → B) × bool`.

Execution: prove that MLTT can be *soundly* translated into MLTT this way. This is best done using a proof assistant.

Test case: a proof that function extensionality is not derivable (following Boulier, Pédrot and Tabareau). The actual proof is a very boring translation of type theory into type theory, with few interesting cases involving products. A perfect job for a proof assistant.

What next?

- improve the library
- find some users
- contributions are welcome

  https://github.com/TheoWinterhalter/formal-type-theory

We're in the process of cleaning up the library and writing more documentation.
We'd be much more motivated if we had some users.
And ideally the users would just do everything by themselves, so please contribute!

On a more serious note, our experience is that if one wants to do type theory *from scratch* and *without shortcuts* then there are many, many details one has to attend to, and it is easy to make small mistakes and omissions. It's not a job for humans, but for computers.

The moral

If you can use Cogda in a shallow way
to do meta-theory of type theory,
then by all means do!

The moral

If you can use Cogda in a shallow way
to do meta-theory of type theory,
then by all means do!

But don’t be shallow just because you use Cogda.

This material is based upon work supported by the Air Force Office of Scientific Research,
Air Force Materiel Command, USAF under Award No. FA9550-14-1-0096