A general definition of dependent type theories

Philipp Haselwarter[1]

j.w.w. Andrej Bauer[1], Peter Lumsdaine[2]

[1]University of Ljubljana, Slovenia

[2]Stockholm University, Sweden

Logic and Semantics seminar,
Aarhus, 15.04.2019

[1]This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0326.

1. Hi, thank you Bas. I'm a finishing PhD student with Andrej Bauer in Ljubljana. I have been working on different aspects of type theory: theory, formalisation, implementation. I will be in Aarhus for the next two weeks.
2. Today I will present some *ongoing* work with Andrej and Peter on a general definition of dependent type theories.

Goals & approach

- Goal: Give a *precise* and *simple* definition of type theories, so that we can prove basic meta theorems and give general semantics; directly applicable to known theories.
- Method: Traditional, stratified approach via raw syntax, raw rules, raw type theories, well-formed rules, well-formed type theories.
- We deliberately avoid using fancy technology (QIITs, ...).
- Everything in this talk should be "just what you'd expect if you wrote it all out".
- Formalised in Coq.
- Implemented in the upcoming Andromeda 2.

1. Our goal with this project is to give a precise and simple definition of type theories, and then use this definition to prove meta theorems that hold for all of these theories, or identify properties that make them provable. We also want to be able to give a general semantics, and we want all of this machinery to be readily applicable to known theories.
2. The way we approached this task is very traditional, in the sense that we first define a signature, then raw syntax over a signature, then rules ecetera, in a stratified way. While there are certainly other ways one might approach this questions with their own merits, I will talk about what *we* did today.
3. If we succeed with the first, and the last three points here on this slide, then I think our method has its merits. So if you're familiar with type theory, there should be few surprises in this talk. If there *are* surprises, then please do interrupt me and ask about me about it.
4. I think people here have heard about the Coq proof assistant. Andromeda is a prover we develop in Ljubljana. For my PhD thesis, I did a lot of things to it, such as adding a meta-language with algebraic effects and handlers, and what we internally call Andromeda 2 is an implementation of general type theories. But lately I'm not allowed to code anymore, so I focus on theory, and writing my thesis. The fact we were able to implement this using OCaml as our meta-language is a benefit of keeping it "simple", and generally we're quite foundationally independent.
5. My talk will follow the progression listed here under "method", and finish on some of the meta-theorems that we have proved.

What kind of type theories?

Four judgement forms:

$$\Gamma \vdash A \text{ type} \qquad \Gamma \vdash s : A \qquad \Gamma \vdash A \equiv B \qquad \Gamma \vdash s \equiv t : A$$

(Think of) contexts as lists of types, and of variables as de Bruijn indices.

Whenever you see lists of some X, think families on X (potentially infinitary contexts, syntax, rules, theories) where "a family on X" is just a map $I \to X$.

1. So let's take a moment to delimit the kinds of theories that we set out to cover.
2. We have the familiar four judgement forms, first the two object judgement forms, and then the equality judgement forms. explain each
3. Our contexts are intuitionistic. I will present them as lists in the talk, even though our formal treatment is a bit more general. The same goes for variables: Today we'll just use de Bruijn indices (did I say that right, Bas?), but we're not married to one particular presentation of variables.
4. This covers a lot of theories that we're interested in.
5. Bonus: Examples: MLTT, MLTT with equality reflection, universes, book HoTT
6. Bonus: Counterexamples: cumulative universes (any kind of subtyping), other judgement forms such as fibrancy, guardedness, etc, non-intuitionistic contexts

Arities and signatures

Definition

An arity is a family of pairs $\langle (class, bind)_i \mid i \in I \rangle$, where $class \in \{\mathsf{ty}, \mathsf{tm}\}$ and $bind \in \mathbb{N}$.

Definition

A signature Σ is a family of pairs $\langle (class, arity)_i \mid i \in I \rangle$.

$$
\begin{aligned}
\mathbb{N} &\mapsto (\mathsf{ty}, []), \\
S &\mapsto (\mathsf{tm}, [(\mathsf{tm}, 0)]), \\
\Pi &\mapsto (\mathsf{ty}, [(\mathsf{ty}, 0), (\mathsf{ty}, 1)]), \\
\lambda &\mapsto (\mathsf{tm}, [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 1)]), \\
\mathsf{app} &\mapsto (\mathsf{tm}, [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 0), (\mathsf{tm}, 0)]).
\end{aligned}
$$

1. No typing info about term arguments or variables bound

Raw expressions

Definition (Raw expressions)

$\mathrm{Expr}_\Sigma^{class}(n)$ is inductively generated by

$$\begin{array}{ll} \mathsf{var}_i & \text{for } 0 \leq i < n \\ S(e) & \text{for } S \in \Sigma,\ e \in \prod_{i \in \arg S} \mathrm{Expr}_\Sigma^{\mathrm{cl}_S\, i}(\gamma \oplus \mathrm{bind}_S\, i) \end{array}$$

1. Expressions are a family indexed by signature and by the *positions* of a raw context. The positions of a context tells us what variables are available.
   For a de Bruijn system, this is simply a finite set of size the length of the context.
   NB: this is one way to "cut the knot": we don't have to know what contexts are yet (which *will* in turn rely on the definition of expressions), but we can still define expressions indexed over their *positions*.
2. This Pi here in the definition is a *meta-level* dependent product
3. pause
4. α is a *meta-variable extension*, A, B are metavariables. In particular, they can take only *term* arguments, not types, and cannot bind anything. Something good happened here: we have a formal account of meta variables that allows us to write $\mathsf{B}(\mathsf{var}_0)$ without hand-waving.

Raw expressions

Definition (Raw expressions)

$\mathrm{Expr}_\Sigma^{class}(n)$ is inductively generated by

$$\begin{array}{ll} \mathsf{var}_i & \text{for } 0 \le i < n \\ S(e) & \text{for } S \in \Sigma,\ e \in \prod_{i \in \arg S} \mathrm{Expr}_\Sigma^{\mathrm{cl}_S\, i}(\gamma \oplus \mathrm{bind}_S\, i) \end{array}$$

Recall $\lambda \mapsto (\mathsf{tm}, [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 1)])$.
Example:

$$\lambda(\mathbb{N}, \mathbb{N}, S(S(\mathsf{var}_0))) \in \mathrm{Expr}_\Sigma^{\mathsf{tm}}(0)$$
$$\lambda(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{var}_0) \in \mathrm{Expr}_{\Sigma+\alpha}^{\mathsf{tm}}(0)$$

where $\alpha = [\mathsf{A} \mapsto (ty, []), \mathsf{B} \mapsto (ty, [(tm, 0)])]$

1. Expressions are a family indexed by signature and by the *positions* of a raw context. The positions of a context tells us what variables are available.
   For a de Bruijn system, this is simply a finite set of size the length of the context.
   NB: this is one way to "cut the knot": we don't have to know what contexts are yet (which *will* in turn rely on the definition of expressions), but we can still define expressions indexed over their *positions*.
2. This Pi here in the definition is a *meta-level* dependent product
3. pause
4. α is a *meta-variable extension*, A, B are metavariables. In particular, they can take only *term* arguments, not types, and cannot bind anything. Something good happened here: we have a formal account of meta variables that allows us to write $\mathsf{B}(\mathsf{var}_0)$ without hand-waving.

Raw contexts and (hypothetical) judgements

Definition

A *raw context* Γ is a shape n (written $|\Gamma|$) with a map $\mathsf{Fin}(n) \to \mathrm{Expr}^{\mathsf{ty}}_{\Sigma}(n)$, denoted as $[0 \colon A_0, \ldots, n-1 \colon A_{n-1}]$, where $A_i \in \mathrm{Expr}^{\mathsf{ty}}_{\Sigma}(|\Gamma|)$.

1. There really should be no surprises here: Contexts are lists of types, one for each variable.
2. pause
3. A substitution from gamma to delta associates to each variable in delta a term expression over gamma. Substitutions act on expressions by replacing the variables of delta by the corresponding terms. We are careful to extend the substitution appropriately when we descend under binders. This is the $\mathrm{id}_{\mathrm{bind}_S\, i}$ part of the second line of the definition.
4. There are also maps between signatures, and they also act on judgements, and these interact nicely, but I won't go into that here. It's the kind of thing that shows up in the formalisation, but is otherwise easy to sweep under the rug.
5. Bonus: The *extension* $f \oplus \eta : \gamma \oplus \eta \to \delta \oplus \eta$ by a shape η is the substitution

$$\begin{aligned} (f \oplus \eta)(i) &= f(i) && \text{if } i \in |\delta|, \\ (f \oplus \eta)(j) &= \mathsf{var}_j && \text{if } j \in |\eta|. \end{aligned}$$

6. Bonus: Substitutions $f : \gamma \to \delta$ and $g : \delta \to \eta$ may be *composed* to give a substitution $g \circ f : \gamma \to \eta$, defined by $(g \circ f)(k) = f^*(g(k))$.

Raw contexts and (hypothetical) judgements

Definition

A *raw context* Γ is a shape n (written $|\Gamma|$) with a map $\mathsf{Fin}(n) \to \mathrm{Expr}^{\mathsf{ty}}_\Sigma(n)$, denoted as $[0 \colon A_0, \ldots, n-1 \colon A_{n-1}]$, where $A_i \in \mathrm{Expr}^{\mathsf{ty}}_\Sigma(|\Gamma|)$.

Definition

A *raw substitution* $f : \Gamma \to \Delta$ over a signature Σ is a map $f : |\Delta| \to \mathrm{Expr}^{\mathsf{tm}}_\Sigma(\Gamma)$. The (contravariant) functorial *action* of f on an expression $e \in \mathrm{Expr}^{c}_\Sigma(|\Delta|)$ gives the expression $f^*e \in \mathrm{Expr}^{c}_\Sigma(|\Gamma|)$, as follows:

$$
\begin{aligned}
f^*(\mathsf{var}_i) &:= f(i), \\
f^*(S(e)) &:= S(\langle (f \oplus \mathrm{id}_{\mathrm{bind}_S\, i})^*(e_i) \rangle_{i \in \arg S}).
\end{aligned}
$$

1. There really should be no surprises here: Contexts are lists of types, one for each variable.
2. pause
3. A substitution from gamma to delta associates to each variable in delta a term expression over gamma. Substitutions act on expressions by replacing the variables of delta by the corresponding terms. We are careful to extend the substitution appropriately when we descend under binders. This is the $\mathrm{id}_{\mathrm{bind}_S\, i}$ part of the second line of the definition.
4. There are also maps between signatures, and they also act on judgements, and these interact nicely, but I won't go into that here. It's the kind of thing that shows up in the formalisation, but is otherwise easy to sweep under the rug.
5. Bonus: The *extension* $f \oplus \eta : \gamma \oplus \eta \to \delta \oplus \eta$ by a shape η is the substitution

$$
\begin{aligned}
(f \oplus \eta)(i) &= f(i) && \text{if } i \in |\delta|, \\
(f \oplus \eta)(j) &= \mathsf{var}_j && \text{if } j \in |\eta|.
\end{aligned}
$$

6. Bonus: Substitutions $f : \gamma \to \delta$ and $g : \delta \to \eta$ may be *composed* to give a substitution $g \circ f : \gamma \to \eta$, defined by $(g \circ f)(k) = f^*(g(k))$.

Judgements

Definition

Given a raw context Γ and a judgement form $\phi \in \{\mathsf{ty}, \mathsf{tm}, \mathsf{tyeq}, \mathsf{tmeq}\}$, a *hypothetical judgement* of that form over Γ is a map J taking each slot of ϕ of syntactic class c to an element of $\mathrm{Expr}^c_\Sigma(\Gamma)$.

For example, a possible tmeq judgement over context $[0 : \mathbb{N}]$ is

$$[lhs \mapsto \mathsf{var}_0,\ rhs \mapsto \mathsf{var}_0,\ ty \mapsto \mathbb{N}]$$

written more conveniently as

$$[0 : \mathbb{N}] \vdash \mathsf{var}_0 \equiv \mathsf{var}_0 : \mathbb{N}$$

1. So like I promised, there are four kinds of judgements, which are formed by filling in the holes with expressions of the correct syntactic class.

Raw rules

Definition

A *raw rule* R over a signature Σ consists of an arity α_R, together with a family of judgements over the extended signature $\Sigma + \alpha_R$, the *premises* of R, and one more judgement over $\Sigma + \alpha_R$, the *conclusion* of R.

1. We now get to the point where have the constituent components of type theory and things should get a little bit more interesting. We can start thinking about deriving judgements. Let's define what a raw *rule* is.
2. pause
3. Let's walk through the definition by following the example of the application rule.
4. These raw rules serve as *templates* for the deductive rules of our type theories. What I mean by that is that the meta-variables introduced by the arity α_R may be instantiated with any particular expressions of the correct class and arity to derive concrete judgements. The derivations are simply generated by the instantiated rules.

Raw rules

Definition

A *raw rule* R over a signature Σ consists of an arity α_R, together with a family of judgements over the extended signature $\Sigma + \alpha_R$, the *premises* of R, and one more judgement over $\Sigma + \alpha_R$, the *conclusion* of R.

For example, function application, app, has arity

$$\alpha_{\mathsf{app}} = [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 0), (\mathsf{tm}, 0)]$$

and premises & conclusion as follows

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad [0 : \mathsf{A}] \vdash \mathsf{B}(\mathsf{var}_0)\ \mathsf{type} \qquad \vdash \mathsf{s} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)) \qquad \vdash \mathsf{t} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{s}, \mathsf{t}) : \mathsf{B}(\mathsf{t})}$$

1. We now get to the point where have the constituent components of type theory and things should get a little bit more interesting. We can start thinking about deriving judgements. Let's define what a raw *rule* is.
2. pause
3. Let's walk through the definition by following the example of the application rule.
4. These raw rules serve as *templates* for the deductive rules of our type theories. What I mean by that is that the meta-variables introduced by the arity α_R may be instantiated with any particular expressions of the correct class and arity to derive concrete judgements. The derivations are simply generated by the instantiated rules.

Structural rules

The structural rules for a signature Σ are given by:

- for each Γ, $i \in |\Gamma|$, there is a rule

$$\frac{\Gamma \vdash \Gamma_i \text{ type}}{\Gamma \vdash \mathsf{var}_i : \Gamma_i}$$

- for each $f : \Delta \to \Gamma$, and $\Gamma \vdash J$:

$$\frac{\Gamma \vdash J \qquad \Delta \vdash f(i) : f^*\Gamma_i \ \text{ for each } i \in |\Gamma|}{\Delta \vdash f^*J}$$

  $+$ two more substitution rules

- judgemental equality is an equivalence relation
- conversion rules

1. The structural rules for some signature sigma are divided into four families:
   - the variable rules,
   - rules for substitutions,
   - rules stating that equality is an equivalence relation, and
   - rules for conversion of terms and term equations between equal types.
2. There are two more substitution rules that tell us that when pointwise judgementally equal substitutions act on expressions, they yield equal results.
3. There are the usual rules for introducing and using judgemental equality, reflexivity, symmetry, and transitivity, and conversion.
4. All type theories over Σ will be assumed to contain these rules unless we explicitly state otherwise.

APP-1

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad [0 : \mathsf{A}] \vdash \mathsf{B}(\mathsf{var}_0)\ \mathsf{type} \qquad \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}$$

APP-2

$$\frac{(\mathsf{A}, \mathsf{B} : \text{same as APP-1}) \qquad \vdash \mathsf{f} : \mathsf{A} \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}$$

APP-3

$$\frac{(\mathsf{A}, \mathsf{B} : \text{same}) \qquad \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)) \qquad \vdash \mathsf{a} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0))}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}$$

APP-4

$$\frac{(\mathsf{A}, \mathsf{B} : \text{same}) \qquad \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)) \qquad \vdash \mathsf{a} : \mathsf{A} \qquad \vdash \mathsf{a} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}$$

APP-5

$$\frac{\vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}$$

1. I can't very well give a talk about type theory without having one slide that's just inference rules, so here we are.
2. Rule number one is the application rule we've seen before, and unless I made a typo, there should be nothing wrong with it.
3. The second rule is a bit funny because it says that f should have type A. This should probably not be called "app", but formally there's nothing wrong with having such a rule.
4. The third rule instead changes the type of a, and this is somehow more troubling, because in the conclusion, we form $B(a)$. But B expects to bind a term of type A. So if we want $B(a)$ to be a valid type, and that's not unreasonable because it appears here in the conclusion, we will need to know that the theory that this rule is a part of allows us to derive that A and $\Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0))$ are equal.
5. Rule number four introduces a twice, at different types. That seems problematic.
6. Finally, rule number five is something you will often encounter in the wild, and for specific theories it may make sense. What's fishy here, is that we don't have any premises asserting the type-hood of A and B. If your theory is sufficiently well-behaved, you may be able to prove an inversion theorem that allows you to construct such premises from the fact that A and B appear in the Π type here in premise one, but in general, we require that each meta-variable is introduced in its own premise.

$$\textsc{app-1}\ \frac{\vdash A\ \text{type} \qquad [0 : A] \vdash B(\mathsf{var}_0)\ \text{type} \qquad \vdash f : \Pi(A, B(\mathsf{var}_0)) \qquad \vdash a : A}{\vdash \mathsf{app}(A, B(\mathsf{var}_0), f, a) : B(a)}$$

$$\textsc{app-2}\ \frac{(A, B : \text{same as } \textsc{app-1}) \qquad \vdash f : A \qquad \vdash a : A}{\vdash \mathsf{app}(A, B(\mathsf{var}_0), f, a) : B(a)}$$

$$\textsc{app-3}\ \frac{(A, B : \text{same}) \qquad \vdash f : \Pi(A, B(\mathsf{var}_0)) \qquad \vdash a : \Pi(A, B(\mathsf{var}_0))}{\vdash \mathsf{app}(A, B(\mathsf{var}_0), f, a) : B(a)}$$

$$\textsc{app-4}\ \frac{(A, B : \text{same}) \qquad \vdash f : \Pi(A, B(\mathsf{var}_0)) \qquad \vdash a : A \qquad \vdash a : \Pi(A, B(\mathsf{var}_0)}{\vdash \mathsf{app}(A, B(\mathsf{var}_0), f, a) : B(a)}$$

$$\textsc{app-5}\ \frac{\vdash f : \Pi(A, B(\mathsf{var}_0)) \qquad \vdash a : A}{\vdash \mathsf{app}(A, B(\mathsf{var}_0), f, a) : B(a)}$$

1. We identified two conditions that capture these problems. The details are technical and not very enlightening, so I'll instead tell you what they mean morally.
2. The first property is tightness. We say that a rule is *tight*, when the arity of the rule matches the object premises: There is a correct number of premises, with matching syntactic classes and arities. Remember that the arity of the rule determines what meta-variables are available.
3. Rules 1, 2, and 3 are tight, 4 and 5 are not.
4. If a rule is tight, the meta-variables it introduces exactly match the premises that are object judgements.
5. Rules three and five motivate the second property. What went wrong in both, is that we were unable to explain that the types that appear in the conclusion were in fact derivable. Let's call the typehood judgement associated to "the type that appears a term judgement" a presupposition of that judgement. The equality judgements also have presuppositions, and there as well, when we assert that an equation holds, we usually want to know that the constituent parts are derivable.
6. The second property is presuppositionality. A rule R is presuppositional over a type theory T, if T derives the presuppositions of the conclusion of R from the premises of R.

Tight rules

Definition

We say that R is *tight* when there exists a bijection β between the arguments of the arity α_R and the object premises of R, such that for each argument i of α_R,

1. the context of the premise $\beta(i)$ has the shape $\mathrm{bind}_{\alpha_R}\, i$;
2. the judgement form of the premise $\beta(i)$ is $\mathrm{cl}_{\alpha_R}\, i$;
3. the head expression of the premise $\beta(i)$ is $\mathsf{meta}_i(\langle \mathsf{var}_j \rangle_{j \in \mathrm{bind}_{\alpha_R}\, i})$.

$$\textsc{app-5}\ \frac{\vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}$$

Presuppositional rules

$$\begin{aligned}
\mathrm{Presup}\,(\Gamma \vdash A\ \mathsf{type}) &= [],\\
\mathrm{Presup}\,(\Gamma \vdash s : A) &= [\Gamma \vdash A\ \mathsf{type}],\\
\mathrm{Presup}\,(\Gamma \vdash A \equiv B) &= [\Gamma \vdash A\ \mathsf{type}, \Gamma \vdash B\ \mathsf{type}],\\
\mathrm{Presup}\,(\Gamma \vdash s \equiv t : A) &= [\Gamma \vdash A\ \mathsf{type}, \Gamma \vdash s : A, \Gamma \vdash t : A].
\end{aligned}$$

Definition

Let T be a raw type theory over Σ and R a raw rule: A raw rule R is *presuppositional over* T when every presupposition of the conclusion of R is derivable in T (translated from Σ to $\Sigma + \alpha_R$) from the premises of R and the presuppositions of the premises of R

$$\textsc{app-5}\ \frac{\vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}$$

1. So here's the formal definition of presuppositionality, but let's not dwell on that.

A word on preventing circularity

$$\frac{}{\vdash \mathsf{ty} : \mathsf{El}(\mathsf{ty})}(1) \qquad \frac{\vdash \mathsf{a} : \mathsf{El}(\mathsf{ty})}{\vdash \mathsf{El}(\mathsf{a})\ \mathsf{type}}(2)$$

The presuppositions of (1) and (2) are $\vdash \mathsf{El}(\mathsf{ty})$ type, which can be derived using both rules. How to make sense of this?

1. I want to briefly talk about how we deal with potential non-well-foundedness. A good example for circularity is the following theory: Let's have type in type, Tarski style.
2. Issues with consistency aside, we can't even make good sense of the presuppositions here. If we look at the presuppositions of rule one here, we see that we need to show that $El(ty)$ is a type. We can do that, by using rule two where we instantiate the meta-variable a with the term symbol ty. But then we also need to provide a derivation for the premise, namely that the symbol ty has type $El(ty)$, which we can do, using rule one. But now we're using rule one to show that one of its presuppositions holds.
3. pause
4. We can require that our signatures, our raw contexts, and our raw rules come with a well-ordering.
5. Bonus: We can save type in type by introducing an additional constant and a defining equation

$$\vdash ty : Ty \qquad \frac{\vdash a : Ty}{\vdash El(a)\ \mathsf{type}} \qquad \vdash Ty\ \mathsf{type} \qquad \vdash El(ty) \equiv Ty$$

A word on preventing circularity

$$\frac{}{\vdash \mathsf{ty} : \mathsf{El}(\mathsf{ty})}(1) \qquad \frac{\vdash \mathsf{a} : \mathsf{El}(\mathsf{ty})}{\vdash \mathsf{El}(\mathsf{a})\ \mathsf{type}}(2)$$

The presuppositions of (1) and (2) are $\vdash \mathsf{El}(\mathsf{ty})$ type, which can be derived using both rules. How to make sense of this?

The way we avoid such problems is of course that we require a well-founded order on the types expressions in a context, on the symbols in a signature, and on premises of each rule.

1. I want to briefly talk about how we deal with potential non-well-foundedness. A good example for circularity is the following theory: Let's have type in type, Tarski style.
2. Issues with consistency aside, we can't even make good sense of the presuppositions here. If we look at the presuppositions of rule one here, we see that we need to show that $El(ty)$ is a type. We can do that, by using rule two where we instantiate the meta-variable a with the term symbol ty. But then we also need to provide a derivation for the premise, namely that the symbol ty has type $El(ty)$, which we can do, using rule one. But now we're using rule one to show that one of its presuppositions holds.
3. pause
4. We can require that our signatures, our raw contexts, and our raw rules come with a well-ordering.
5. Bonus: We can save type in type by introducing an additional constant and a defining equation

   $\vdash ty : Ty \quad \dfrac{\vdash a : Ty}{\vdash El(a)\ \mathsf{type}} \quad \vdash Ty\ \mathsf{type} \quad \vdash El(ty) \equiv Ty$

Meta theorems

Theorem

The substitution rules are admissible.

Theorem

Suppose all rules of a raw type theory T are presuppository. If T derives a judgement, then it derives all of its presuppositions.

Theorem

Suppose all rules of a type theory T are tight, and the symbols in Σ are in bijection with the object rules of T. If T derives $\Gamma \vdash A$ type, $\Gamma \vdash B$ type, $\Gamma \vdash t : A$ and $\Gamma \vdash t : B$ then it also derives $\Gamma \vdash A \equiv B$.

Semantics — more soon.