Second-order generalised algebraic theories, by examples (under construction)

Ambrus Kaposi

September 11, 2025

In these notes, we study non-substructural programming languages at a high level of abstraction: a language is a second-order generalised algebraic theory (SOGAT). The syntax of such a language is its initial model, in which there are only well-typed (intrinsic) terms quotiented by conversion, every operation is automatically a congruence with respect to conversion and every operation is stable under substitution. These notes are pedagogical excerpts of the papers [KX24, BKS23]. A more categorical treatment can be found in the upcoming PhD thesis of Rafaël Bocquet.

Related ideas are higher-order abstract syntax [Hof99], logical frameworks [HHP93], two-level type theories [ACKS23], synthethic Tait computability [Ste22]. We will rely on being able to work informally in type theory (informal Agda, Coq, Idris or Lean).

These notes can be formalised in the following metatheories: observational type theory, extensional type theory with quotient inductive-inductive types and propositional extensionality, homotopy type theory, constructive set theory.

Contents

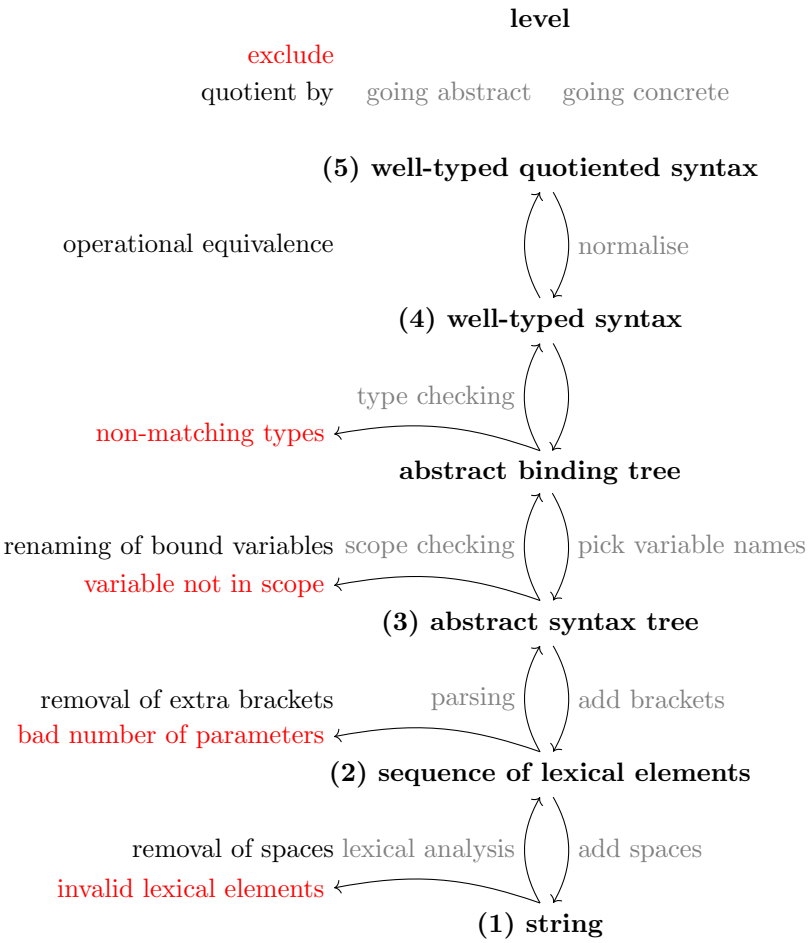
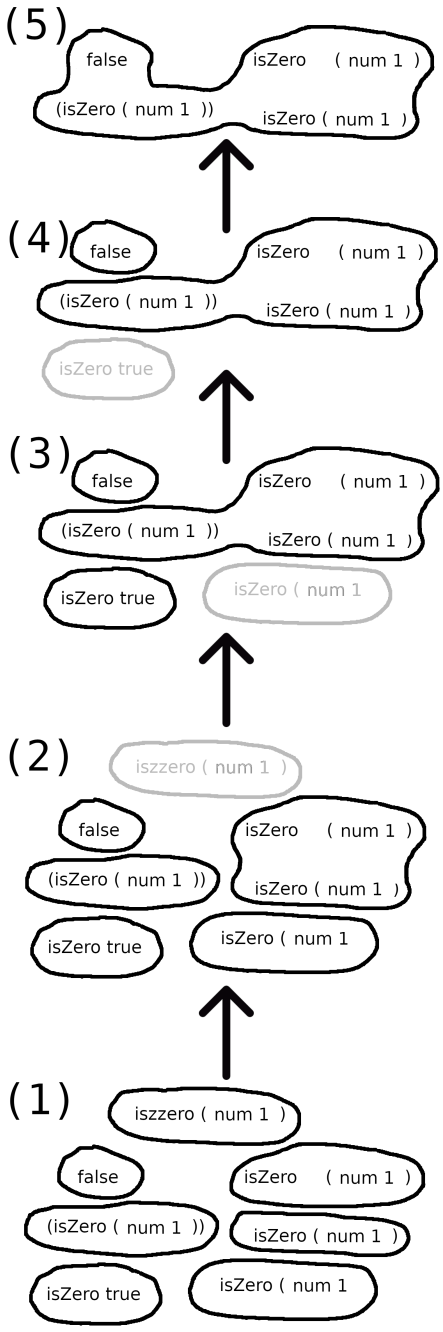


Figure 1: Different levels of abstraction when defining a programming language and transformations between levels. At more abstract levels, certain programs are excluded and others identified. Abstract binding trees are sometimes called well-scoped syntax trees.



(5) well-typed quotiented syntax

```
Ty        :  Set
Tm        :  Ty → Set
Bool      :  Ty
Nat       :  Ty
true      :  Tm Bool
false     :  Tm Bool
ite       :  Tm Bool → Tm A → Tm A → Tm A
num       :  ℕ → Tm Nat
isZero    :  Tm Nat → Tm Bool
_+_       :  Tm Nat → Tm Nat → Tm Nat
iteβ1     :  ite true u v = u
iteβ2     :  ite false u v = v
isZeroβ1  :  isZero (num 0) = true
isZeroβ2  :  isZero (num (1+n)) = false
+β        :  num m + num n = num (m + n)
```

(4) well-typed syntax

```
Ty        :  Set
Tm        :  Ty → Set
Bool      :  Ty
Nat       :  Ty
true      :  Tm Bool
false     :  Tm Bool
ite       :  Tm Bool → Tm A → Tm A → Tm A
num       :  ℕ → Tm Nat
isZero    :  Tm Nat → Tm Bool
_+_       :  Tm Nat → Tm Nat → Tm Nat
```

(3) abstract syntax tree

```
Tm        :  Set
true      :  Tm
false     :  Tm
ite       :  Tm → Tm → Tm → Tm
num       :  ℕ → Tm
isZero    :  Tm → Tm
_+_       :  Tm → Tm → Tm
```

(2) list of the following lexical elements:

```
(, ), true, false, if, then, else, num, isZero, +,
0, 1, 2, 3, ...
```

(1) any string

Figure 2: Left: example Razor programs at different levels of abstraction. Each bubble represents a separate program. Right: description of the Razor expression language at levels (1)–(5).

1 Levels of abstraction

A language can be described in different ways ranging from concrete to abstract, see Figure 1. In this section, we explain the different levels briefly via a simple expression language (Razor, see Subsection 2.3). The following example program can be written in Razor.

```
if isZero (num 0 + num 1) then false else isZero (num 0)
```

A Razor program is either a numeric or a boolean expression. Numbers can be formed using `num i` where `i` is a natural number. Booleans are `true` or `false`. We have the usual if-then-else operator, addition and an isZero operator which says whether a number is 0. The above program evaluates (runs) in the following steps (each new line is a step).

```
if isZero (num 0 + num 1) then false else isZero (num 0)
if isZero (num 1) then false else isZero (num 0)
if false then false else isZero (num 0)
isZero (num 0)
true
```

Figure 2 gives complete descriptions of Razor at levels of abstraction (1)–(5):

(1) As a first approximation, a program is a string, that is, a sequence of (ASCII) characters. This is how we write programs on a computer. Any string is a program. Many strings do not correspond to meaningful programs in our language such as `num 3 - num 2` as we don't have subtraction. Also, there are different strings which represent the same program. For example, `isZero  (num 1)` and `isZero (num 1)` are different as strings but should be the same programs as the extra spaces after `isZero` shouldn't matter. Instead of describing which strings are meaningful programs and defining an equivalence relation for identifying strings that represent the same program, we will describe programs using a more abstract structure.

(2) The more abstract structure is list of lexical elements. Now we have much fewer programs and `num 3 - num 2` is not a program anymore because there is no lexical element for `-`. Any two programs given as strings which differ only in the number of spaces will end up as the same program at this level: `isZero  (num 1)` and `isZero (num 1)` are both given by the sequence [`isZero`, `(`, `num`, `1`, `)`]. However, we still have meaningless programs, e.g. [`(`, `true`] (there is no closing parenthesis) or [`num`, `1`, `+`] (`+` needs two arguments), and so on. Also, there are programs which could be identified, e.g. [`(`, `true`, `)`] and [`true`] (the parentheses are redundant in the former). Again, to solve these issues, we move to a higher-level representation of programs. Note that there are standard ways to navigate between levels (1) and (2): (2) to (1) is printing. (1) to (2) is performed by a lexical analyser (lexer) which turns a string into a sequence of lexical elements or returns an error.

(3) Abstract syntax tree descriptions are usually given by BNF grammars. At this level, we only have well-parenthesised expressions and each operator receives the correct number of arguments. Programs are now trees which have `true`, `false` or `num i` at their leaves and they can have ternary branching with `ite` at the branching node, unary branching with `isZero` at the node or binary branching with `+` at the node.

(4) In well-typed (intrinsic) syntax trees, the types of the arguments are restricted to the correct ones.

(5) In well-typed quotiented syntax, two programs which have the same result are identified.

Abstract binding trees are not relevant for Razor as there are no variables.

The *extrinsic* approach moves to more abstract levels by defining relations which select the well-formed (well-scoped, well-typed, etc) expressions in the concrete representation. In contrast, we use the *intrinsic* approach where in the more abstract representation the non-well-formed expressions are not even expressible. At level (5), there is no need to prove that conversion preserves typing ("preservation" from "progress and preservation") because the equations are already expressed in a typed way. Certain properties of the language cannot be expressed at the abstract levels. For example it is not possible to count the number of brackets in a program at level (3). At level (6), it is not possible to reason about program efficiency because all convertible programs are in the same equivalence class: we cannot write a function which distinguishes convertible programs. We view the lower levels as important parts of a programming language, but we see them as intermediate technical steps to reach the most abstract level which describes the essence of the language.

In these notes, we will use the most abstract level to describe languages.

2 Derivability and admissibility in GATs

As a warmup, we review how languages without binders can be seen as generalised algebraic theories (GATs). The goal of this section is to familiarise the reader with the following concepts: model, derivability, morphism, dependent model, dependent morphism, syntax, induction, iteration, admissibility, logical consistency, equational consistency, normal forms, normalisation. These concepts are specific to the particular GAT, and we show what they are for different example GATs. By the end of this section, the reader should be able to formulate them for any GAT (except normal forms which only exist for certain languages).

2.1 Monoid

We start with a well-known algebraic theory (AT): monoids. A model of the theory of monoid is also called a monoid algebra or simply monoid.

Definition 1 (Monoid)**.** *A monoid model comprises the following components:*

$$\begin{aligned}
&\mathsf{C} && : \mathsf{Set}\\
&- \cdot - && : \mathsf{C} \to \mathsf{C} \to \mathsf{C}\\
&\mathsf{ass} && : x \cdot (y \cdot z) = (x \cdot y) \cdot z\\
&\mathsf{u} && : \mathsf{C}\\
&\mathsf{idl} && : \mathsf{u} \cdot x = x\\
&\mathsf{idr} && : x \cdot \mathsf{u} = x
\end{aligned}$$

A model contains one carrier set (sort), two operations (one binary and one nullary) which satisfy three equations.

Example models (the equations also hold, but we don't write their proofs):

$\mathsf{C} := \mathbb{N}$	$\mathsf{C} := \mathbb{N}$	$\mathsf{C} := \{*\}$	$\mathsf{C} := \{\mathsf{tt}, \mathsf{ff}\}$
$x \cdot y := x + y$	$x \cdot y := x * y$	$x \cdot y := *$	$x \cdot y := x \wedge y$
$\mathsf{u} := 0$	$\mathsf{u} := 1$	$\mathsf{u} := *$	$\mathsf{u} := \mathsf{tt}$

Exercise 2. *Prove the equations for the above four monoids. Define all the monoids with sort* $\{\mathsf{tt}, \mathsf{ff}\}$*. Define the following monoids: strings with concatenation, square matrices with multiplication,* $A \to A$ *functions with composition, subsets of* A *with intersection/union.*

Non-examples: C is the empty set, $\mathsf{C} = \mathbb{N}$ with exponentiation, $\mathsf{C} = \mathbb{Z}$ with subtraction. We give names to the models and refer to their components via subscript. E.g. if we call the above first model M, then $\mathsf{C}_M = \mathbb{N}$, $x \cdot_M y = x + y$ and $\mathsf{u}_M = 0$.

A *derivable operation* is one that is defined for any model, for example $\mathsf{dup}\,(x : \mathsf{C}) : \mathsf{C} := x \cdot x$. A *derivable equation* is one that holds in any model, for example $(\mathsf{u} \cdot \mathsf{u}) \cdot \mathsf{u} = \mathsf{u}$ which is derived by

$$(\mathsf{u} \cdot \mathsf{u}) \cdot \mathsf{u} \overset{\mathsf{idr}}{=} \mathsf{u} \cdot \mathsf{u} \overset{\mathsf{idl}}{=} \mathsf{u}.$$

In a proof assistant, we can define derivable operations and equations by assuming a model as a module parameter / postulate / axiom / variable, and only using this when defining the operation or proving the equation. Then we can specialise the derivable things to particular models, for the above M, we have $\mathsf{dup}_M\, 3 = 3 \cdot_M 3 = 3 + 3 = 6$ and $(\mathsf{u}_M \cdot_M \mathsf{u}_M) \cdot_M \mathsf{u}_M = (0 + 0) + 0 = 0 + 0 = 0 = \mathsf{u}_M$.

A *morphism* (or homomorphism) between models is a function between the carriers that preserves the operations, precisely a morphism from M to N comprises the following components:

$$\begin{aligned}
&\mathsf{C} && : \mathsf{C}_M \to \mathsf{C}_N\\
&- \cdot - && : (x\, y : \mathsf{C}_M) \to \mathsf{C}\,(x \cdot_M y) = \mathsf{C}\,x \cdot_N \mathsf{C}\,y\\
&\mathsf{u} && : \mathsf{C}\,\mathsf{u}_M = \mathsf{u}_N
\end{aligned}$$

Note that there are no components corresponding to the equations (this only changes when the GAT has sort equations).

Exercise 3. *Define all the morphisms between any pair of models from the above four examples.*

A *dependent model* (displayed model, motive and methods of the induction principle) over a model M has the same number of components as a model, and is dependent over them, that is:

$$\begin{aligned}
&\mathsf{C} && : \mathsf{C}_M \to \mathsf{Set}\\
&- \cdot - && : \mathsf{C}\,x_M \to \mathsf{C}\,y_M \to \mathsf{C}\,(x_M \cdot_M y_M)\\
&\mathsf{ass} && : x \cdot (y \cdot z) = (x \cdot y) \cdot z\\
&\mathsf{u} && : \mathsf{C}\,\mathsf{u}_M\\
&\mathsf{idl} && : \mathsf{u} \cdot x = x\\
&\mathsf{idr} && : x \cdot \mathsf{u} = x
\end{aligned}$$

Here we used (the somewhat extreme) notation where metavariables have subscripts. $- \cdot -$ has two implicit arguments x_M and y_M, both in C_M (the equations ass, idl, idr in the notion of model also had implicit arguments). Note that ass also depends on ass_M: we have $x : \mathsf{C}\,x_M$, $y : \mathsf{C}\,y_M$, $z : \mathsf{C}\,z_M$ and the left hand side $x \cdot (y \cdot z)$ is in $\mathsf{C}\,(x_M \cdot_M (y_M \cdot_M z_M))$, the right hand side is in $\mathsf{C}\,((x_M \cdot_M y_M) \cdot_M z_M)$. These sets are equal by ass_M. The situation is similar for idl, idr. Examples where $M = (\mathbb{N}, +, 0)$:

$$\begin{aligned}
&\mathsf{C}\,n := \mathsf{Vec}\,\mathbb{N}\,n && \qquad \mathsf{C}\,_ := \{*\}\\
&x \cdot y := x +\!\!+ y && \qquad x \cdot y := *\\
&\mathsf{u} := [] && \qquad \mathsf{u} := *
\end{aligned}$$

Exercise 4. *Any model can be turned into a dependent model where we ignore the dependency.*

Exercise 5. *Any dependent model D over M can be turned into a model together with a morphism into M. The carrier will be $(x_M : \mathsf{C}_M) \times \mathsf{C}_D\,x_M$ (a dependent Descartes-product, or Σ-type in the metatheory).*

A *dependent morphism* (section) from a model M to a dependent model D over M is like a homomorphism, but the function is dependent:

$$\begin{aligned}
&\mathsf{C} && : (x : \mathsf{C}_M) \to \mathsf{C}_D\,x\\
&- \cdot - && : (x\,y : \mathsf{C}_M) \to \mathsf{C}\,(x \cdot_M y) = \mathsf{C}\,x \cdot_D \mathsf{C}\,y\\
&\mathsf{u} && : \mathsf{C}\,\mathsf{u}_M = \mathsf{u}_D
\end{aligned}$$

The *syntax* is a model from which there is a dependent moprhism into any dependent model (the dependent model has to be over the syntax for this to make sense). We denote the syntax by I (for initial model). The function which takes a dependent model over the syntax and returns the dependent morphism is called *induction* (also called (dependent) eliminator, universal property).

Exercise 6. *Show that there is a syntax for monoids (hint: the carrier is a particularly simple set).*

Exercise 7. *Show that that for a given model M, the following two are equivalent:*

- *there is a dependent morphism into any model over M,*
- *there is a unique homomorphism from M into any model (initiality).*

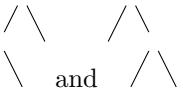
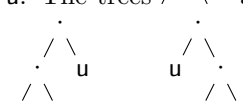
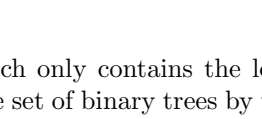
Dependent models and morphisms were introduced in order to specify syntax and induction. Special cases of induction are iteration (fold, catamorphism, non-dependent eliminator, interpreter) and recursion (sometimes also called non-dependent eliminator). The syntax has iteration, which means that for any model M, there is a morphism from I to M. Recursion is the special case of induction where the $\mathsf{C}_\mathsf{I} \to \mathsf{Set}$ component in the dependent model is a constant function.

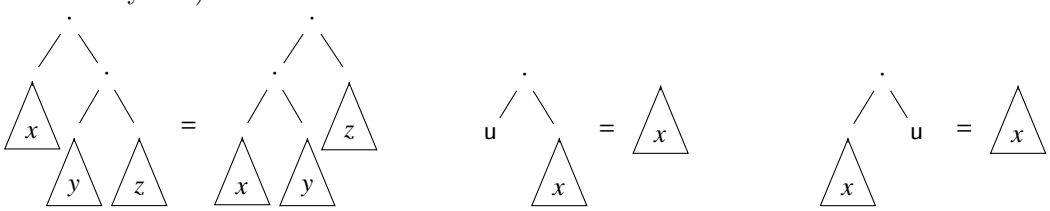
Exercise 8. *There is an identity morphism from any model to itself. Morphisms can be composed. An isomorphism between models M and N (denoted $M \cong N$) comprises morphisms $M \to N$ and $N \to M$ such that there composites are the identity morphisms. Show that any two syntaxes are isomorphic.*

An *admissible* operation / equation is one that can be defined / proven for the syntax via induction. For the syntax of monoids, we prove that for any $x : \mathsf{C}_\mathsf{I}$ (note that I is the syntax), $x = \mathsf{u}_\mathsf{I}$ by defining the following dependent model over I:

$$\begin{aligned}
&\mathsf{C}\,x && := (x = \mathsf{u}_\mathsf{I})\\
&(e : x = \mathsf{u}_\mathsf{I}) \cdot (e' : x' = \mathsf{u}_\mathsf{I}) && : x \cdot_\mathsf{I} x' \overset{e}{=} \mathsf{u}_\mathsf{I} \cdot_\mathsf{I} x' \overset{e'}{=} \mathsf{u}_\mathsf{I} \cdot_\mathsf{I} \mathsf{u}_\mathsf{I} \overset{\mathsf{idl}_\mathsf{I}}{=} \mathsf{u}_\mathsf{I}\\
&\mathsf{u} && : \mathsf{u}_\mathsf{I} = \mathsf{u}_\mathsf{I}
\end{aligned}$$

Via induction, we obtain a function $(x : \mathsf{C}_\mathsf{I}) \to \mathsf{C}\,x$ which is the same as $(x : \mathsf{C}_\mathsf{I}) \to x = \mathsf{u}_\mathsf{I}$. We say that $x = \mathsf{u}$ is an admissible equation, but it is not derivable.

For monoids we were able to define the synta in an ad-hoc way (Exercise 6), but there is a generic way to construct the syntax which works for any algebraic theory. For monoids, we first try to define C_I as the set of binary trees where the binary branching nodes denote $\cdot_\mathsf{I}$ and the leaves denote u. The trees  and  denote $(\mathsf{u} \cdot \mathsf{u}) \cdot \mathsf{u}$ and $\mathsf{u} \cdot (\mathsf{u} \cdot \mathsf{u})$, respectively. Actually, we like to draw them as  to make the connection between the operations and the nodes / leaves of the tree explicit. This definition of C_I however does not suffice. We cannot prove e.g. that the tree  and the tree which only contains the leaf u are equal, so we cannot provide the component idl_I. We solve this by *quotienting* the set of binary trees by the three equations ass, idl and idr, that is, the trees which have the following shapes will be identified (where a triangle denotes any tree):



We call these quotiented trees *syntax trees* (we could call them quotiented syntax trees – however we can consider the language of monoids without equations, and the syntax for that language provides the unquotiented monoid syntax trees). One can prove that the model defined like this is a syntax, that is, it has induction. However, we don't do this, we simply assume that there is a syntax (which just means a model with induction). For a generic construction of syntaxes, see e.g. [KKA19]. In a type theory with support for quotient inductive sets (quotient inductive types), one can define the syntax of monoids as the quotient inductive set with two point-constructors $(- \cdot -)$, u) and three equality (path) constructors (ass, idl and idr). The dependent eliminator for this quotient inductive set exactly says that this model has induction.

2.2 Pointed set with endofunction

This language is particularly simple, it does not have equations.

Definition 9 (PSE)**.** *A model comprises the following components:*

$$\begin{aligned}\mathsf{N} &: \mathsf{Set}\\ \mathsf{z} &: \mathsf{N}\\ \mathsf{s} &: \mathsf{N} \to \mathsf{N}\end{aligned}$$

A derivable function is e.g. $\mathsf{sss}\,(n : \mathsf{N}) : \mathsf{N} := \mathsf{s}\,(\mathsf{s}\,(\mathsf{s}\,n))$. There are no interesting derivable equations.

The syntax is $(\mathbb{N}, 0, +1)$, a dependent model over this contains $\mathsf{N} : \mathbb{N} \to \mathsf{Set}$, $\mathsf{z} : \mathsf{N}\,0$ and $\mathsf{s} : \mathsf{N}\,n \to \mathsf{N}\,(1 + n)$. Induction for $(\mathbb{N}, 0, +1)$ thus says the usual notion of induction: given a (proof-relevant) predicate which holds for 0 and which preserves successor, the predicate holds for all natural numbers. Hence I is the natural numbers. The fact that natural numbers form an exponential semiring is admissible for the language PSE:

Exercise 10. *Define the admissible operations of addition, multiplication, exponentiation. Prove the admissible equations associativity of addition, left and right identity, commutativity, etc.*

Exercise 11. *Show via induction that* $0 \neq 1 + n$ *and that* $+1$ *is injective.*

When we draw syntax trees, they are just unary branching, like $\mathsf{s}\,(\mathsf{s}\,(\mathsf{s}\,\mathsf{z}))$:

Exercise 12. *Show that induction is equivalent to initiality.*

2.3 An expression language

The following is a simple expression language which is not algebraic, but *generalised* algebraic. There are two sorts, and the second one is indexed over the first one.[1] We call the language Razor following [Hut23].

Definition 13 (Razor)**.** *A model comprises the following components:*

$$
\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\mathsf{Bool} & : \mathsf{Ty} \\
\mathsf{Nat} & : \mathsf{Ty} \\
\mathsf{true} & : \mathsf{Tm}\,\mathsf{Bool} \\
\mathsf{false} & : \mathsf{Tm}\,\mathsf{Bool} \\
\mathsf{ite} & : \mathsf{Tm}\,\mathsf{Bool} \to \mathsf{Tm}\,A \to \mathsf{Tm}\,A \to \mathsf{Tm}\,A \\
\mathsf{num} & : \mathbb{N} \to \mathsf{Tm}\,\mathsf{Nat} \\
- + - & : \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\,\mathsf{Nat} \\
\mathsf{isZero} & : \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\,\mathsf{Bool} \\
\mathsf{ite}\beta_1 & : \mathsf{ite}\,\mathsf{true}\,u\,v = u \\
\mathsf{ite}\beta_2 & : \mathsf{ite}\,\mathsf{false}\,u\,v = v \\
+\beta & : \mathsf{num}\,m + \mathsf{num}\,n = \mathsf{num}\,(m+n) \\
\mathsf{isZero}\beta_1 & : \mathsf{isZero}\,(\mathsf{num}\,0) = \mathsf{true} \\
\mathsf{isZero}\beta_2 & : \mathsf{isZero}\,(\mathsf{num}\,(1+n)) = \mathsf{false}
\end{array}
$$

We call elements of Ty types and elements of $\mathsf{Tm}\,A$ terms of type A. There is no single set of terms, instead for each type, there is a separate set of terms of that type. Each operation is *typed*: their input types and output type is restricted. The operation ite has one implicit type argument A and its first explicit argument needs to have type Bool, while the second and third explicit arguments need to have (the same) type A.

The derivable operations can be seen as programs that can be defined in this language, e.g. $\mathsf{not}\,(b : \mathsf{Tm}\,\mathsf{Bool}) : \mathsf{Tm}\,\mathsf{Bool} := \mathsf{ite}\,b\,\mathsf{false}\,\mathsf{true}$. The equations explain how to run the programs, e.g. $\mathsf{not}\,\mathsf{true} = \mathsf{ite}\,\mathsf{true}\,\mathsf{false}\,\mathsf{true} \overset{\mathsf{ite}\beta_1}{=} \mathsf{false}$. Another example:

$$
\begin{array}{ll}
(\mathsf{num}\,1 + \mathsf{num}\,2) + (\mathsf{num}\,3 + \mathsf{num}\,4) & =(+\beta) \\
(\mathsf{num}\,(1+2)) + (\mathsf{num}\,3 + \mathsf{num}\,4) & = \\
\mathsf{num}\,3 + (\mathsf{num}\,3 + \mathsf{num}\,4) & =(+\beta) \\
\mathsf{num}\,3 + \mathsf{num}\,(3+4) & = \\
\mathsf{num}\,3 + \mathsf{num}\,7 & =(+\beta) \\
\mathsf{num}\,(3+7) & = \\
\mathsf{num}\,10 &
\end{array}
$$

Note that in the expression $\mathsf{num}\,(1+2)$, the $+$ refers to the metatheoretic addition, while in $\mathsf{num}\,1 + \mathsf{num}\,2$ the $+$ refers to object theoretic addition.

The standard (metacircular, set) model of Razor is where types are sets (metatheoretic types) and terms

[1] Certain such GATs can be reduced to ATs by replacing the Ty-indexing with a function from Tm to Ty. For Razor, this is not the case because after such a reduction, the ite operation becomes partial. The resulting theory is called an essentially algebraic theory (EAT). EATs are another extension of algebraic theories. GATs and EATs are equivalent in the sense that for each GAT, there is an EAT with an equivalent category of models, and for each EAT, there is a GAT with an equivalent category of models.

are elements of the sets (metatheoretic terms):

$$\begin{aligned}
\mathsf{Ty} &:= \mathsf{Set}\\
\mathsf{Tm}\,A &:= A\\
\mathsf{Bool} &:= \{\mathsf{tt}, \mathsf{ff}\}\\
\mathsf{Nat} &: \mathbb{N}\\
\mathsf{true} &:= \mathsf{tt}\\
\mathsf{false} &:= \mathsf{ff}\\
\mathsf{ite}\,b\,t\,f &:= \mathsf{match}\,b\,\{\mathsf{tt} \mapsto t; \mathsf{ff} \mapsto f\}\\
\mathsf{num}\,n &:= n\\
u + v &:= u + v\\
\mathsf{isZero}\,u &:= \mathsf{match}\,u\,\{0 \mapsto \mathsf{tt}; (1+n) \mapsto \mathsf{ff}\}
\end{aligned}$$

We do not write metatheoretic universe levels, hence informally $\mathsf{Set} : \mathsf{Set}$, but formally types in the standard model are a large metatheoretic type (they are in Set_1). In the definition of ite and isZero we used pattern matching on elements of metatheoretic booleans $\{\mathsf{tt}, \mathsf{ff}\}$. In the standard model, all equations hold by reflexivity, as they hold definitionally in the metatheory.

Exercise 14. *If in any model,* $\mathsf{true} = \mathsf{false}$*, then for any* A *and* $u, v : \mathsf{Tm}\,A$*, we have* $u = v$*.*

Exercise 15. *If in any model,* $\mathsf{num}\,0 = \mathsf{num}\,1$*, then for any* A *and* $u, v : \mathsf{Tm}\,A$*, we have* $u = v$*.*

Exercise 16. *There is a model where* $\mathsf{num}\,1 = \mathsf{num}\,2$*, but* $\mathsf{true} \neq \mathsf{false}$*.*

Exercise 17. *There is a model where* $\mathsf{Tm}\,\mathsf{Bool}$ *has three elements (nonstandard model).*

A morphism $M \to N$ contains two functions, one for types and one for terms. The latter refers to the former:

$$\begin{aligned}
\mathsf{Ty} &: \mathsf{Ty}_M \to \mathsf{Ty}_N\\
\mathsf{Tm} &: \mathsf{Tm}_M\,A_M \to \mathsf{Tm}_N\,(\mathsf{Ty}\,A_M)\\
\mathsf{Bool} &: \mathsf{Ty}\,\mathsf{Bool}_M = \mathsf{Bool}_N\\
\mathsf{Nat} &: \mathsf{Ty}\,\mathsf{Nat}_M = \mathsf{Nat}_N\\
\mathsf{true} &: \mathsf{Tm}\,\mathsf{true}_M = \mathsf{true}_N\\
\mathsf{false} &: \mathsf{Tm}\,\mathsf{false}_M = \mathsf{false}_N\\
\mathsf{ite} &: (b_M : \mathsf{Tm}_M\,\mathsf{Bool}_M)(t_M\,f_M : \mathsf{Tm}_M\,A) \to \mathsf{Tm}\,(\mathsf{ite}_M\,b_M\,t_M\,f_M) = \mathsf{ite}_N\,(\mathsf{Tm}\,b_M)\,(\mathsf{Tm}\,t_M)\,(\mathsf{Tm}\,f_M)\\
\mathsf{num} &: (n : \mathbb{N}) \to \mathsf{Tm}\,(\mathsf{num}_M\,n) = \mathsf{num}_N\,n\\
- + - &: (u_M\,v_M : \mathsf{Tm}_M\,\mathsf{Nat}_M) \to \mathsf{Tm}\,(u_M +_M v_M) = \mathsf{Tm}\,u_M +_N \mathsf{Tm}\,v_M\\
\mathsf{isZero} &: (u_M : \mathsf{Tm}_M\,\mathsf{Nat}_M) \to \mathsf{Tm}\,(\mathsf{isZero}_M\,u_M) = \mathsf{isZero}_N\,(\mathsf{Tm}\,u_M)
\end{aligned}$$

The equality true depends on the equality Bool as its left hand side is in $\mathsf{Tm}_N\,(\mathsf{Ty}\,\mathsf{Bool}_M)$, while the right hand side is in $\mathsf{Tm}_N\,\mathsf{Bool}_N$. The situation is similar for other term equations.

Theorem 18. *Razor is logically inconsistent, that is, every sort has an element.*

Proof. Ty_I has an element Bool_I, $\mathsf{Tm}_\mathsf{I}\,\mathsf{Bool}_\mathsf{I}$ has an element true_I, $\mathsf{Tm}_\mathsf{I}\,Nat_\mathsf{I}$ has an element $\mathsf{num}\,0$. □

Theorem 19. *Razor is equationally consistent, that is, not all terms are equal, that is, there is a type* A_I *and terms* $a, a' : \mathsf{Tm}\,A_\mathsf{I}$ *such that* $a \neq a'$*.*

Proof. We choose $A_\mathsf{I} := \mathsf{Bool}_\mathsf{I}$ and $a := \mathsf{true}_\mathsf{I}$ and $a' := \mathsf{false}_\mathsf{I}$, then assuming $a = a'$, their interpretations into the standard models are also equal, hence $\mathsf{tt} = \mathsf{ff}$. □

Iteration into the standard model can be called normalisation (recall that this is a morphism from I to the standard model). Normal forms are given by the Ty component of iteration which we rename to Nf, and we omit the names of the equations; this looks like a pattern-matching definition:

$$\begin{aligned}
&\mathsf{Nf} : \mathsf{Ty}_\mathsf{I} \to \mathsf{Set}\\
&\mathsf{Nf}\,\mathsf{Bool}_\mathsf{I} = \{\mathsf{tt}, \mathsf{ff}\}\\
&\mathsf{Nf}\,\mathsf{Ty}_\mathsf{I} \quad = \mathbb{N}
\end{aligned}$$

The Tm component of the iteration morphism gives the normalisation function, we rename it to norm, its computation rules are the components of the morphism for the term operators:

$$
\begin{aligned}
&\mathsf{norm} : \mathsf{Tm}_{\mathsf{I}}\, A_{\mathsf{I}} \to \mathsf{Nf}\, A_{\mathsf{I}}\\
&\mathsf{norm}\, \mathsf{true}_{\mathsf{I}} &&= \mathsf{tt}\\
&\mathsf{norm}\, \mathsf{false}_{\mathsf{I}} &&= \mathsf{ff}\\
&\mathsf{norm}\, (\mathsf{ite}_{\mathsf{I}}\, b_{\mathsf{I}}\, t_{\mathsf{I}}\, f_{\mathsf{I}}) &&= \mathsf{match}\, (\mathsf{norm}\, b_{\mathsf{I}})\, \{\mathsf{tt} \mapsto \mathsf{norm}\, t_{\mathsf{I}}; \mathsf{ff} \mapsto \mathsf{norm}\, f_{\mathsf{I}}\}\\
&\mathsf{norm}\, (\mathsf{num}_{\mathsf{I}}\, n) &&= n\\
&\mathsf{norm}\, (u_{\mathsf{I}} +_{\mathsf{I}} v_{\mathsf{I}}) &&= \mathsf{norm}\, u_{\mathsf{I}} + \mathsf{norm}\, v_{\mathsf{I}}\\
&\mathsf{norm}\, (\mathsf{isZero}_{\mathsf{I}}\, u_{\mathsf{I}}) &&= \mathsf{match}\, (\mathsf{norm}\, u_{\mathsf{I}})\, \{0 \mapsto \mathsf{tt}; (1+n) \mapsto \mathsf{ff}\}
\end{aligned}
$$

A dependent model over a model M comprises the following components:

$$
\begin{aligned}
&\mathsf{Ty} &&: \mathsf{Ty}_M \to \mathsf{Set}\\
&\mathsf{Tm} &&: \{A_M : \mathsf{Ty}_M\} \to \mathsf{Ty}\, A_M \to \mathsf{Tm}_M\, A_M \to \mathsf{Set}\\
&\mathsf{Bool} &&: \mathsf{Ty}\, \mathsf{Bool}_M\\
&\mathsf{Nat} &&: \mathsf{Ty}\, \mathsf{Nat}_M\\
&\mathsf{true} &&: \mathsf{Tm}\, \{\mathsf{Bool}_M\}\, \mathsf{Bool}\, \mathsf{true}_M\\
&\mathsf{false} &&: \mathsf{Tm}\, \{\mathsf{Bool}_M\}\, \mathsf{Bool}\, \mathsf{false}_M\\
&\mathsf{ite} &&: \mathsf{Tm}\, \mathsf{Bool}\, b_M \to \mathsf{Tm}\, A\, t_M \to \mathsf{Tm}\, A\, f_M \to \mathsf{Tm}\, A\, (\mathsf{ite}_M\, b_M\, t_M\, f_M)\\
&\mathsf{num} &&: (n : \mathbb{N}) \to \mathsf{Tm}\, \mathsf{Nat}\, (\mathsf{num}_M\, n)\\
&- + - &&: \mathsf{Tm}\, \mathsf{Nat}\, u_M \to \mathsf{Tm}\, \mathsf{Nat}\, v_M \to \mathsf{Tm}\, \mathsf{Nat}\, (u_M +_M v_M)\\
&\mathsf{isZero} &&: \mathsf{Tm}\, \mathsf{Nat}\, u_M \to \mathsf{Tm}\, \mathsf{Bool}\, (\mathsf{isZero}_M\, u_M)\\
&\mathsf{ite}\beta_1 &&: \mathsf{ite}\, \mathsf{true}\, u\, v = u\\
&\mathsf{ite}\beta_2 &&: \mathsf{ite}\, \mathsf{false}\, u\, v = v\\
&+\beta &&: \mathsf{num}\, m + \mathsf{num}\, n = \mathsf{num}\, (m+n)\\
&\mathsf{isZero}\beta_1 &&: \mathsf{isZero}\, (\mathsf{num}\, 0) = \mathsf{true}\\
&\mathsf{isZero}\beta_2 &&: \mathsf{isZero}\, (\mathsf{num}\, (1+n)) = \mathsf{false}
\end{aligned}
$$

Note that the equations in the dependent model depend on the corresponding equations in M. For example, the left hand side of $\mathsf{ite}\beta_1$ is in $\mathsf{Tm}\, \{A_M\}\, A\, (\mathsf{ite}_M\, \mathsf{true}_M\, u_M\, v_M)$, while the right hand side is in $\mathsf{Tm}\, \{A_M\}\, A\, u_M$.

The syntax supports induction, which means that there is the following dependent morphism from the syntax to a dependent model D over it:

$$
\begin{aligned}
&\mathsf{Ty} &&: (A_{\mathsf{I}} : \mathsf{Ty}_{\mathsf{I}}) \to \mathsf{Ty}_D\, A_{\mathsf{I}}\\
&\mathsf{Tm} &&: (a_{\mathsf{I}} : \mathsf{Tm}_{\mathsf{I}}\, A_{\mathsf{I}}) \to \mathsf{Tm}_D\, (\mathsf{Ty}\, A_{\mathsf{I}})\, a_{\mathsf{I}}\\
&\mathsf{Bool} &&: \mathsf{Ty}\, \mathsf{Bool}_{\mathsf{I}} = \mathsf{Bool}_D\\
&\mathsf{Nat} &&: \mathsf{Ty}\, \mathsf{Nat}_{\mathsf{I}} = \mathsf{Nat}_D\\
&\mathsf{true} &&: \mathsf{Tm}\, \mathsf{true}_{\mathsf{I}} = \mathsf{true}_D\\
&\mathsf{false} &&: \mathsf{Tm}\, \mathsf{false}_{\mathsf{I}} = \mathsf{false}_D\\
&\mathsf{ite} &&: (b_{\mathsf{I}} : \mathsf{Tm}_{\mathsf{I}}\, \mathsf{Bool}_{\mathsf{I}})(t_{\mathsf{I}}\, f_{\mathsf{I}} : \mathsf{Tm}_{\mathsf{I}}\, A_{\mathsf{I}}) \to \mathsf{Tm}\, (\mathsf{ite}_{\mathsf{I}}\, b_{\mathsf{I}}\, t_{\mathsf{I}}\, f_{\mathsf{I}}) = \mathsf{ite}_D\, (\mathsf{Tm}\, b_{\mathsf{I}})\, (\mathsf{Tm}\, t_{\mathsf{I}})\, (\mathsf{Tm}\, f_{\mathsf{I}})\\
&\mathsf{num} &&: (n : \mathbb{N}) \to \mathsf{Tm}\, (\mathsf{num}_{\mathsf{I}}\, n) = \mathsf{num}_D\, n\\
&- + - &&: (u_{\mathsf{I}}\, v_{\mathsf{I}} : \mathsf{Tm}_{\mathsf{I}}\, \mathsf{Nat}_{\mathsf{I}}) \to \mathsf{Tm}\, (u_{\mathsf{I}} +_{\mathsf{I}} v_{\mathsf{I}}) = \mathsf{Tm}\, u_{\mathsf{I}} +_D \mathsf{Tm}\, v_{\mathsf{I}}\\
&\mathsf{isZero} &&: (u_{\mathsf{I}} : \mathsf{Tm}_{\mathsf{I}}\, \mathsf{Nat}_{\mathsf{I}}) \to \mathsf{Tm}\, (\mathsf{isZero}_{\mathsf{I}}\, u_{\mathsf{I}}) = \mathsf{isZero}_D\, (\mathsf{Tm}\, u_{\mathsf{I}})
\end{aligned}
$$

We define a dependent model where the term components are trivial (we don't list them):

$$
\begin{aligned}
&\mathsf{Ty}\, A_{\mathsf{I}} &&:= \mathsf{Nf}\, A_{\mathsf{I}} \to \mathsf{Tm}_{\mathsf{I}}\, A_{\mathsf{I}}\\
&\mathsf{Tm}\, A\, a_{\mathsf{I}} &&:= \mathbb{1}\\
&\mathsf{Bool} &&: \underbrace{\mathsf{Nf}\, \mathsf{Bool}_{\mathsf{I}}}_{=\{\mathsf{tt},\mathsf{ff}\}} \to \mathsf{Tm}_{\mathsf{I}}\, \mathsf{Bool}_{\mathsf{I}}\\
&\mathsf{Bool}\, b &&:= \mathsf{match}\, b\, \{\mathsf{tt} \mapsto \mathsf{true}_{\mathsf{I}}; \mathsf{ff} \mapsto \mathsf{false}_{\mathsf{I}}\}\\
&\mathsf{Nat} &&: \underbrace{\mathsf{Nf}\, \mathsf{Nat}_{\mathsf{I}}}_{=\mathbb{N}} \to \mathsf{Tm}_{\mathsf{I}}\, \mathsf{Nat}_{\mathsf{I}}\\
&\mathsf{Nat}\, n &&:= \mathsf{num}_{\mathsf{I}}\, n
\end{aligned}
$$

Induction into this dependent model provides us with a quote function which maps normal forms back into syntactic terms:

$$\begin{aligned}
&\mathsf{quote} : (A_\mathsf{I} : \mathsf{Ty}_\mathsf{I}) \to \mathsf{Nf}\, A_\mathsf{I} \to \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}\\
&\mathsf{quote}\, \mathsf{Bool}_\mathsf{I}\, b = \mathsf{match}\, b\, \{\mathsf{tt} \mapsto \mathsf{true}_\mathsf{I}; \mathsf{ff} \mapsto \mathsf{false}_\mathsf{I}\}\\
&\mathsf{quote}\, \mathsf{Nat}_\mathsf{I}\ \ n = \mathsf{num}_\mathsf{I}\, n
\end{aligned}$$

Completeness of normalisation says that for any $a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}$, $\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, a_\mathsf{I}) = a_\mathsf{I}$. We prove this by constructing another dependent model over I where the Ty component is trivial:

$$\begin{aligned}
&\mathsf{Ty}\, A_\mathsf{I} && := \mathbb{1}\\
&\mathsf{Tm}\, \{A_\mathsf{I}\} * a_\mathsf{I} && := (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, a_\mathsf{I}) = a_\mathsf{I})\\
&\mathsf{true} && : \mathsf{quote}\, \mathsf{Bool}_\mathsf{I}\, (\mathsf{norm}\, \mathsf{true}_\mathsf{I}) = \mathsf{quote}\, \mathsf{Bool}_\mathsf{I}\, \mathsf{tt} = \mathsf{match}\, \mathsf{tt}\, \{\mathsf{tt} \mapsto \mathsf{true}_\mathsf{I}; \mathsf{ff} \mapsto \mathsf{false}_\mathsf{I}\} = \mathsf{true}_\mathsf{I}\\
&\mathsf{false} && : \mathsf{quote}\, \mathsf{Bool}_\mathsf{I}\, (\mathsf{norm}\, \mathsf{false}_\mathsf{I}) = \mathsf{quote}\, \mathsf{Bool}_\mathsf{I}\, \mathsf{ff} = \mathsf{match}\, \mathsf{ff}\, \{\mathsf{tt} \mapsto \mathsf{true}_\mathsf{I}; \mathsf{ff} \mapsto \mathsf{false}_\mathsf{I}\} = \mathsf{false}_\mathsf{I}\\
&\mathsf{ite}\, (e_b : \mathsf{quote}\, \mathsf{Bool}_\mathsf{I}\, (\mathsf{norm}\, b_\mathsf{I}) = b_\mathsf{I})(e_t : \mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, t_\mathsf{I}) = t_\mathsf{I})(e_f : \mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, f_\mathsf{I}) = f_\mathsf{I})\\
&\qquad : \mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, (\mathsf{ite}_\mathsf{I}\, b_\mathsf{I}\, t_\mathsf{I}\, f_\mathsf{I}))\\
&\qquad\quad \mathsf{quote}\, A_\mathsf{I}\, (\mathsf{match}\, (\mathsf{norm}\, b_\mathsf{I})\, \{\mathsf{tt} \mapsto \mathsf{norm}\, t_\mathsf{I}; \mathsf{ff} \mapsto \mathsf{norm}\, f_\mathsf{I}\}) && =(\mathsf{norm}\, b_\mathsf{I} = \mathsf{tt})\\
&\qquad\quad \mathsf{quote}\, A_\mathsf{I}\, (\mathsf{match}\, \mathsf{tt}\, \{\mathsf{tt} \mapsto \mathsf{norm}\, t_\mathsf{I}; \mathsf{ff} \mapsto \mathsf{norm}\, f_\mathsf{I}) && =\\
&\qquad\quad \mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, t_\mathsf{I}) && =(\mathsf{ite}\beta_{1\mathsf{I}})\\
&\qquad\quad \mathsf{ite}_\mathsf{I}\, \mathsf{true}_\mathsf{I}\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, t_\mathsf{I}))\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, f_\mathsf{I})) && =\\
&\qquad\quad \mathsf{ite}_\mathsf{I}\, (\mathsf{match}\, \mathsf{tt}\, \{\mathsf{tt} \mapsto \mathsf{true}_\mathsf{I}; \mathsf{ff} \mapsto \mathsf{false}_\mathsf{I}\})\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, t_\mathsf{I}))\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, f_\mathsf{I})) && =(\mathsf{norm}\, b_\mathsf{I} = \mathsf{tt})\\
&\qquad\quad \mathsf{ite}_\mathsf{I}\, (\mathsf{match}\, (\mathsf{norm}\, b_\mathsf{I})\, \{\mathsf{tt} \mapsto \mathsf{true}_\mathsf{I}; \mathsf{ff} \mapsto \mathsf{false}_\mathsf{I}\})\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, t_\mathsf{I}))\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, f_\mathsf{I})) && =\\
&\qquad\quad \mathsf{ite}_\mathsf{I}\, (\mathsf{quote}\, \mathsf{Bool}_\mathsf{I}\, (\mathsf{norm}\, b_\mathsf{I}))\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, t_\mathsf{I}))\, (\mathsf{quote}\, A_\mathsf{I}\, (\mathsf{norm}\, f_\mathsf{I})) && =(e_b, e_t, e_f)\\
&\qquad\quad \mathsf{ite}_\mathsf{I}\, b_\mathsf{I}\, t_\mathsf{I}\, f_\mathsf{I}
\end{aligned}$$

As terms in this dependent model are equations between elements of $\mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}$ for some A_I, we don't have to provide the equation components ($\mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}$ is a set in the sense of homotopy type theory, it has uniqueness of identity proofs).

Exercise 20. *Finish defining the dependent model: write the case* $\mathsf{norm}\, b_\mathsf{I} = \mathsf{ff}$ *for* ite *and define the components* num, $- + -$, isZero.

Exercise 21. *Merge* quote *and completeness, so that they both are parts of induction into a single dependent model.*

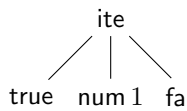
Exercise 22. *Prove* stability*: that is, given a normal form, if we quote it and then normalise it, we get back the same normal form. In summary, normalisation with completeness and stability says that there is an isomorphism between the sets* $\mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}$ *and* $\mathsf{Nf}\, A_\mathsf{I}$.

In general, *normal forms* are a complete representation of the syntax where equality is easily decidable (for example, they are given by an inductive definition, that is, the syntax of a GAT without equations). Decidable equality of normal forms implies decidable equality of terms by completeness: given $a_\mathsf{I}, a_\mathsf{I}' : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}$, if $e : \mathsf{norm}\, a_\mathsf{I} = \mathsf{norm}\, a_\mathsf{I}'$ then

$$a_\mathsf{I} \overset{\text{completeness}}{=} \mathsf{quote}\, (\mathsf{norm}\, a_\mathsf{I}) \overset{e}{=} \mathsf{quote}\, (\mathsf{norm}\, a_\mathsf{I}') \overset{\text{completeness}}{=} a_\mathsf{I}',$$

and given $\mathsf{norm}\, a_\mathsf{I} \neq \mathsf{norm}\, a_\mathsf{I}'$, from $a_\mathsf{I} = a_\mathsf{I}'$, by congruence we get $\mathsf{norm}\, a_\mathsf{I} = \mathsf{norm}\, a_\mathsf{I}'$. Stability is not needed for decidability of equality of the syntax, but it is useful to obtain a new induction principle for the syntax: one can prove things about the syntax by induction on normal forms.

Notation 23 (Derivation rules)**.** *Elements of the syntax can be depicted by syntax trees, but then the well-typedness of the tree is not immediately visible, it is not enforced by the drawing that the tree does not make sense. Instead, we first describe the notion of model using derivation rule notation which means that we uncurry all the operators, give names to the arguments and in each operator, we replace the remaining* $\to$ *with a horizontal line:*



$$\frac{}{\mathsf{Ty} : \mathsf{Set}} \qquad \frac{A : \mathsf{Ty}}{\mathsf{Tm}\, A : \mathsf{Set}} \qquad \frac{}{\mathsf{Bool} : \mathsf{Ty}} \qquad \frac{}{\mathsf{Nat} : \mathsf{Ty}} \qquad \frac{}{\mathsf{true} : \mathsf{Tm}\, \mathsf{Bool}} \qquad \frac{}{\mathsf{false} : \mathsf{Tm}\, \mathsf{Bool}}$$

$$\frac{b : \mathsf{Tm}\,\mathsf{Bool} \qquad t : \mathsf{Tm}\,A \qquad f : \mathsf{Tm}\,A}{\mathsf{ite}\,b\,t\,f : \mathsf{Tm}\,A} \qquad \frac{n : \mathbb{N}}{\mathsf{num}\,n : \mathsf{Tm}\,\mathsf{Nat}} \qquad \frac{u : \mathsf{Tm}\,\mathsf{Nat} \qquad v : \mathsf{Tm}\,\mathsf{Nat}}{u + v : \mathsf{Tm}\,\mathsf{Nat}} \qquad \frac{u : \mathsf{Tm}\,\mathsf{Bool}}{\mathsf{isZero}\,u : \mathsf{Tm}\,\mathsf{Bool}}$$

$$\frac{}{\mathsf{ite}\beta_1 : \mathsf{ite}\,\mathsf{true}\,u\,v = u} \qquad \frac{}{\mathsf{ite}\beta_2 : \mathsf{ite}\,\mathsf{false}\,u\,v = v} \qquad \frac{}{+\beta : \mathsf{num}\,m + \mathsf{num}\,n = \mathsf{num}\,(m+n)}$$

$$\frac{}{\mathsf{isZero}\beta_1 : \mathsf{isZero}\,(\mathsf{num}\,0) = \mathsf{true}} \qquad \frac{}{\mathsf{isZero}\beta_2 : \mathsf{isZero}\,(\mathsf{num}\,(1+n)) = \mathsf{false}}$$

Just as in the algebraic notation we have implicit arguments, for example $A : \mathsf{Ty}$ *is not written above the line for* ite*, but we assume it is (implicitly) there.*

Notation 24 (Derivation trees)**.** *Derivation trees are like upside-down syntax trees where at the nodes the full term and its set are both written; the children are subterms just as before. Example:*

$$\frac{\dfrac{\dfrac{}{\mathsf{num}\,1 : \mathsf{Tm}\,\mathsf{Nat}}}{\mathsf{isZero}\,(\mathsf{num}\,1) : \mathsf{Tm}\,\mathsf{Bool}} \qquad \frac{}{\mathsf{true} : \mathsf{Tm}\,\mathsf{Bool}} \qquad \frac{}{\mathsf{false} : \mathsf{Tm}\,\mathsf{Bool}}}{\mathsf{ite}\,(\mathsf{isZero}\,(\mathsf{num}\,1))\,\mathsf{true}\,\mathsf{false} : \mathsf{Tm}\,\mathsf{Bool}}$$

Each horizontal line in the derivation tree is a special case of one of the derivation rules.

It is clear that the following tree cannot be finished:

$$\frac{\dfrac{}{\mathsf{true} : \mathsf{Tm}\,\mathsf{Bool}} \qquad \dfrac{}{\mathsf{num}\,1 : \mathsf{Tm}\,\mathsf{Nat}} \qquad \dfrac{????}{\mathsf{false} : \mathsf{Tm}\,\mathsf{Nat}}}{\mathsf{ite}\,\mathsf{true}\,(\mathsf{num}\,1)\,\mathsf{false} : \mathsf{Tm}\,\mathsf{Nat}}$$

Exercise 25. *Show that induction is equivalent to initiality.*

2.4 Simply typed combinator calculus

Using Moses Schönfinkel's K and S combinators [Sch24], higher order functions can be described without variables.[2]

Definition 26 (STCC)**.** *A model comprises the following components:*

$$\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\iota & : \mathsf{Ty} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
- \cdot - & : \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\,A \Rightarrow \mathsf{Tm}\,B \\
\mathsf{K} & : \mathsf{Tm}\,(A \Rightarrow B \Rightarrow A) \\
\mathsf{S} & : \mathsf{Tm}\,((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C) \\
\mathsf{K}\beta & : \mathsf{K} \cdot a \cdot b = a \\
\mathsf{S}\beta & : \mathsf{S} \cdot f \cdot g \cdot a = f \cdot a \cdot (g \cdot a)
\end{array}$$

We added one base type so that the syntax is not empty. Note that $- \cdot -$ and K have two, S has three implicit arguments. $- \Rightarrow -$ is right-associative, application $- \cdot -$ is left-associative.

Exercise 27. *Derive the following combinators, their expected definitional behaviour is specified on the right:*

$$\begin{array}{ll}
\mathsf{I} : \mathsf{Tm}\,(A \Rightarrow A) & \mathsf{I} \cdot a = a \\
\mathsf{B} : \mathsf{Tm}\,((B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C) & \mathsf{B} \cdot f \cdot g \cdot a = f \cdot (g \cdot a) \\
\mathsf{C} : \mathsf{Tm}\,((A \Rightarrow B \Rightarrow C) \Rightarrow B \Rightarrow A \Rightarrow C) & \mathsf{C} \cdot f \cdot b \cdot a = f \cdot a \cdot b
\end{array}$$

Exercise 28. *Define the standard model where* $\mathsf{Ty} = \mathsf{Set}$*,* $\mathsf{Tm}\,A = A$*. The interpretation of* ι *is a parameter of the standard model.*

Theorem 29. *STCC is logically consistent, that is, there is a type* A_I *such that* $\mathsf{Tm}_\mathsf{I}\,A_\mathsf{I}$ *is empty.*

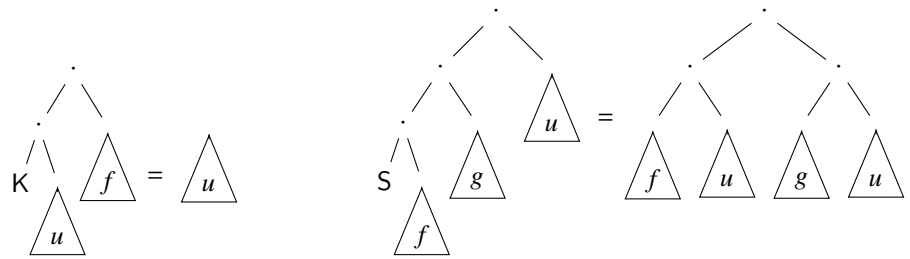
Proof. Interpretating $\mathsf{Tm}_\mathsf{I}\,\iota_\mathsf{I}$ into the standard model with $\iota = \mathbb{0}$ gives an element of $\mathbb{0}$. □

Theorem 30. *STCC is equationally consistent, that is, not all terms are equal, that is, there is a type* A_I *and terms* $a, a' : \mathsf{Tm}\,A_\mathsf{I}$ *such that* $a \neq a'$*.*

[2] It is remarkable that Schönfinkel's untyped combinator calculus is still the simplest Turing complete language, even though it was the first such. Note that in contrast with STCC, untyped combinator calculus does not have normalisation.

Proof. We choose $A_\mathsf{I} := (\iota_\mathsf{I} \Rightarrow_\mathsf{I} \iota_\mathsf{I} \Rightarrow_\mathsf{I} \iota_\mathsf{I})$ and $a := \mathsf{lam}_\mathsf{I}\, \lambda x_\mathsf{I}.\, \mathsf{lam}\, \lambda y_\mathsf{I}.\, x_\mathsf{I}$ and $a' := \mathsf{lam}_\mathsf{I}\, \lambda x_\mathsf{I}.\, \mathsf{lam}\, \lambda y_\mathsf{I}.\, y_\mathsf{I}$. We interpret both a and a' into the standard model with $\iota = 2$, then asssuming $a = a'$, their interpetations are also equal, so we get $(\lambda x\, y.\, x) = (\lambda x\, y.\, y)$, and from this we get $\mathsf{tt} = (\lambda x\, y.\, x)\, \mathsf{tt}\, \mathsf{ff} = (\lambda x\, y.\, y)\, \mathsf{tt}\, \mathsf{ff} = \mathsf{ff}$. □

In the syntax of STCC, types are binary trees (nodes are $\Rightarrow$, leaves are ι). If we ignore types, terms are boolean-labelled binary trees quotiented by the following equations:



If we don't ignore types, then we can only build a node if the left hand subtree is in $\mathsf{Tm}\,(A \Rightarrow B)$ for some A, B, and the right hand subtree is in $\mathsf{Tm}\,A$ for the same A. Again, this is encforced by derivation trees, for example:

$$\dfrac{\dfrac{\overline{\mathsf{S} : \mathsf{Tm}\,((\iota \Rightarrow (\iota \Rightarrow \iota) \Rightarrow \iota) \Rightarrow ((\iota \Rightarrow \iota \Rightarrow \iota) \Rightarrow \iota \Rightarrow \iota))} \quad \overline{\mathsf{K} : \mathsf{Tm}\,(\iota \Rightarrow (\iota \Rightarrow \iota) \Rightarrow \iota)}}{\mathsf{S} \cdot \mathsf{K} : \mathsf{Tm}\,((\iota \Rightarrow \iota \Rightarrow \iota) \Rightarrow \iota \Rightarrow \iota)} \quad \overline{\mathsf{K} : \mathsf{Tm}\,(\iota \Rightarrow \iota \Rightarrow \iota)}}{\mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K} : \mathsf{Tm}\,(\iota \Rightarrow \iota)}$$

Note that we didn't write implicit arguments. The two Ks on the right hand side of the tree are different: there is $\mathsf{K}\,\{\iota\}\,\{\iota\} : \mathsf{Tm}\,(\iota \Rightarrow \iota \Rightarrow \iota)$ and $\mathsf{K}\,\{\iota\}\,\{\iota \Rightarrow \iota\} : \mathsf{Tm}\,(\iota \Rightarrow (\iota \Rightarrow \iota) \Rightarrow \iota)$.

Exercise 31. *What is a nice minimal presyntax (untyped AST description) for STCC where types can be always inferred? That is, which implicit arguments have to be written explicitly?*

What are the normal forms for the syntax of STCC? We give an inductive description of the expressions where $\mathsf{K}\beta$ and $\mathsf{S}\beta$ cannot be applied because K and S does not have enough arguments:

Definition 32 (STCC normal forms)**.** *A normal form model comprises the following:*

$$\begin{aligned}
&\mathsf{Nf} : (A_\mathsf{I} : \mathsf{Ty}_\mathsf{I}) \to \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I} \to \mathsf{Set}\\
&\mathsf{K}_0 : \mathsf{Nf}\,(A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I} \Rightarrow_\mathsf{I} A_\mathsf{I})\,\mathsf{K}_\mathsf{I}\\
&\mathsf{K}_1 : \mathsf{Nf}\,A_\mathsf{I}\,a_\mathsf{I} \to \mathsf{Nf}\,(B_\mathsf{I} \Rightarrow_\mathsf{I} A_\mathsf{I})\,(\mathsf{K}_\mathsf{I} \cdot_\mathsf{I} a_\mathsf{I})\\
&\mathsf{S}_0 : \mathsf{Nf}\,((A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I} \Rightarrow_\mathsf{I} C_\mathsf{I}) \Rightarrow_\mathsf{I} (A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I}) \Rightarrow_\mathsf{I} A_\mathsf{I} \Rightarrow_\mathsf{I} C_\mathsf{I})\,\mathsf{S}_\mathsf{I}\\
&\mathsf{S}_1 : \mathsf{Nf}\,(A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I} \Rightarrow_\mathsf{I} C_\mathsf{I})\,f_\mathsf{I} \to \mathsf{Nf}\,((A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I}) \Rightarrow_\mathsf{I} A_\mathsf{I} \Rightarrow_\mathsf{I} C_\mathsf{I})\,(\mathsf{S}_\mathsf{I} \cdot_\mathsf{I} f_\mathsf{I})\\
&\mathsf{S}_2 : \mathsf{Nf}\,(A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I} \Rightarrow_\mathsf{I} C_\mathsf{I})\,f_\mathsf{I} \to \mathsf{Nf}\,(A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I})\,g_\mathsf{I} \to \mathsf{Nf}\,(A_\mathsf{I} \Rightarrow_\mathsf{I} C_\mathsf{I})\,(\mathsf{S}_\mathsf{I} \cdot_\mathsf{I} f_\mathsf{I} \cdot_\mathsf{I} g_\mathsf{I})
\end{aligned}$$

These normal forms are indexed not only by their syntactic type, but also by the term they correspond to (the result of quote).

Exercise 33. *Show that equality is decidable in the syntax of normal forms (this relies on equality of syntactic types). The nicest way is to do double-induction on normal forms to prove decidability of equality of the total space of normal forms, that is, the set* $(A_\mathsf{I} : \mathsf{Ty}_\mathsf{I}) \times (a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}) \times \mathsf{Nf}_\mathsf{I}\, A_\mathsf{I}\, a_\mathsf{I}$.

A dependent model over I comprises the following components:

$$\begin{aligned}
&\mathsf{Ty} && : \mathsf{Ty}_\mathsf{I} \to \mathsf{Set}\\
&\mathsf{Tm} && : \mathsf{Ty}\,A_\mathsf{I} \to \mathsf{Tm}_\mathsf{I}\,A_\mathsf{I} \to \mathsf{Set}\\
&\iota && : \mathsf{Ty}\,\iota_\mathsf{I}\\
&- \Rightarrow - && : \mathsf{Ty}\,A_\mathsf{I} \to \mathsf{Ty}\,B_\mathsf{I} \to \mathsf{Ty}\,(A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I})\\
&- \cdot - && : \mathsf{Tm}\,(A \Rightarrow B)\,f_\mathsf{I} \to \mathsf{Tm}\,A\,a_\mathsf{I} \Rightarrow \mathsf{Tm}\,B\,(f_\mathsf{I} \cdot_\mathsf{I} a_\mathsf{I})\\
&\mathsf{K} && : \mathsf{Tm}\,(A \Rightarrow B \Rightarrow A)\,\mathsf{K}_\mathsf{I}\\
&\mathsf{S} && : \mathsf{Tm}\,((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C)\,S_\mathsf{I}\\
&\mathsf{K}\beta && : \mathsf{K} \cdot a \cdot b = a\\
&\mathsf{S}\beta && : \mathsf{S} \cdot f \cdot g \cdot a = f \cdot a \cdot (g \cdot a)
\end{aligned}$$

Given a dependent model D over I, a section comprises the following:

$$\begin{aligned}
\mathsf{Ty} &: (A_\mathsf{I} : \mathsf{Ty}_\mathsf{I}) \to \mathsf{Ty}_D\, A_\mathsf{I}\\
\mathsf{Tm} &: (a_\mathsf{I} : \mathsf{Tm}_A\, A_\mathsf{I}) \to \mathsf{Tm}_D\, (\mathsf{Ty}\, A_\mathsf{I})\, a_\mathsf{I}\\
\iota &: \mathsf{Ty}\, \iota_\mathsf{I} = \iota_D\\
- \Rightarrow - &: (A\, B : \mathsf{Ty}_\mathsf{I}) \to \mathsf{Ty}\, (A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I}) = \mathsf{Ty}\, A_\mathsf{I} \Rightarrow_D \mathsf{Ty}\, B_\mathsf{I}\\
- \cdot - &: (f_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, (A_\mathsf{I} \Rightarrow_\mathsf{I} B_\mathsf{I}))(a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}) \Rightarrow \mathsf{Tm}\, (f_\mathsf{I} \cdot_\mathsf{I} a_\mathsf{I}) = \mathsf{Tm}\, f_\mathsf{I} \cdot_D \mathsf{Tm}\, a_\mathsf{I}\\
\mathsf{K} &: \mathsf{Tm}\, \mathsf{K}_\mathsf{I} = K_D\\
\mathsf{S} &: \mathsf{Tm}\ S_\mathsf{I} = S_D
\end{aligned}$$

We use Tait's method [Tai67] (also called logical predicate or reducibility method) to prove normalisation for STCC. We define a dependent model over I where types are a proof-relevant predicate over terms together with a reify function, and terms are witnesses of the predicate. The predicate for ι is constant false. The predicate for a function says that if the predicate holds for an input, it also holds for the output, and the function is in normal form.

$$\begin{aligned}
&\mathsf{Ty}\, A_\mathsf{I} && := (P_A : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I} \to \mathsf{Set}) \times \left(\{a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}\} \to P_A\, a_\mathsf{I} \to \mathsf{Nf}\, A_\mathsf{I}\, a_\mathsf{I}\right)\\
&\mathsf{Tm}\, (P_A, r_A)\, a_\mathsf{I} && := P_A\, a_\mathsf{I}\\
&\iota && := (\lambda_.\, \mathbb{0}, \lambda b.\, \mathsf{match}\, b\, \{\})\\
&(P_A, r_A) \Rightarrow (P_B, r_B) && := \left(\lambda f_\mathsf{I}.\, \left(\{a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}\} \to P_A\, a_\mathsf{I} \to P_B\, (f_\mathsf{I} \cdot_\mathsf{I} a_\mathsf{I})\right) \times \mathsf{Nf}\, A_\mathsf{I}\, f_\mathsf{I}, \lambda(p_f, n_f).n_f\right)\\
&(p_f, n_f) \cdot p_a && := p_f\, p_a\\
&\mathsf{K}\, \{P_A, r_A\}\{P_B, r_B\} && := \left(\lambda p_a.\, (\lambda p_b.\, p_a, \mathsf{K}_1\, (r_A\, a_\mathsf{I})), \mathsf{K}_0\right)\\
&\mathsf{S}\, \{P_A, r_A\}\{P_B, r_B\}\{P_C, r_C\} && := \left(\lambda(p_f, n_f).\, \left(\lambda(p_g, n_g).\, (\lambda p_a.\, (p_f\, p_a).1\, (p_g\, p_a), \mathsf{S}_2\, n_f\, n_g), \mathsf{S}_1\, n_f\right), \mathsf{S}_0\right)
\end{aligned}$$

Equations $\mathsf{K}\beta$ and $\mathsf{S}\beta$ hold by definition. When defining K and S, we implicitly made use of $\mathsf{K}\beta_\mathsf{I}$ and $\mathsf{S}\beta_\mathsf{I}$, respectively. By induction into this model we obtain the predicate for each type, the reify function for each type and the witness of the predicate for each term:

$$\begin{aligned}
P &: (A_\mathsf{I} : \mathsf{Ty}_\mathsf{I}) \to \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I} \to \mathsf{Set}\\
r &: (A_\mathsf{I} : \mathsf{Ty}_\mathsf{I})\{a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}\} \to P\, A_\mathsf{I}\, a_\mathsf{I} \to \mathsf{Nf}\, A_\mathsf{I}\, a_\mathsf{I}\\
p &: (a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}) \to P\, A_\mathsf{I}\, a_\mathsf{I}
\end{aligned}$$

We put these together to obtain normalisation:

$$\mathsf{norm}\, (a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\, A_\mathsf{I}) := r\, A_\mathsf{I}\, a_\mathsf{I}\, (p\, a_\mathsf{I})$$

Note that in the normalisation dependent model, we only assumed an STCC normal form model, we did not rely on it being syntax. However, if it is the syntax of STCC normal forms, we can do the following:

Exercise 34. *Via normalisation, show decidability of equality for syntactic STCC terms.*

We were quite clever when coming up with the notion of normal forms. We don't need to be clever:

Exercise 35. *Show that the syntax of STCC without equations also suffices for normalisation and decidability of equality of (quotiented) STCC syntax.*

Exercise 36. *Intuitionistic logic with only implication as a connective is the same as STCC where we don't care about equations, a model is the following:*

$$\begin{aligned}
\mathsf{For} &: \mathsf{Set}\\
\mathsf{Pf} &: \mathsf{Pf} \to \mathsf{Set}\\
\mathsf{irr} &: (a\, a' : \mathsf{Pf}\, A) \to a = a'\\
\iota &: \mathsf{For}\\
- \supset - &: \mathsf{For} \to \mathsf{For} \to \mathsf{For}\\
- \cdot - &: \mathsf{Pf}\, (A \supset B) \to \mathsf{Pf}\, A \Rightarrow \mathsf{Pf}\, B\\
\mathsf{K} &: \mathsf{Pf}\, (A \supset B \supset A)\\
\mathsf{S} &: \mathsf{Pf}\, ((A \supset B \supset C) \supset (A \supset B) \supset A \supset C)
\end{aligned}$$

Define normal forms as a proof-irrelevant predicate on Pf_I and show normalisation. Analysing the shape of normal forms, show that Peirce's law is not provable.

Food for thought 37. *Extend this logic with true, false, conjunction, disjunction, and show via normalisation that the law of excluded middle is not derivable.*

2.5 Other GATs

See [KKA19] for an algorithm which derives the notion of model, morphism, dependent model, dependent morphism from any GAT signature. Models and morphisms organise into a category (morphisms can be composed, composition is associative, etc), dependent models and dependent morphisms form a family structure over this category, together they produce a category with family (CwF). CwF is a model of type theory without any type formers. For any GAT, this CwF supports Σ, $\top$, extensional equality types, booleans and quotient types (the category is finitely complete and cocomplete). We will discuss CwFs in Section 4.

Classes of inductive sets (meta-types) and classes of algebraic theories roughly correspond to each other. Inductive sets are initial models (syntaxes) for some algebraic theory, algebraic theories determine classes of algebras for some inductive types. The correspondence:

single sorted algebraic theory without equations	simple inductive type (W-type)
multi sorted algebraic theory without equations	mutually defined inductive types (indexed W-type)
single sorted algebraic theory	quotient inductive type (QIT, QW-type)
generalised algebraic theory without equations	inductive inductive type (IIT)
generalised algebraic theory (GAT)	quotient inductive inductive type (QIIT)
higher generalised algebraic theory	higher inductive inductive type (HIIT)

However algebraic theories are more fine-grained: for example, as inductive types $\mathsf{List}\,A$ and the free monoid over A are equivalent (Exercise 38), but as algebraic theories, the latter has a larger category of models. The situation is similar between the lambda calculus and combinatory calculus [AKSV23]. Mutual inductive types can be reduced to indexed inductive types [KvR20] which can again be reduced to simple inductive types [Kap19], but this is not the case for the corresponding classes of algebraic theories.

Exercise 38. *A model of monoid over A is a model of monoid with an extra operation $A \to \mathsf{C}$. For an $A : \mathsf{Set}$, a A-nil-cons model comprises the following:*

$$\begin{aligned}\mathsf{L} &: \mathsf{Set}\\ \mathsf{nil} &: \mathsf{L}\\ \mathsf{cons} &: A \to \mathsf{L} \to \mathsf{L}\end{aligned}$$

Show that the syntax of A-nil-cons (also called $\mathsf{List}\,A$) gives rise to a syntax of monoid over A (the syntax of monoid over A is also called the free monoid over A). Show normalisation for monoids over A where normal forms are $\mathsf{List}\,A$.

Definition 39 (Preorder)**.**

$$\begin{aligned}\mathsf{Ob} &: \mathsf{Set}\\ \mathsf{Mor} &: \mathsf{Ob} \to \mathsf{Ob} \to \mathsf{Set}\\ - \circ - &: \mathsf{Mor}\,J\,I \to \mathsf{Mor}\,K\,J \to \mathsf{Mor}\,K\,I\\ \mathsf{id} &: \mathsf{Mor}\,I\,I\\ \mathsf{irr} &: (f\,g : \mathsf{Mor}\,J\,I) \to f = g\end{aligned}$$

Exercise 40. *Category is a GAT. Define morphisms, dependent models, dependent morphisms. What is the simplest definition of syntax? Now do the same for cartesian closed category.*

Normal forms and normalisation cannot be defined for arbitrary GATs. For example, in the syntax of untyped combinator calculus, equality is undecidable, hence it does not have normalisation.

3 Derivability in SOGATs

Second-order GATs (SOGATs) allow second-order operations. Second-order operations are also called binders. In this section, we define several languages with binders as SOGATs, and show how to define derivable operations and prove derivable equations in second-order models. The phrase "second-order" is a negative qualifier: second-order algebraic theories are not really algebraic (this is like illiberal democracy or people's democracy, which are not really democracies). For example, there is no good notion of morphism between second-order models. However, for derivability, second-order models are good enough. We will treat this problem in Section 4.

3.1 Simply typed lambda calculus

Definition 41 (Simply typed lambda calculus, STLC)**.** *A second-order model of STLC comprises the following components:*

$$\begin{aligned}
&\mathsf{Ty} && : \mathsf{Set}\\
&\mathsf{Tm} && : \mathsf{Ty} \to \mathsf{Set}\\
&\iota && : \mathsf{Ty}\\
&- \Rightarrow - && : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}\\
&\mathsf{lam} && : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \to \mathsf{Tm}\,(A \Rightarrow B)\\
&- \cdot - && : \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\, A \to \mathsf{Tm}\, B\\
&\beta && : \mathsf{lam}\, b \cdot a = b\, a\\
&\eta && : f = \mathsf{lam}\, \lambda x.\, f \cdot x
\end{aligned}$$

The operators lam and $- \cdot -$ take *implicit arguments*, A and B. E.g. the explicit type of lam is $\{A : \mathsf{Ty}\}\{B : \mathsf{Ty}\} \to (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \to \mathsf{Tm}\,(A \Rightarrow B)$. All the arguments of the equations β and η are implicit. E.g. the explicit type of β is $\{A : \mathsf{Ty}\}\{B : \mathsf{Ty}\}\{b : \mathsf{Tm}\, A \to \mathsf{Tm}\, B\}\{a : \mathsf{Tm}\, A\} \to (- \cdot -)\,\{A\}\{B\}\,(\mathsf{lam}\,\{A\}\,\{B\}\, b)\, a = b\, a$. Note that in the equation η, f has to be in $\mathsf{Tm}\,(A \Rightarrow B)$ for some A and B for the equation to make sense.

The operation lam is second-order (also called a *binder*): its third input (after the two implicit inputs A, B) is a function.

Notation 42 (Isomorphism)**.** *The last four lines can be written more consisely:*

$$\mathsf{lam} : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \cong \mathsf{Tm}\,(A \Rightarrow B) : - \cdot -$$

We say that the sets $(\mathsf{Tm}\, A \to \mathsf{Tm}\, B)$ *and* $\mathsf{Tm}\,(A \Rightarrow B)$ *are isomorphic. We write the maps on the two sides of* $\cong$*, the notation in general is*

$$(f : X \cong Y : g) \;:=\; (f : X \to Y) \times (g : Y \to X) \times \big((x : X) \to g\,(f\, x) = x\big) \times \big((y : Y) \to f\,(g\, y) = y\big).$$

Derivability is programming: we can program in a SOGAT by postulating a second-order model in Agda or Coq and combining the components to build new programs. We can run the programs by proving (deriving) equalities between them.

Example 43. *In any second-order model of STLC, we define the identity function on* ι *by* $\mathsf{lam}\,\{\iota\}\{\iota\}\,(\lambda x.\, x)$*, which is in* $\mathsf{Tm}\,(\iota \Rightarrow \iota)$*. Note the difference between the object theory* lam *and meta* λ*. Actually, we can define the identity function on any type using meta quantification:*

$$\lambda A.\, \mathsf{lam}\,\{A\}\{A\}\,(\lambda x.\, x) : (A : \mathsf{Ty}) \to \mathsf{Tm}\,(A \Rightarrow A).$$

We compute (run the programs) in a second-order model deriving equalities. E.g. given $u : \mathsf{Tm}\,\iota$ *we argue*

$$(\mathsf{lam}\,(\lambda x.\, x)) \cdot u \overset{\beta}{=} (\lambda x.\, x)\, u \overset{\text{meta function application}}{=} u.$$

This seems like cheating: we avoided writing substitution by pushing it into the metatheory. But this is OK, we do not want to fully bootstrap STLC, we just want to define it. In the metatheory we of course assume (higher-order) functions, function application, its β *rule, and so on.*

Notation 44 (Derivation rules for SOGATs)**.** *Similarly to GATs, the derivation rule notation uses uncurried function space, named parameters and horizontal lines instead of the arrow* $\to$ *symbol. For arrows in second-order positions, we use the turnstile* $\vdash$*, and we use named function application for the* $\vdash$ *function space. Second-order models of STLC are described by the following derivation rules:*

$$\frac{}{\mathsf{Ty} : \mathsf{Set}} \qquad \frac{A : \mathsf{Ty}}{\mathsf{Tm}\, A : \mathsf{Set}} \qquad \frac{}{\iota : \mathsf{Ty}} \qquad \frac{A : \mathsf{Ty} \qquad B : \mathsf{Ty}}{A \Rightarrow B : \mathsf{Ty}} \qquad \frac{x : \mathsf{Tm}\, A \vdash t : \mathsf{Tm}\, B}{\mathsf{lam}\, x.t : \mathsf{Tm}\,(A \Rightarrow B)}$$

$$\frac{t : \mathsf{Tm}\,(A \Rightarrow B) \qquad a : \mathsf{Tm}\, A}{t \cdot u : \mathsf{Tm}\, B} \qquad \frac{}{(\mathsf{lam}\, x.b) \cdot a = b[x \mapsto a]}\,\beta \qquad \frac{}{f = \mathsf{lam}\, x.f \cdot x}\,\eta$$

Some arguments of the horizontal line function space can be implicit. Sometimes the name of the rule is written on the right hand side of the horizontal line.

We can omit the $: \mathsf{Ty}$ *and* Tm *parts and more concisely write the following without sacrificing precision (this is always the case when there are two sorts* $\mathsf{Ty} : \mathsf{Set}$*,* $\mathsf{Tm} : \mathsf{Ty} \to \mathsf{Set}$*).*

$$\bar{\iota} \qquad \frac{A \qquad B}{A \Rightarrow B} \qquad \frac{x : A \vdash t : B}{\mathsf{lam}\, x.t : A \Rightarrow B} \qquad \frac{t : A \Rightarrow B \qquad a : A}{t \cdot u : B} \qquad \frac{}{(\mathsf{lam}\, x.b) \cdot a = b[x \mapsto a]} \qquad \frac{}{f = \mathsf{lam}\, x.f \cdot x}$$

Example 45 (Second-order operations in mathematics)**.** *For the integral operation, the derivation rule notation is on the left and the second-order algebraic notation is on the right:*

$$\frac{a : \mathbb{R} \qquad b : \mathbb{R} \qquad x : \mathbb{R} \vdash t : \mathbb{R}}{\int_a^b t\, dx} \qquad\qquad \int : \mathbb{R} \to \mathbb{R} \to (\mathbb{R} \to \mathbb{R}) \to \mathbb{R},$$

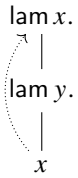
the expression $\int_0^1 \frac{1}{x^2}\, dx$ *corresponds to* $\int 0\, 1\, (\lambda x.\, \frac{1}{x^2})$.

Notation 46 (Derivation trees for SOGATs)**.** *When we write derivation trees, the second-order arguments are collected on the left hand side of the* $\vdash$*. For example, given* $A : \mathsf{Ty}$ *and* $B : \mathsf{Ty}$*, we derive the constant function as follows: the verbose notation is on the left, the concise is on the right. In the concise notation we also omit the* λ*s.*

$$\dfrac{\dfrac{\dfrac{}{x : \mathsf{Tm}\, A, y : \mathsf{Tm}\, B \vdash x : \mathsf{Tm}\, A}}{x : \mathsf{Tm}\, A \vdash \mathsf{lam}\, \lambda y.\, x : \mathsf{Tm}\, (B \Rightarrow A)}}{\mathsf{lam}\, \lambda x.\, \mathsf{lam}\, \lambda y.\, x : \mathsf{Tm}\, (A \Rightarrow B \Rightarrow A)} \qquad \dfrac{\dfrac{\dfrac{}{x : A, y : B \vdash x : A}}{x : A \vdash \mathsf{lam}\, y.\, x : B \Rightarrow A}}{\mathsf{lam}\, x.\, \mathsf{lam}\, y.\, x : A \Rightarrow B \Rightarrow A}$$

Now the leaves of the tree can be assumptions from the left hand side of the turnstile in addition to derivation rules without arguments. The algebraic (or Coq/Agda) version of this is simply the conclusion $\mathsf{lam}\, \lambda x.\, \mathsf{lam}\, \lambda y.\, x$*, or* $\mathsf{lam}\, (\lambda x.\, \mathsf{lam}\, (\lambda y.\, x))$ *with more brackets.*

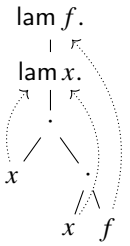
Notation 47 (Abstract binding trees)**.** *Another notation for SOGAT-derivation trees is abstract* binding *trees where variables are pointers to the binders which have to reachable through the path to the root. The constant function is drawn as follows.*



Abstract binding tree notation has the same issue as abstract syntax trees: the restriction that trees can only be put together in well-typed ways is not apparent.

Example 48. *We define the "double" function using short derivation rule and abstract binding tree notation.*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \Rightarrow A, x : A \vdash f : A \Rightarrow A} \qquad \dfrac{\dfrac{}{f : A \Rightarrow A, x : A \vdash f : A \Rightarrow A} \qquad \dfrac{}{f : A \Rightarrow A, x : A \vdash x : A}}{f : A \Rightarrow A, x : A \vdash f \cdot x : A}}{f : A \Rightarrow A, x : A \vdash f \cdot (f \cdot x) : A}}{f : A \Rightarrow A \vdash \mathsf{lam}\, x.\, f \cdot (f \cdot x) : A \Rightarrow A}}{\mathsf{lam}\, f.\mathsf{lam}\, x.\, f \cdot (f \cdot x) : (A \Rightarrow A) \Rightarrow A \Rightarrow A}$$



Example 49. *Function composition:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f : B \Rightarrow A, g : C \Rightarrow B, x : C \vdash f : B \Rightarrow A} \qquad \dfrac{\dfrac{}{f : B \Rightarrow A, g : C \Rightarrow B, x : C \vdash g : C \Rightarrow B} \qquad \dfrac{}{\ldots, x : C \vdash x : C}}{f : B \Rightarrow A, g : C \Rightarrow B, x : C \vdash g \cdot x : B}}{f : B \Rightarrow A, g : C \Rightarrow B, x : C \vdash f \cdot (g \cdot x) : A}}{f : B \Rightarrow A, g : C \Rightarrow B \vdash \mathsf{lam}\, x.\, f \cdot (g \cdot x) : C \Rightarrow A}}{f : B \Rightarrow A \vdash \mathsf{lam}\, g.\mathsf{lam}\, x.\, f \cdot (g \cdot x) : (C \Rightarrow B) \Rightarrow C \Rightarrow A}}{\mathsf{lam}\, f.\mathsf{lam}\, g.\mathsf{lam}\, x.\, f \cdot (g \cdot x) : (B \Rightarrow A) \Rightarrow (C \Rightarrow B) \Rightarrow C \Rightarrow A}$$

Example 50. *We define a model of STCC (Definition 26) assuming a second-order model of STLC.*

$$\begin{aligned}
\mathsf{Ty} &:= \mathsf{Ty} \\
\mathsf{Tm}\, A &:= \mathsf{Tm}\, A \\
\iota &:= \iota \\
A \Rightarrow B &:= A \Rightarrow B \\
\mathsf{K} &:= \mathsf{lam}\, \big(\lambda u.\, \mathsf{lam}\, (\lambda f.\, u)\big) \\
\mathsf{S} &:= \mathsf{lam}\, \Big(\lambda f.\, \mathsf{lam}\, \big(\lambda g.\, \mathsf{lam}\, (\lambda u.\, f \cdot u \cdot (g \cdot u))\big)\Big)
\end{aligned}$$

The equations hold, e.g.:

$$\begin{aligned}
&\mathsf{K}\beta : \mathsf{K} \cdot u \cdot f && =(\textit{definition of } \mathsf{K})\\
&\quad \Big(\big(\mathsf{lam}\,(\lambda u.\,\mathsf{lam}\,(\lambda f.\,u))\big) \cdot u\Big) \cdot f && =(\beta)\\
&\quad \Big((\lambda u.\,\mathsf{lam}\,(\lambda f.\,u))\,u\Big) \cdot f && =(\textit{meta function application})\\
&\quad \Big(\mathsf{lam}\,(\lambda f.\,u)\big)\Big) \cdot f && =(\beta)\\
&\quad (\lambda f.\,u)\,f && =(\textit{meta function application})\\
&\quad u
\end{aligned}$$

Exercise 51. *Prove that* $\mathsf{S}\beta$ *holds.*

Food for thought 52. *Can we derive a second-order model of STLC from a model of STCC? What if we start with a second-order model having some extra equations [AKSV23]?*

Example 53. *We can use derivation tree building to do informal type inference. For example, we don't know whether the expression*

$$\mathsf{lam}\,\lambda x.\mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z$$

has a type (makes sense in a second-order model of STLC), so we try to build its derivation tree from the bottom. Let's say it has an arbitrary type A*.*

$$\frac{?}{\mathsf{lam}\,\lambda x.\mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : A}$$

As the expression starts with lam*, we use its derivation rule and we learn that* $A = B \Rightarrow C$ *for some* B *and* C*:*

$$\frac{\dfrac{?}{x : B \vdash \mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : C}}{\mathsf{lam}\,\lambda x.\mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : B \Rightarrow C}$$

We use lam *again and we learn* $C = D \Rightarrow E$*, so we replace all occurrences of* C *with* $D \Rightarrow E$*:*

$$\frac{\dfrac{\dfrac{?}{x : B, y : D \vdash \mathsf{lam}\,\lambda z.y \cdot x \cdot z : E}}{x : B \vdash \mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : D \Rightarrow E}}{\mathsf{lam}\,\lambda x.\mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : B \Rightarrow (D \Rightarrow E)}$$

We use lam *again and we learn* $E = F \Rightarrow G$*:*

$$\frac{\dfrac{\dfrac{\dfrac{?}{x : B, y : D, z : F \vdash y \cdot x \cdot z : G}}{x : B, y : D \vdash \mathsf{lam}\,\lambda z.y \cdot x \cdot z : F \Rightarrow G}}{x : B \vdash \mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : D \Rightarrow (F \Rightarrow G)}}{\mathsf{lam}\,\lambda x.\mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : B \Rightarrow (D \Rightarrow (F \Rightarrow G))}$$

Now we can only use the derivation rule of application $(\cdot)$*, and we introduce a new metavariable* H *because we cannot read off the domain of the function from its conclusion. Recall that* $y \cdot x \cdot z = (y \cdot x) \cdot z$*.*

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{?}{x : B, y : D, z : F \vdash y \cdot x : H \Rightarrow G} \qquad \dfrac{?}{x : B, y : D, z : F \vdash z : H}}{x : B, y : D, z : F \vdash y \cdot x \cdot z : G}}{x : B, y : D \vdash \mathsf{lam}\,\lambda z.y \cdot x \cdot z : F \Rightarrow G}}{x : B \vdash \mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : D \Rightarrow (F \Rightarrow G)}}{\mathsf{lam}\,\lambda x.\mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : B \Rightarrow (D \Rightarrow (F \Rightarrow G))}$$

We first reach a leaf on the right hand side: z *is to the left of the turnstile, so we learn* $H = F$*:*

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{?}{x : B, y : D, z : F \vdash y \cdot x : F \Rightarrow G} \qquad \dfrac{}{x : B, y : D, z : F \vdash z : F}}{x : B, y : D, z : F \vdash y \cdot x \cdot z : G}}{x : B, y : D \vdash \mathsf{lam}\,\lambda z.y \cdot x \cdot z : F \Rightarrow G}}{x : B \vdash \mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : D \Rightarrow (F \Rightarrow G)}}{\mathsf{lam}\,\lambda x.\mathsf{lam}\,\lambda y.\mathsf{lam}\,\lambda z.y \cdot x \cdot z : B \Rightarrow (D \Rightarrow (F \Rightarrow G))}$$

On the left hand side, there is an application $\cdot$, we introduce a new metavariable I:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{?}{x : B, y : D, z : F \vdash y : I \Rightarrow F \Rightarrow G} \quad \dfrac{?}{x : B, y : D, z : F \vdash x : I}}{x : B, y : D, z : F \vdash y \cdot x : F \Rightarrow G} \quad \dfrac{}{x : B, y : D, z : F \vdash z : F}}{x : B, y : D, z : F \vdash y \cdot x \cdot z : G}}{x : B, y : D \vdash \mathsf{lam}\, \lambda z.y \cdot x \cdot z : F \Rightarrow G}}{x : B \vdash \mathsf{lam}\, \lambda y.\mathsf{lam}\, \lambda z.y \cdot x \cdot z : D \Rightarrow (F \Rightarrow G)}}{\mathsf{lam}\, \lambda x.\mathsf{lam}\, \lambda y.\mathsf{lam}\, \lambda z.y \cdot x \cdot z : B \Rightarrow (D \Rightarrow (F \Rightarrow G))}$$

The solution to the right hand side ? forces $I = B$, because this is the type of x:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{?}{x : B, y : D, z : F \vdash y : B \Rightarrow F \Rightarrow G} \quad \dfrac{}{x : B, y : D, z : F \vdash x : B}}{x : B, y : D, z : F \vdash y \cdot x : F \Rightarrow G} \quad \dfrac{}{x : B, y : D, z : F \vdash z : F}}{x : B, y : D, z : F \vdash y \cdot x \cdot z : G}}{x : B, y : D \vdash \mathsf{lam}\, \lambda z.y \cdot x \cdot z : F \Rightarrow G}}{x : B \vdash \mathsf{lam}\, \lambda y.\mathsf{lam}\, \lambda z.y \cdot x \cdot z : D \Rightarrow (F \Rightarrow G)}}{\mathsf{lam}\, \lambda x.\mathsf{lam}\, \lambda y.\mathsf{lam}\, \lambda z.y \cdot x \cdot z : B \Rightarrow (D \Rightarrow (F \Rightarrow G))}$$

On the left hand side we learn $D = B \Rightarrow F \Rightarrow G$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : B, y : B \Rightarrow F \Rightarrow G, z : F \vdash y : B \Rightarrow F \Rightarrow G} \quad \dfrac{}{x : B, y : B \Rightarrow F \Rightarrow G, z : F \vdash x : B}}{x : B, y : B \Rightarrow F \Rightarrow G, z : F \vdash y \cdot x : F \Rightarrow G} \quad \dfrac{}{\ldots, z : F \vdash z : F}}{x : B, y : B \Rightarrow F \Rightarrow G, z : F \vdash y \cdot x \cdot z : G}}{x : B, y : B \Rightarrow F \Rightarrow G \vdash \mathsf{lam}\, \lambda z.y \cdot x \cdot z : F \Rightarrow G}}{x : B \vdash \mathsf{lam}\, \lambda y.\mathsf{lam}\, \lambda z.y \cdot x \cdot z : (B \Rightarrow F \Rightarrow G) \Rightarrow (F \Rightarrow G)}}{\mathsf{lam}\, \lambda x.\mathsf{lam}\, \lambda y.\mathsf{lam}\, \lambda z.y \cdot x \cdot z : B \Rightarrow ((B \Rightarrow F \Rightarrow G) \Rightarrow (F \Rightarrow G))}$$

So the expression indeed has a type.

Exercise 54. *Show that function composition (Example 49) is associative.*

Exercise 55. *Show that for any $A : \mathsf{Ty}$, $\mathsf{Tm}\,(A \Rightarrow A)$ is a monoid with function composition and identity.*

Exercise 56. *Show that we can define a category where objects are Ty, a morphism from A to be is $\mathsf{Tm}\,(A \Rightarrow B)$, composition is function composition (a refined version of the previous exercise).*

We cannot prove logical consistency just assuming a second-order model, because it might be the trivial model. We also cannot prove equational consistency saying that there are terms a, a' such that $a = a' \to \mathbb{0}$, but we can get something similar: instead of obtaining a meta $\mathbb{0}$, we obtain that the second-order model is trivial:

Exercise 57. *Prove that for any $B : \mathsf{Ty}$ we have a $A : \mathsf{Ty}$ and $a, a' : \mathsf{Tm}\, A$ where $a = a'$ implies that any $b, b' : \mathsf{Tm}\, B$, $b = b'$.*

Example 58. *There is a second-order model of STLC where there is an $A : \mathsf{Ty}$ and $a, a' : \mathsf{Tm}\, A$ such that $a \neq a'$.*

3.2 System T

The following language has a proper type of natural numbers, it does not simply inherit $\mathbb{N}$ from the metatheory, like Razor did.

Definition 59 (System T)**.** *A second-order model of System T comprises the following components:*

$$\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{lam} & : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \cong \mathsf{Tm}\,(A \Rightarrow B) : - \cdot - \\
\mathsf{Nat} & : \mathsf{Ty} \\
\mathsf{zero} & : \mathsf{Tm}\, \mathsf{Nat} \\
\mathsf{suc} & : \mathsf{Tm}\, \mathsf{Nat} \to \mathsf{Tm}\, \mathsf{Nat} \\
\mathsf{ite} & : \mathsf{Tm}\, A \to (\mathsf{Tm}\, A \to \mathsf{Tm}\, A) \to \mathsf{Tm}\, \mathsf{Nat} \to \mathsf{Tm}\, A \\
\mathsf{Nat}\beta_1 & : \mathsf{ite}\, z\, s\, \mathsf{zero} = z \\
\mathsf{Nat}\beta_2 & : \mathsf{ite}\, z\, s\, (\mathsf{suc}\, t) = s\, (\mathsf{ite}\, z\, s\, t)
\end{array}$$

Example 60. *The double function:*

$$\begin{aligned}&\mathsf{double} : \mathsf{Tm}\,(\mathsf{Nat} \Rightarrow \mathsf{Nat})\\&\mathsf{double} := \mathsf{lam}\,\lambda x.\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,x\end{aligned}$$

Computing the double of 2:

$$\begin{aligned}&\mathsf{double} \cdot \mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero}) && =(\textit{abbreviation})\\&\big(\mathsf{lam}\,\lambda x.\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,x\big) \cdot \mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero}) && =(\beta)\\&\big(\lambda x.\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,x\big)\,(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})) && =(\textit{meta application})\\&\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})) && =(\mathsf{Nat}\beta_2)\\&(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,\big(\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,(\mathsf{suc}\;\mathsf{zero})\big) && =(\textit{meta aplication})\\&\mathsf{suc}\,\Big(\mathsf{suc}\,\big(\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,(\mathsf{suc}\;\mathsf{zero})\big)\Big) && =(\mathsf{Nat}\beta_2)\\&\mathsf{suc}\,\Big(\mathsf{suc}\,\big((\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,(\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,\mathsf{zero})\big)\Big) && =(\textit{meta application})\\&\mathsf{suc}\,\Big(\mathsf{suc}\,\big(\mathsf{suc}\,(\mathsf{suc}\,(\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,\mathsf{zero}))\big)\Big) && =(\mathsf{Nat}\beta_1)\\&\mathsf{suc}\,\Big(\mathsf{suc}\,\big(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})\big)\big)\Big)\end{aligned}$$

Definition 61 (Embedding $\mathbb{N}$ into $\mathsf{Tm}\,\mathsf{Nat}$)**.** *The meta function* $\ulcorner - \urcorner : \mathbb{N} \to \mathsf{Tm}\,\mathsf{Nat}$ *is defined as* $\ulcorner n \urcorner := \mathsf{suc}^n\,\mathsf{zero}$*, e.g.* $\ulcorner 3 \urcorner = \mathsf{suc}\,(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero}))$*.*

Definition 62 (Definability)**.** *An* $f : \mathbb{N} \to \mathbb{N}$ *is definable in a second-order model of System T if there is a* $t : \mathsf{Tm}\,(\mathsf{Nat} \Rightarrow \mathsf{Nat})$ *such that for all* n*,* $\ulcorner f\,n \urcorner = t \cdot \ulcorner n \urcorner$*. An* $f : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ *is definable if there is a* $t : \mathsf{Tm}\,(\mathsf{Nat} \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat})$ *such that for all* m*,* n*,* $\ulcorner f\,m\,n \urcorner = t \cdot \ulcorner m \urcorner \cdot \ulcorner n \urcorner$*.*

Exercise 63. *Addition, multiplication, exponentiation, the Ackermann function are definable in any second-order model of System T.*

Exercise 64. *Which of the following terms define identity?*

$$\begin{aligned}&\mathsf{lam}\,\lambda x.x\\&\mathsf{lam}\,\lambda x.\mathsf{ite}\,\mathsf{zero}\,\mathsf{suc}\,x\\&\mathsf{lam}\,\lambda x.\mathsf{ite}\,x\,(\lambda y.y)\,x\\&\mathsf{lam}\,\lambda x.\mathsf{ite}\,\mathsf{zero}\,(\lambda y.y)\,x\\&\mathsf{lam}\,\lambda x.\mathsf{ite}\,(\mathsf{suc}\,\mathsf{zero})\,\mathsf{suc}\,x\\&\mathsf{lam}\,\lambda x.\mathsf{ite}\,(\mathsf{suc}\,\mathsf{zero})\,(\lambda y.y)\,x\end{aligned}$$

Exercise 65. *Which of the following terms define the function* $x \mapsto 2 * x + 1$*?*

$$\begin{aligned}&\mathsf{lam}\,\lambda x.\mathsf{suc}\,x\\&\mathsf{lam}\,\lambda x.\mathsf{suc}\,(\mathsf{ite}\,x\,\mathsf{suc}\,x)\\&\mathsf{lam}\,\lambda x.\mathsf{ite}\,(\mathsf{suc}\,x)\,\mathsf{suc}\,x\\&\mathsf{lam}\,\lambda x.\mathsf{ite}\,(\mathsf{suc}\,\mathsf{zero})\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,x\\&\mathsf{lam}\,\lambda x.\mathsf{suc}\,(\mathsf{ite}\,\mathsf{zero}\,(\lambda y.\mathsf{suc}\,(\mathsf{suc}\,y))\,x)\end{aligned}$$

Exercise 66. *Show that the predecessor function is definable (difficult).*

It is not possible to define a term $\mathsf{pred} : \mathsf{Tm}\,(\mathsf{Nat} \Rightarrow \mathsf{Nat})$ such that $\mathsf{pred}\,(\mathsf{suc}\,n) = n$ for any $n : Tm\,\mathsf{Nat}$. [**?**] For this, we would need the recursor for Nat which in its method for successor receives the natural number, not only the result of the recursive call:

$$\begin{aligned}&\mathsf{rec} : \mathsf{Tm}\,A \to (\mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\,A \to \mathsf{Tm}\,A) \to \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Tm}\,A\\&\mathsf{rec}\,z\,s\,\mathsf{zero} = z\\&\mathsf{rec}\,z\,s\,(\mathsf{suc}\,t) = s\,t\,(\mathsf{rec}\,z\,s\,t)\end{aligned}$$

3.3 PCF

Definition 67 (PCF)**.** *A second-order model of PCF has two sorts (types and type-indexed terms) and the following additional components using concise derivation rule notation:*

$$\frac{A \qquad B}{A \Rightarrow B} \qquad \frac{x : A \vdash b : B}{\mathsf{lam}\, x.b : A \Rightarrow B} \qquad \frac{f : A \Rightarrow B \qquad a : A}{f \cdot a : B} \qquad \frac{}{(\mathsf{lam}\, x.b) \cdot a = b[x \mapsto a]} \qquad \frac{x : A \vdash t : A}{\mathsf{fix}\, x.t : A} \qquad \frac{}{\mathsf{fix}\, x.t = t[x \mapsto \mathsf{fix}\, x.t]}$$

$$\frac{}{\mathsf{Nat}} \qquad \frac{}{\mathsf{zero} : \mathsf{Nat}} \qquad \frac{n : \mathsf{Nat}}{\mathsf{suc}\, n : \mathsf{Nat}} \qquad \frac{b : \mathsf{Nat} \qquad t : A \qquad f : A}{\mathsf{ifZero}\, b\, t\, f : A} \qquad \frac{}{\mathsf{ifZero}\, \mathsf{zero}\, t\, f = t} \qquad \frac{}{\mathsf{ifZero}\, (\mathsf{suc}\, n)\, t\, f = f}$$

$$\frac{n : \mathsf{Nat}}{\mathsf{pred}\, n : \mathsf{Nat}} \qquad \frac{}{\mathsf{pred}\, (\mathsf{suc}\, n) = n} \qquad \frac{}{\mathsf{pred}\, \mathsf{zero} = \mathsf{zero}}$$

Exercise 68. *Define the iterator of natural numbers and show its computation rules.*

Exercise 69. *Show that every type has an element.*

Exercise 70. *Show equational consistency: that is, if* $\mathsf{zero} = \mathsf{suc}\, \mathsf{zero}$*, then for all* A *and* $u, v : \mathsf{Tm}\, A$*,* $u = v$*.*

Exercise 71. *Show that using a second-order model of PCF, a second-order model of System T without* η *can be given.*

Exercise 72. *Which functions do the following PCF terms define?*

$$\mathsf{fix}\, (\lambda t.\mathsf{lam}\, \lambda x.x)$$
$$\mathsf{fix}\, (\lambda t.\mathsf{lam}\, \lambda x.\mathsf{ifZero}\, x\, x\, x)$$
$$\mathsf{fix}\, (\lambda t.\mathsf{lam}\, \lambda x.\mathsf{ifZero}\, x\, x\, (\mathsf{suc}\, \mathsf{zero}))$$
$$\mathsf{fix}\, (\lambda t.\mathsf{lam}\, \lambda x.\mathsf{ifZero}\, x\, (\mathsf{suc}\, \mathsf{zero})\, \mathsf{zero})$$
$$\mathsf{fix}\, \Big(\lambda t.\mathsf{lam}\, \lambda x.\mathsf{ifZero}\, x\, x\, \big(\mathsf{suc}\, (t \cdot \mathsf{pred}\, x)\big)\Big)$$
$$\mathsf{fix}\, \Big(\lambda t.\mathsf{lam}\, \lambda x.\mathsf{ifZero}\, x\, \mathsf{zero}\, \big(\mathsf{suc}\, (t \cdot \mathsf{pred}\, x)\big)\Big)$$
$$\mathsf{fix}\, \Big(\lambda t.\mathsf{lam}\, \lambda x.\mathsf{ifZero}\, x\, (\mathsf{suc}\, \mathsf{zero})\, \big(\mathsf{suc}\, (t \cdot \mathsf{pred}\, x)\big)\Big)$$

3.4 First-order logic

Definition 73 (Minimal intuitionistic first-order logic with natural deduction style proof theory)**.**

$$\frac{}{\mathsf{For} : \mathsf{Set}} \qquad \frac{}{\mathsf{Tm} : \mathsf{Set}} \qquad \frac{A : \mathsf{For} \qquad B : \mathsf{For}}{A \supset B : \mathsf{For}} \qquad \frac{x : \mathsf{Tm} \vdash A : \mathsf{For}}{\forall x.A : \mathsf{For}} \qquad \frac{t : \mathsf{Tm} \qquad t' : \mathsf{Tm}}{\mathsf{Eq}\, t\, t' : \mathsf{For}} \qquad \frac{A : \mathsf{For}}{\mathsf{Pf}\, A : \mathsf{Set}}$$

$$\frac{p : \mathsf{Pf}\, A \qquad q : \mathsf{Pf}\, A}{p = q} \qquad \frac{\mathsf{Pf}\, A \vdash \mathsf{Pf}\, B}{\mathsf{Pf}\, (A \supset B)} \qquad \frac{\mathsf{Pf}\, (A \supset B) \qquad \mathsf{Pf}\, A}{\mathsf{Pf}\, B} \qquad \frac{x : \mathsf{Tm} \vdash \mathsf{Pf}\, A}{\mathsf{Pf}\, (\forall x.A)} \qquad \frac{\mathsf{Pf}\, (\forall x.A) \qquad t : \mathsf{Tm}}{\mathsf{Pf}\, (A[x \mapsto t])}$$

$$\frac{t : \mathsf{Tm}}{\mathsf{Pf}\, (\mathsf{Eq}\, t\, t)} \qquad \frac{x : \mathsf{Tm} \vdash A : \mathsf{For} \qquad \mathsf{Pf}\, (\mathsf{Eq}\, t\, t') \qquad \mathsf{Pf}\, (A[x \mapsto t])}{\mathsf{Pf}\, (A[x \mapsto t'])}$$

Exercise 74. *Define full intuitionistic first-order logic as a SOGAT. Add something to make it classical.*

3.5 Polymorphism

Definition 75 (Hindley-Milner)**.**

$$\begin{aligned}
&\mathsf{MTy} && : \mathsf{Set}\\
&\mathsf{Ty} && : \mathsf{Set}\\
&\mathsf{Tm} && : \mathsf{Ty} \to \mathsf{Set}\\
&- \Rightarrow - && : \mathsf{MTy} \to \mathsf{MTy} \to \mathsf{MTy}\\
&\forall && : (\mathsf{MTy} \to \mathsf{Ty}) \to \mathsf{Ty}\\
&\mathsf{i} && : \mathsf{MTy} \to \mathsf{Ty}\\
&\mathsf{lam} && : \big(\mathsf{Tm}\, (\mathsf{i}\, A) \to \mathsf{Tm}\, (\mathsf{i}\, B)\big) \cong \mathsf{Tm}\, (\mathsf{i}\, (A \Rightarrow B)) : - \cdot -\\
&\mathsf{Lam} && : \big((A : \mathsf{MTy}) \to \mathsf{Tm}\, (B\, A)\big) \cong \mathsf{Tm}\, (\forall\, B) : - \bullet -
\end{aligned}$$

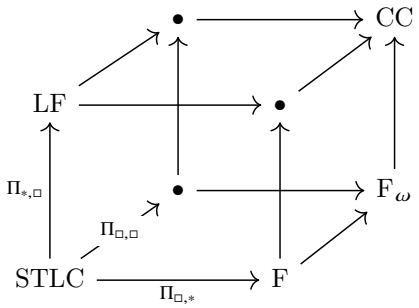
Definition 76 (System F)**.**

$$\begin{array}{ll} \mathsf{Ty} & : \mathsf{Set} \\ - \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\ \forall & : (\mathsf{Ty} \to \mathsf{Ty}) \to \mathsf{Ty} \end{array} \qquad \begin{array}{ll} \mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\ \mathsf{lam} & : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \cong \mathsf{Tm}\, (A \Rightarrow B) : - \cdot - \\ \mathsf{Lam} & : ((X : \mathsf{Ty}) \to \mathsf{Tm}\, (A\, X)) \cong \mathsf{Tm}\, (\forall\, A) : - \bullet - \end{array}$$

Definition 77 (System F$_\omega$)**.**

$$\begin{array}{ll} \mathsf{Kind} & : \mathsf{Set} \\ \mathsf{Ty} & : \mathsf{Kind} \to \mathsf{Set} \\ - \Rrightarrow - & : \mathsf{Kind} \to \mathsf{Kind} \to \mathsf{Kind} \\ \mathsf{LAM} & : (\mathsf{Ty}\, K \to \mathsf{Ty}\, L) \cong \mathsf{Ty}\, (K \Rrightarrow L) : - \CIRCLE - \\ * & : \mathsf{Kind} \\ \mathsf{Tm} & : \mathsf{Ty}\, * \to \mathsf{Set} \\ \forall & : (\mathsf{Ty}\, K \to \mathsf{Ty}\, *) \to \mathsf{Ty}\, * \\ \mathsf{Lam} & : ((X : \mathsf{Ty}\, K) \to \mathsf{Tm}\, (A\, X)) \cong \mathsf{Tm}\, (\forall\, A) : - \bullet - \\ - \Rightarrow - & : \mathsf{Ty}\, * \to \mathsf{Ty}\, * \to \mathsf{Ty}\, * \\ \mathsf{lam} & : (\mathsf{Tm}\, A \to \mathsf{Tm}\, B) \cong \mathsf{Tm}\, (A \Rightarrow B) : - \cdot - \end{array}$$

The following definition shows that all the languages in the lambda cube [Bar91] can be given as SOGATs. The simply typed lambda calculus (STLC) only includes $\Pi_{*,*}$, and the edges in each dimension add one of the other three Π types, respectively. The calculus of constructions (CC) includes all four Π types.



We don't give names to the maps in the universal properties.

Definition 78 (CC)**.**

$$\begin{array}{ll} \Box & : \mathsf{Set} \\ \mathsf{Ty} & : \Box \to \mathsf{Set} \\ * & : \Box \\ \mathsf{Tm} & : \mathsf{Ty}\, * \to \mathsf{Set} \\ \Pi_{*,*} & : (A : \mathsf{Ty}\, *) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, *) \to \mathsf{Ty}\, * \\ \Pi_{*,\Box} & : (A : \mathsf{Ty}\, *) \to (\mathsf{Tm}\, A \to \Box) \to \Box \\ \Pi_{\Box,*} & : (K : \Box) \to (\mathsf{Ty}\, K \to \mathsf{Ty}\, *) \to \mathsf{Ty}\, * \\ \Pi_{\Box,\Box} & : (K : \Box) \to (\mathsf{Ty}\, K \to \Box) \to \Box \end{array} \qquad \begin{array}{ll} \mathsf{Tm}\, (\Pi_{*,*}\, A\, B) & \cong (a : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (B\, a) \\ \mathsf{Ty}\, (\Pi_{*,\Box}\, A\, L) & \cong (a : \mathsf{Tm}\, A) \to \mathsf{Ty}\, (L\, a) \\ \mathsf{Tm}\, (\Pi_{\Box,*}\, K\, B) & \cong (A : \mathsf{Ty}\, K) \to \mathsf{Tm}\, (B\, A) \\ \mathsf{Ty}\, (\Pi_{\Box,\Box}, K\, L) & \cong (A : \mathsf{Ty}\, K) \to \mathsf{Ty}\, (L\, A) \end{array}$$

3.6 Martin-Löf type theory

Definition 79 (Minimal Martin-Löf type theory (mini-MLTT))**.**

$$\begin{array}{ll} \mathsf{Ty} & : \mathsf{Set} \\ \mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\ \iota & : \mathsf{Ty} \\ \Pi & : (A : \mathsf{Ty}) \to (\mathsf{Tm}\, A \to \mathsf{Ty}) \to \mathsf{Ty} \\ \mathsf{lam} & : ((a : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (B\, a)) \cong \mathsf{Tm}\, (\Pi\, A\, B) : - \cdot - \end{array}$$

Definition 80 (Martin-Löf type theory with Π and universes)**.**

$$\begin{array}{ll}
\mathsf{Ty} & : \mathbb{N} \to \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty}\, i \to \mathsf{Set} \\
\Pi & : (A : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
\mathsf{lam} & : ((a : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (B\, a)) \cong \mathsf{Tm}\, (\Pi\, A\, B) : - \cdot - \\
\mathsf{U} & : (i : \mathbb{N}) \to \mathsf{Ty}\, (1 + i) \\
\mathsf{c} & : \mathsf{Ty}\, i \cong \mathsf{Tm}\, (\mathsf{U}\, i) : \mathsf{El} \\
\mathsf{Lift} & : \mathsf{Ty}\, i \to \mathsf{Ty}\, (1 + i) \\
\mathsf{mk} & : \mathsf{Tm}\, A \cong \mathsf{Tm}\, (\mathsf{Lift}\, A) : \mathsf{un}
\end{array}$$

Definition 81 (Martin-Löf type theory with inductive types)**.** *We extend Definition 80 with the following.*

$$\begin{array}{ll}
\Sigma & : (A : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
(-, -) & : (a : \mathsf{Tm}\, A) \times \mathsf{Tm}\, (B\, a) \cong \mathsf{Tm}\, (\Sigma\, A\, B) : \mathsf{fst}, \mathsf{snd} \\
\bot & : \mathsf{Ty}\, 0 \\
\mathsf{exfalso} & : \mathsf{Tm}\, \bot \to \mathsf{Tm}\, A \\
\top & : \mathsf{Ty}\, 0 \\
\mathsf{tt} & : \top \cong \mathsf{Tm}\, \top \\
\mathsf{Bool} & : \mathsf{Ty}\, 0 \\
\mathsf{true} & : \mathsf{Tm}\, \mathsf{Bool} \\
\mathsf{false} & : \mathsf{Tm}\, \mathsf{Bool} \\
\mathsf{indBool} & : (C : \mathsf{Tm}\, \mathsf{Bool} \to \mathsf{Ty}\, i) \to \mathsf{Tm}\, (C\, \mathsf{true}) \to \mathsf{Tm}\, (C\, \mathsf{false}) \to (b : \mathsf{Tm}\, \mathsf{Bool}) \to \mathsf{Tm}\, (C\, b) \\
\mathsf{Bool}\beta_1 & : \mathsf{indBool}\, t\, f\, \mathsf{true} = t \\
\mathsf{Bool}\beta_2 & : \mathsf{indBool}\, t\, f\, \mathsf{false} = f \\
\mathsf{Id} & : (A : \mathsf{Ty}\, i) \to \mathsf{Tm}\, A \to \mathsf{Tm}\, A \to \mathsf{Ty}\, i \\
\mathsf{refl} & : (a : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (\mathsf{Id}\, A\, a\, a) \\
\mathsf{J} & : \big(C : (x : \mathsf{Tm}\, A) \to \mathsf{Tm}\, (\mathsf{Id}\, A\, a\, x) \to \mathsf{Ty}\, i\big) \to \\
 & \quad \mathsf{Tm}\, \big(C\, a\, (\mathsf{refl}\, a)\big) \to (x : \mathsf{Tm}\, A)(e : \mathsf{Tm}\, (\mathsf{Id}\, A\, a\, x)) \to \mathsf{Tm}\, (C\, x\, e) \\
\mathsf{Id}\beta & : \mathsf{J}\, C\, w\, a\, (\mathsf{refl}\, a) = w \\
\mathsf{W} & : (S : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, S \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i \\
\mathsf{sup} & : (s : \mathsf{Tm}\, S) \to (\mathsf{Tm}\, (P\, s) \to \mathsf{Tm}\, (\mathsf{W}\, S\, P)) \to \mathsf{Tm}\, (\mathsf{W}\, S\, P) \\
\mathsf{indW} & : (C : \mathsf{Tm}\, (\mathsf{W}\, S\, P) \to \mathsf{Ty}\, i) \to \Big(((p : \mathsf{Tm}\, (P\, s)) \to \mathsf{Tm}\, (C\, (f\, p))) \to \mathsf{Tm}\, \big(C\, (\mathsf{sup}\, s\, f)\big)\Big) \to \\
 & \quad (w : \mathsf{Tm}\, (\mathsf{W}\, \mathsf{S}\, \mathsf{P})) \to \mathsf{Tm}\, \big(C\, w\big) \\
\mathsf{W}\beta & : \mathsf{indW}\, C\, h\, (\mathsf{sup}\, s\, f) = h\, (\lambda p.\mathsf{indW}\, C\, h\, (f\, p))
\end{array}$$

3.7 Theories of signatures for (SO)(G)ATs

Theories of signatures (ToSs) are languages for describing signatures for algebraic theories. Above we defined (SO)GATs by giving the notion of (second-order) model, signatures are a more precise language which enforces that all operators result in a sort, strict positivity etc.

Definition 82 (AT)**.** *A second-order model of the ToS for ATs:*

$$\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\Sigma & : (A : \mathsf{Ty}) \to (\mathsf{Tm}\, A \to \mathsf{Ty}) \to \mathsf{Ty} \\
\mathsf{Srt} & : \mathsf{Ty} \\
\Pi\mathsf{Srt} & : (\mathsf{Tm}\, \mathsf{Srt} \to \mathsf{Ty}) \to \mathsf{Ty} \\
- \cdot - & : \mathsf{Tm}\, (\Pi\mathsf{Srt}\, B) \to (x : \mathsf{Tm}\, \mathsf{Srt}) \to \mathsf{Tm}\, (B\, x) \\
\mathsf{Id} & : \mathsf{Tm}\, \mathsf{Srt} \to \mathsf{Tm}\, \mathsf{Srt} \to \mathsf{Ty}
\end{array}$$

A signature is an element of Ty*.*

For a $B : \mathsf{Ty}$, we introduce the abbreviation $\mathsf{Srt} \Rightarrow B := \Pi\mathsf{Srt}\, (\lambda\_.\, B)$.

Example 83. *The AT of monoids (see Definition 1) is given by the following signature:*

$$\Sigma\, (\mathsf{Srt} \Rightarrow \mathsf{Srt} \Rightarrow \mathsf{Srt})\, \lambda op.\, \Pi\mathsf{Srt}\, \lambda x.\, \Pi\mathsf{Srt}\, \lambda y.\, \Pi\mathsf{Srt}\, \lambda z.\, \mathsf{Id}\, \big(op \cdot (op \cdot x \cdot y) \cdot z\big)\, \big(op \cdot x \cdot (op \cdot y \cdot z)\big) \times$$
$$\Sigma\, \mathsf{Srt}\, \lambda u.\, \big(\Pi\mathsf{Srt}\, \lambda x.\, \mathsf{Id}\, (op \cdot u \cdot x)\, x\big) \times \big(\Pi\mathsf{Srt}\, \lambda x.\, \mathsf{Id}\, (op \cdot x \cdot u)\, x\big)$$

Exercise 84. *Define the signature for the following ATs: pointed set with an endofunction (see Definition 9), untyped combinator calculus.*

Definition 85 (GAT)**.** *A second-order model of the ToS for GATs:*

$$
\begin{aligned}
&\mathsf{Ty} && : \mathsf{Set}\\
&\mathsf{Tm} && : \mathsf{Ty} \to \mathsf{Set}\\
&\Sigma && : (A : \mathsf{Ty}) \to (\mathsf{Tm}\, A \to \mathsf{Ty}) \to \mathsf{Ty}\\
&(-,-) && : (a : \mathsf{Tm}\, A) \times \mathsf{Tm}\,(B\, a) \cong \mathsf{Tm}\,(\Sigma\, A\, B) : \mathsf{fst}, \mathsf{snd}\\
&\mathsf{U} && : \mathsf{Ty}\\
&\mathsf{El} && : \mathsf{Tm}\,\mathsf{U} \to \mathsf{Ty}\\
&\Pi && : (a : \mathsf{Tm}\,\mathsf{U}) \to (\mathsf{Tm}\,(\mathsf{El}\, a) \to \mathsf{Ty}) \to \mathsf{Ty}\\
&- \cdot - && : \mathsf{Tm}\,(\Pi\, a\, B) \to (x : \mathsf{Tm}\,(\mathsf{El}\, a)) \to \mathsf{Tm}\,(B\, x)\\
&\mathsf{Id} && : (a : \mathsf{Tm}\,\mathsf{U}) \to \mathsf{Tm}\,(\mathsf{El}\, a) \to \mathsf{Tm}\,(\mathsf{El}\, a) \to \mathsf{Ty}\\
&\mathsf{reflect} && : \mathsf{Tm}\,(\mathsf{Id}\, a\, u\, v) \to u = v
\end{aligned}
$$

A signature is an element of Ty*.*

We will use the abbreviations $A \Rightarrow B := \Pi\, A\, (\lambda\_.B)$ and $A \times B := \Sigma\, A\, (\lambda\_.B)$.

Example 86. *The GAT of preorders (see Definition 39) is given by the following signature:*

$$
\begin{aligned}
&\Sigma\, \mathsf{U}\, \lambda Ob.\Sigma\\
&\quad (Ob \Rightarrow Ob \Rightarrow \mathsf{U})\, \lambda Mor.\\
&\quad \big(\Pi\, Ob\, \lambda I.\Pi\, Ob\, \lambda J.\Pi\, Ob\, \lambda K.Mor \cdot J \cdot I \Rightarrow Mor \cdot K \cdot J \Rightarrow \mathsf{El}\,(Mor \cdot K \cdot I)\big)\times\\
&\quad \big(\Pi\, Ob\, \lambda I.\mathsf{El}\,(Mor \cdot I \cdot I)\big)\times\\
&\quad \big(\Pi\, Ob\, \lambda I.\Pi\, Ob\, \lambda J.\Pi\,(Mor \cdot J \cdot I)\, \lambda f.\Pi\,(Mor \cdot J \cdot I)\, \lambda g.\mathsf{Id}\,(Mor \cdot J \cdot I)\, f\, g\big)
\end{aligned}
$$

Exercise 87. *Define the signature for the following GATs: category, Razor, STCC.*

Definition 88 (SOGAT)**.** *A second-order model of the ToS for SOGATs is a second-order model of the ToS for GATs (Definition 85) extended with the following components:*

$$
\begin{aligned}
&\mathsf{U}^+ && : \mathsf{Ty}\\
&\mathsf{el}^+ && : \mathsf{Tm}\,\mathsf{U}^+ \to \mathsf{Tm}\,\mathsf{U}\\
&\pi^+ && : (a^+ : \mathsf{Tm}\,\mathsf{U}^+) \to \Big(\mathsf{Tm}\,\big(\mathsf{El}\,(\mathsf{el}^+\, a^+)\big) \to \mathsf{Tm}\,\mathsf{U}\Big) \to \mathsf{Tm}\,\mathsf{U}\\
&\mathsf{lam}^+ && : \Big(\big(x : \mathsf{El}\,(\mathsf{el}^+\, a^+)\big) \to \mathsf{Tm}\,(\mathsf{El}\,(b\, x))\Big) \cong \mathsf{Tm}\,\big(\mathsf{El}\,(\pi^+\, a^+\, b)\big) : - \cdot^+ -
\end{aligned}
$$

U^+ is the universe of those sorts which can appear at the left hand side of an arrow in an argument of an operator. In STLC, Ty is in U, while Tm is in U^+.

Example 89. *The signature for STLC (see Definition 41):*

$$
\begin{aligned}
&\Sigma\, \mathsf{U} \lambda Ty.\, \Sigma\\
&(Ty \Rightarrow \mathsf{U}^+)\, \lambda Tm.\\
&\mathsf{El}\, Ty \times \Sigma\\
&(Ty \Rightarrow Ty \Rightarrow \mathsf{El}\, Ty)\, \lambda arr.\, \Sigma\\
&\Big(\Pi\, Ty\, \lambda A.\, \Pi\, Ty\, \lambda B.\, \big(Tm \cdot A \Rightarrow^+ \mathsf{el}^+\,(Tm \cdot B)\big) \Rightarrow \mathsf{El}\,\big(\mathsf{el}^+\,(Tm \cdot (arr \cdot A \cdot B))\big)\Big)\, \lambda \mathsf{lam}.\, \Sigma\\
&\Big(\Pi\, Ty\, \lambda A.\, \Pi\, Ty\, \lambda B.\, \mathsf{el}^+\,(Tm \cdot (arr \cdot A \cdot B)) \Rightarrow \mathsf{el}^+\,(Tm \cdot A) \Rightarrow \mathsf{El}\,\big(\mathsf{el}^+\,(Tm \cdot B)\big)\Big)\, \lambda app.\\
&\Big(\Pi\, Ty\, \lambda A.\, \Pi\, Ty\, \lambda B.\, \Pi\,\big(Tm \cdot A \Rightarrow^+ \mathsf{el}^+\,(Tm \cdot B)\big)\, \lambda b.\, \Pi\,\big(\mathsf{el}^+\,(Tm \cdot A)\big)\, \lambda a.\\
&\quad \mathsf{Id}\,\big(\mathsf{el}^+\,(Tm \cdot B)\big)\,\big(app \cdot A \cdot B \cdot (lam \cdot A \cdot B \cdot b) \cdot a\big)\,(b \cdot^+ a)\Big)\times\\
&\Big(\Pi\, Ty\, \lambda A.\, \Pi\, Ty\, \lambda B.\, \Pi\,\big(Tm \cdot (arr \cdot A \cdot B)\big)\, \lambda f.\\
&\quad \mathsf{Id}\,\Big(\mathsf{el}^+\,\big(Tm \cdot (arr \cdot A \cdot B)\big)\Big)\, f\,\big(lam \cdot A \cdot B \cdot (\mathsf{lam}^+\, \lambda x.\, app \cdot A \cdot B \cdot f \cdot x)\big)\Big)
\end{aligned}
$$

Exercise 90. *Define the signature for the following SOGATs: untyped lambda calculus, System T, PCF, first-order logic, System F, System F_ω, Martin-Löf type theory, ToS of ATs, ToS of GATs, ToS of SOGATs. Carefully consider which sorts should be in* U *and which should be in* U^+*.*

Remark: we only showed how to define *closed* SOGATs, but e.g. Definition 80 is open as it refers to the external set $\mathbb{N}$. The ToS of (SO)GATs can be extended to allow open signatures, see [KX24].

4 Converting SOGATs into GATs

As mentioned in the beginning of Section 3, second-order algebraic theories are not really algebraic, there is no way to define morphisms of second-order models:

Example 91. *A second-order model of the untyped lambda calculus consists of a* $\mathsf{Tm} : \mathsf{Set}$ *together with*

$$\mathsf{lam} : (\mathsf{Tm} \to \mathsf{Tm}) \cong \mathsf{Tm} : - \cdot -.$$

A morphism between second-order models M and N would be a function $\mathsf{Tm} : \mathsf{Tm}_M \to \mathsf{Tm}_N$ *which preserves* $- \cdot -$ *by* $\mathsf{Tm}\,(t_M \cdot_M a_M) = \mathsf{Tm}\,t_M \cdot_N \mathsf{Tm}\,a_M$*, but how do we express preservation of* lam*? This would be an equation such as* $\mathsf{Tm}\,(\mathsf{lam}_M\,f_M) = \mathsf{lam}_N\,(\mathsf{Tm} \circ f_M \circ ?)$*, but we don't know what to put in place of the* $?$ *which is in* $\mathsf{Tm}_N \to \mathsf{Tm}_M$*. If we wanted to only define isomorphisms, this would work.*

Instead, we translate SOGATs to GATs and work with the resulting GAT. For GATs, we have good notions of morphism, dependent model, syntax, and so on, as we have seen in Section 2. That is, by a category of models of a SOGAT, we mean the category of models of the GAT which is the result of the translation.

Food for thought 92. *Actually, there are multiple different translations which end up in different notions of models, e.g. instead of parallel substitution calculus, one can use single substitutions [KX24], models can be contextual (where contexts are inductively built, which is a special case of models where certain sorts are inductively generated), contexts can come with concatenation, and so on. For each SOGAT, there should also be a combinatory first-order GAT, just like there is the combinatory first-order version of the SOGAT of lambda calculus.*

4.1 STLC

The idea of the translation is that we introduce a new sort of contexts which are a list of the free variables. We index the sorts of our theory with contexts, the context index lists the possible free variables in a term. This way we can get rid of second-order function spaces. For example, the $\mathsf{lam} : (\mathsf{Tm}\,A \to \mathsf{Tm}\,B) \to \mathsf{Tm}\,(A \Rightarrow B)$ second-order operation becomes $\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)$. That is, the $\mathsf{Tm}\,A$ dependency of the $\mathsf{Tm}\,B$ argument of lam becomes an extra variable of type A in the context (we read $\Gamma \triangleright A$ as the context Γ extended with an extra free variable of type A). We also introduce a sort of substitutions with their action on terms called instantiation. These allow expressing the β law in STLC: $\mathsf{lam}\,b \cdot a = b\,a$ becomes $\mathsf{lam}\,b \cdot a = b[\mathsf{id}, a]$ where $-[-]$ is the instantiation operator for Tm, (id, a) is a substitution from Γ to $\Gamma \triangleright A$ which leaves Γ untouched and substitutes a for the free variable of type A. The GAT that we obtain is called an explicit substitution calculus with parallel substitutions.

We first present and explain the result of the translation for STLC.

Definition 93 (First-order model of STLC (optimised version), see Definition 41 and Example 89)**.**

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
- \circ - & : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma \\
\mathsf{ass} & : (\gamma \circ \delta) \circ \theta = \gamma \circ (\delta \circ \theta) \\
\mathsf{id} & : \mathsf{Sub}\,\Gamma\,\Gamma \\
\mathsf{idl} & : \mathsf{id} \circ \gamma = \gamma \\
\mathsf{idr} & : \gamma \circ \mathsf{id} = \gamma \\
\diamond & : \mathsf{Con} \\
\epsilon & : \mathsf{Sub}\,\Gamma\,\diamond \\
\diamond\eta & : (\sigma : \mathsf{Sub}\,\Gamma\,\diamond) \to \sigma = \epsilon \\
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set} \\
-[-] & : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,A \\
[\circ] & : a[\gamma \circ \delta] = a[\gamma][\delta] \\
[\mathsf{id}] & : a[\mathsf{id}] = a
\end{array}
\qquad
\begin{array}{ll}
- \triangleright - & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Con} \\
-, - & : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,A \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A) \\
\mathsf{p} & : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma \\
\mathsf{q} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,A \\
\triangleright\beta_1 & : \mathsf{p} \circ (\gamma, a) = \gamma \\
\triangleright\beta_2 & : \mathsf{q}[\gamma, a] = a \\
\triangleright\eta & : \sigma = (\mathsf{p} \circ \sigma, \mathsf{q}[\sigma]) \\
\iota & : \mathsf{Ty} \\
- \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \\
\mathsf{lam}[] & : (\mathsf{lam}\,b)[\gamma] = \mathsf{lam}\,(b[\gamma \circ \mathsf{p}, \mathsf{q}]) \\
- \cdot - & : \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,B \\
\cdot[] & : (f \cdot a)[\gamma] = (f[\gamma]) \cdot (a[\gamma]) \\
\beta & : \mathsf{lam}\,b \cdot a = b[\mathsf{id}, a] \\
\eta & : f = \mathsf{lam}\,(f[\mathsf{p}] \cdot \mathsf{q})
\end{array}
$$

Contexts (lists of types) and substitutions (context morphisms, lists of terms) form a category ($\mathsf{Con}, \ldots, \mathsf{idr}$) with a terminal object ($\diamond$ is the empty context denoting no free variables, ϵ is the empty substitution into $\diamond$, $\diamond\eta$ expresses that it is unique). The sort Ty is unchanged compared to the second-order version. The sort of terms however has an extra context-index. $\mathsf{Tm}\,\Gamma\,A$ is a term of type A which might have free variables which are declared in Γ. The $-\triangleright-$ context extension operator allows us to put extra variables in a context. For example, the context $\diamond\triangleright\iota\triangleright\iota \Rightarrow \iota$ has two free variables of types ι and $\iota \Rightarrow \iota$, respectively. The sort $\mathsf{Sub}\,\Delta\,\Gamma$ is a substitution from Δ to Γ which means a list of Γ-many terms, all in context Δ. For example, if $\Gamma = \diamond \triangleright \iota \triangleright \iota \Rightarrow \iota$, then $\mathsf{Sub}\,\Delta\,\Gamma \cong \mathsf{Tm}\,\Delta\,\iota \times \mathsf{Tm}\,\Delta\,(\iota \Rightarrow \iota)$. This is ensured by the components $-, -, \ldots, \triangleright\eta$ which can be summarised as

$$(\mathsf{p} \circ -, \mathsf{q}[-]) : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A) \cong \mathsf{Sub}\,\Delta\,\Gamma \times \mathsf{Tm}\,\Delta\,A : (-, -).$$

Instantiation $-[-]$ gives meaning to variables via substitutions. For example, assume a term which depends only on a variable of type $\iota \Rightarrow \iota$, that is, $t : \mathsf{Tm}\,(\diamond \triangleright \iota \Rightarrow \iota)\,A$, and assume a closed term of type $\iota \Rightarrow \iota$, that is $u : \mathsf{Tm}\,\diamond\,(\iota \Rightarrow \iota)$. Now we can substitute u into t making it closed: $t[\epsilon, u] : \mathsf{Tm}\,\diamond\,A$. The rules $\mathsf{lam}[], \cdot[]$ explain how to commute instantiation and term formers, the rule $\triangleright\beta_2$ explains how to instantiate the last variable (De Bruijn index 0) in the context. Further De Bruijn indices are given by weakening: $1 = \mathsf{q}[\mathsf{p}]$, $2 = \mathsf{q}[\mathsf{p}][\mathsf{p}]$, $3 = \mathsf{q}[\mathsf{p}][\mathsf{p}][\mathsf{p}]$, and so on. The rule $\mathsf{lam}[]$ is interesting: here $\gamma : \mathsf{Sub}\,\Delta\,\Gamma$ and $b : \mathsf{Tm}\,(\Gamma \triangleright A)\,B$, so $b[\gamma]$ does not make sense. We have to *lift* γ so that we obtain a substitution which does not touch the last variable, $(\gamma \circ \mathsf{p}, \mathsf{q}) : \mathsf{Sub}\,(\Delta \triangleright A)\,(\Gamma \triangleright A)$, and apply this under the lam.

Example 94 (Naturality of $-, -$)**.** *We prove that in a first-order model of STLC, $(\gamma, a) \circ \delta = (\gamma \circ \delta, a[\delta])$ for any γ, a and δ.*

$$
\begin{array}{ll}
(\gamma, a) \circ \delta & =(\triangleright\eta) \\
(\mathsf{p} \circ ((\gamma, a) \circ \delta), \mathsf{q}[(\gamma, a) \circ \delta]) & =(\mathsf{ass}) \\
((\mathsf{p} \circ (\gamma, a)) \circ \delta, \mathsf{q}[(\gamma, a) \circ \delta]) & =(\triangleright\beta_1) \\
(\gamma \circ \delta, \mathsf{q}[(\gamma, a) \circ \delta]) & =([\circ]) \\
(\gamma \circ \delta, \mathsf{q}[\gamma, a][\delta]) & =(\triangleright\beta_2) \\
(\gamma \circ \delta, a[\delta]) &
\end{array}
$$

Exercise 95. *Prove that assuming naturality of $-, -$ (i.e. $(\gamma, a) \circ \delta = (\gamma \circ \delta, a[\delta])$), the functor laws for Tm ($[\circ]$, $[\mathsf{id}]$) can be derived.*

Exercise 96. *Define substitutions which swap variables, duplicate variables, forget variables somewhere in the middle of the context, i.e. elements of the following sets:*

$$\mathsf{Sub}\,(\Gamma \triangleright A \triangleright B_1 \triangleright \ldots \triangleright B_n \triangleright C \triangleright D_1 \triangleright \ldots \triangleright D_m)\,(\Gamma \triangleright C \triangleright B_1 \triangleright \ldots \triangleright B_n \triangleright A \triangleright D_1 \triangleright \ldots \triangleright D_m)$$
$$\mathsf{Sub}\,(\Gamma \triangleright A \triangleright B_1 \triangleright \ldots \triangleright B_n \triangleright C_1 \triangleright \ldots \triangleright C_m)\,(\Gamma \triangleright A \triangleright B_1 \triangleright \ldots \triangleright B_n \triangleright A \triangleright C_1 \triangleright \ldots \triangleright C_m)$$
$$\mathsf{Sub}\,(\Gamma \triangleright A \triangleright B_1 \triangleright \ldots \triangleright B_n)\,(\Gamma \triangleright B_1 \triangleright \ldots \triangleright B_n)$$

Exercise 97. *Lifting of $\gamma : \mathsf{Sub}\,\Delta\,\Gamma$ is $\gamma^\uparrow := (\gamma \circ \mathsf{p}, \mathsf{q})$. Show $(\gamma \circ \delta)^\uparrow = (\gamma^\uparrow) \circ (\delta^\uparrow)$, $\mathsf{id}^\uparrow = \mathsf{id}$, $\mathsf{p}^\uparrow \circ (\mathsf{id}, \mathsf{q}) = \mathsf{id}$.*

Example 98 (C.f. Example 43)**.** *In any first-order model of STLC, we have* $\mathsf{lam}\,\mathsf{q} : \mathsf{Tm}\,\diamond\,(\iota \Rightarrow \iota)$*. If we write the implicit arguments, this is* $\mathsf{lam}\,\{\diamond\}\,\{\iota\}\,\{\iota\}\,(\mathsf{q}\,\{\diamond\}\,\{\iota\})$*. We can move this program into any context* Γ *by instantiating it by* $\epsilon\,\{\Gamma\} : \mathsf{Sub}\,\Gamma\,\diamond$*, and obtain* $(\mathsf{lam}\,\mathsf{q})[\epsilon] \overset{\mathsf{lam}[]}{=} \mathsf{lam}\,(\mathsf{q}[\epsilon \circ \mathsf{p}, \mathsf{q}]) \overset{\triangleright\beta_2}{=} \mathsf{lam}\,\mathsf{q}$*. The original and the instantiated terms have different implicit arguments: the right hand side* $\mathsf{lam}\,\mathsf{q}$ *is actually* $\mathsf{lam}\,\{\Gamma\}\,\{\iota\}\,\{\iota\}\,(\mathsf{q}\,\{\Gamma\}\,\{\iota\})$*.*

Applying the identity function to some argument gives $\mathsf{lam}\,\mathsf{q} \cdot u \overset{\beta}{=} \mathsf{q}[\mathsf{id}, u] \overset{\triangleright\beta_2}{=} u$*.*

Example 99 (C.f. Example 48)**.** *In any first-order model of STLC, the double function is defined as*

$$\mathsf{lam}\left(\mathsf{lam}\left((\mathsf{q}[\mathsf{p}]) \cdot ((\mathsf{q}[\mathsf{p}]) \cdot \mathsf{q})\right)\right) : \mathsf{Tm}\ \diamond\ ((A \Rightarrow A) \Rightarrow A \Rightarrow A)$$

Exercise 100. *Derive* $\left(\mathsf{lam}\left(\mathsf{lam}\left((\mathsf{q}[\mathsf{p}]) \cdot ((\mathsf{q}[\mathsf{p}]) \cdot \mathsf{q})\right)\right)\right)[\epsilon] = \mathsf{lam}\left(\mathsf{lam}\left((\mathsf{q}[\mathsf{p}]) \cdot ((\mathsf{q}[\mathsf{p}]) \cdot \mathsf{q})\right)\right)$.

Exercise 101. *Derive* $\left(\mathsf{lam}\left(\mathsf{lam}\left((\mathsf{q}[\mathsf{p}]) \cdot ((\mathsf{q}[\mathsf{p}]) \cdot \mathsf{q})\right)\right)\right) \cdot t \cdot u = t \cdot (t \cdot u)$.

4.2 The general translation

The recipe for translating a SOGAT into a GAT is the following:

- the GAT starts with a category with a terminal object,
- closed sorts become presheaves on this category,
- open sorts become dependent presheaves,
- operations become natural transformations (that is, operations indexed by contexts together with substitution laws),
- equations become equations indexed by contexts,
- sorts which are in U^+ (which appear on the left hand side of an arrow in an operator argument) moreover have local representability structure (that is, come with a context extension operation)
- second-order arguments become context extensions,
- Π^+-applications become explicit instantiations,
- Π^+-abstractions become weakenings,
- variables bound by Π^+-functions become De Bruijn indices.

Here we show more examples. For the precise algorithm, see [KX24].

In Definition 93, we used some cleverness: we knew that there are no second-order Ty-operators (no binders in Ty, no types refer to variables), so we made Ty a closed sort. The generic algorithm does not know about this (we might have dependent types), so the first part of the GAT will be simply the result of translating the SOGAT signature $\Sigma \cup \lambda T y.\, T y \Rightarrow \mathsf{U}^+$ (types and terms indexed by types where we only have term-variables):

Definition 102 (Category with family, CwF)**.**

Con : Set
Sub : $\mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$
$- \circ -$: $\mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma$
ass : $(\gamma \circ \delta) \circ \theta = \gamma \circ (\delta \circ \theta)$
id : $\mathsf{Sub}\,\Gamma\,\Gamma$
idl : $\mathsf{id} \circ \gamma = \gamma$
idr : $\gamma \circ \mathsf{id} = \gamma$
$\diamond$: Con
ϵ : $\mathsf{Sub}\,\Gamma\,\diamond$
$\diamond\eta$: $(\sigma : \mathsf{Sub}\,\Gamma\,\diamond) \to \sigma = \epsilon$
Ty : $\mathsf{Con} \to \mathsf{Set}$
$-[-]$: $\mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta$
$[\circ]$: $A[\gamma \circ \delta] = A[\gamma][\delta]$
$[\mathsf{id}]$: $A[\mathsf{id}] = A$
Tm : $(\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$
$-[-]$: $\mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma])$
$[\circ]$: $a[\gamma \circ \delta] = a[\gamma][\delta]$
$[\mathsf{id}]$: $a[\mathsf{id}] = a$
$- \triangleright -$: $(\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$
$-, -$: $(\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]) \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A)$
p : $\mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$
q : $\mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])$
$\triangleright\beta_1$: $\mathsf{p} \circ (\gamma, a) = \gamma$
$\triangleright\beta_2$: $\mathsf{q}[\gamma, a] = a$
$\triangleright\eta$: $\sigma = (\mathsf{p} \circ \sigma, \mathsf{q}[\sigma])$

Exercise 103. *Define the syntax of CwF without using quotients or QIITs. This involves proving its induction principle.*

Exercise 104 (very very long)**.** *Show that any GAT (element of* Ty *in the syntax of Definition 85) gives rise to a CwF of models. Hint: define a first-order model of Definition 85 where* Con *is CwF. For checking your solution, see [KKA19, Kov22].*

The unoptimised first-order version of STLC is the following.

Definition 105 (First-order model of STLC (unoptimised version), see Definition 41 and Example 89)**.** *We extend CwF (Definition 102) with the following components.*

$$
\begin{array}{ll}
\iota & : \mathsf{Ty}\,\Gamma \\
\iota[] & : \iota[\gamma] = \iota \\
- \Rightarrow - & : \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \\
\Rightarrow[] & : (A \Rightarrow B)[\gamma] = (A[\gamma]) \Rightarrow (B[\gamma]) \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,(B[\mathsf{p}]) \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \\
\mathsf{lam}[] & : (\mathsf{lam}\,b)[\gamma] = \mathsf{lam}\,(b[\gamma \circ \mathsf{p}, \mathsf{q}])
\end{array}
\qquad
\begin{array}{ll}
- \cdot - & : \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,B \\
\cdot[] & : (f \cdot a)[\gamma] = (f[\gamma]) \cdot (a[\gamma]) \\
\beta & : \mathsf{lam}\,b \cdot a = b[\mathsf{id}, a] \\
\eta & : f = \mathsf{lam}\,(f[\mathsf{p}] \cdot \mathsf{q}) \\
[\mathsf{id}] & : a[\mathsf{id}] = a
\end{array}
$$

The unoptimised version of STLC has more models than the optimised version, but the syntaxes are equivalent.

TODO: different versions of application.

Definition 106 (First-order model of mini-MLTT, see Definition 79)**.** *We extend CwF (Definition 102) with the following components.*

$$
\begin{array}{ll}
\iota & : \mathsf{Ty}\,\Gamma \\
\iota[] & : \iota[\gamma] = \iota \\
\Pi & : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma \\
\Pi[] & : (\Pi\,A\,B)[\gamma] = \Pi\,(A[\gamma])\,(B[\gamma \circ \mathsf{p}, \mathsf{q}]) \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \\
\mathsf{lam}[] & : (\mathsf{lam}\,b)[\gamma] = \mathsf{lam}\,(b[\gamma \circ \mathsf{p}, \mathsf{q}])
\end{array}
\qquad
\begin{array}{ll}
- \cdot - & : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\mathsf{id}, a]) \\
\cdot[] & : (f \cdot a)[\gamma] = (f[\gamma]) \cdot (a[\gamma]) \\
\beta & : \mathsf{lam}\,b \cdot a = b[\mathsf{id}, a] \\
\eta & : f = \mathsf{lam}\,(f[\mathsf{p}] \cdot \mathsf{q}) \\
[\mathsf{id}] & : a[\mathsf{id}] = a
\end{array}
$$

Exercise 107. *Show that the equations* $\cdot[]$, β, η *make sense (the two sides are in the same set (meta type)).*

Exercise 108 (long)**.** *What are the extra equations that we need to add to a CwF with* Π, Σ, $\top$ *to obtain something equivalent with cartesian closed categories? (CCC)*

First-order logic is interesting because it has two different kinds of variables: term and proof variables, so there are two context extensions corresponding to these (there are no formula-variables). The only optimisation that we do is to omit equations for Pf which all hold by irr.

Definition 109 (First-order model of minimal intuitionistic first-order logic, see Definition 73)**.**

$\mathsf{Con} : \mathsf{Set}$
$\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$
$- \circ - : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma$
$\mathsf{ass} : (\gamma \circ \delta) \circ \theta = \gamma \circ (\delta \circ \theta)$
$\mathsf{id} : \mathsf{Sub}\,\Gamma\,\Gamma$
$\mathsf{idl} : \mathsf{id} \circ \gamma = \gamma$
$\mathsf{idr} : \gamma \circ \mathsf{id} = \gamma$
$\diamond : \mathsf{Con}$
$\epsilon : \mathsf{Sub}\,\Gamma\,\diamond$
$\diamond\eta : (\sigma : \mathsf{Sub}\,\Gamma\,\diamond) \to \sigma = \epsilon$
$\mathsf{For} : \mathsf{Con} \to \mathsf{Set}$
$-[-] : \mathsf{For}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{For}\,\Delta$
$[\circ] : A[\gamma \circ \delta] = A[\gamma][\delta]$
$[\mathsf{id}] : A[\mathsf{id}] = A$
$\mathsf{Tm} : \mathsf{Con} \to \mathsf{Set}$
$-[-] : \mathsf{Tm}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta$
$[\circ] : t[\gamma \circ \delta] = t[\gamma][\delta]$
$[\mathsf{id}] : t[\mathsf{id}] = t$
$-\triangleright_{\mathsf{Tm}} : \mathsf{Con} \to \mathsf{Con}$
$- ,_{\mathsf{Tm}} - : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright_{\mathsf{Tm}})$
$\mathsf{p}_{\mathsf{Tm}} : \mathsf{Sub}\,(\Gamma \triangleright_{\mathsf{Tm}})\,\Gamma$
$\mathsf{q}_{\mathsf{Tm}} : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Tm}})$
$\triangleright_{\mathsf{Tm}}\beta_1 : \mathsf{p}_{\mathsf{Tm}} \circ (\gamma ,_{\mathsf{Tm}} t) = \gamma$
$\triangleright_{\mathsf{Tm}}\beta_2 : \mathsf{q}_{\mathsf{Tm}}[\gamma ,_{\mathsf{Tm}} t] = t$
$\triangleright_{\mathsf{Tm}}\eta : \sigma = (\mathsf{p} \circ \sigma, \mathsf{q}[\sigma])$
$- \supset - : \mathsf{For}\,\Gamma \to \mathsf{For}\,\Gamma \to \mathsf{For}\,\Gamma$
$\supset[] : (A \supset B)[\gamma] = (A[\gamma]) \supset (B[\gamma])$
$\forall : \mathsf{For}\,(\Gamma \triangleright_{\mathsf{Tm}}) \to \mathsf{For}\,\Gamma$
$\forall[] : (\forall\,A)[\gamma] = \forall\,(A[\gamma \circ \mathsf{p}_{\mathsf{Tm}} ,_{\mathsf{Tm}} \mathsf{q}_{\mathsf{Tm}}])$
$\mathsf{Eq} : \mathsf{Tm}\,\Gamma \to \mathsf{Tm}\,\Gamma \to \mathsf{For}\,\Gamma$
$\mathsf{Eq}[] : (\mathsf{Eq}\,t\,t')[\gamma] = \mathsf{Eq}\,(t[\gamma])\,(t'[\gamma])$
$\mathsf{Pf} : (\Gamma : \mathsf{Con}) \to \mathsf{For}\,\Gamma \to \mathsf{Set}$
$-[-] : \mathsf{Pf}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Pf}\,\Delta\,(A[\gamma])$
$\mathsf{irr} : (p\,q : \mathsf{Pf}\,\Gamma\,A) \to p = q$
$- \triangleright_{\mathsf{Pf}} - : (\Gamma : \mathsf{Con}) \to \mathsf{For}\,\Gamma \to \mathsf{Con}$
$- ,_{\mathsf{Pf}} - : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Pf}\,\Delta\,(A[\gamma]) \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright_{\mathsf{Pf}} A)$
$\mathsf{p}_{\mathsf{Pf}} : \mathsf{Sub}\,(\Gamma \triangleright_{\mathsf{Pf}} A)\,\Gamma$
$\mathsf{q}_{\mathsf{Pf}} : \mathsf{Pf}\,(\Gamma \triangleright_{\mathsf{Pf}} A)\,(A[\mathsf{p}_{\mathsf{Pf}}])$
$\triangleright_{\mathsf{Pf}}\beta_1 : \mathsf{p}_{\mathsf{Pf}} \circ (\gamma ,_{\mathsf{Pf}} p) = \gamma$
$\triangleright_{\mathsf{Pf}}\eta : \sigma = (\mathsf{p}_{\mathsf{Pf}} \circ \sigma ,_{\mathsf{Pf}} \mathsf{q}_{\mathsf{Pf}}[\sigma])$
$\mathsf{intro}_{\supset} : \mathsf{Pf}\,(\Gamma \triangleright_{\mathsf{Pf}} A)\,(B[\mathsf{p}_{\mathsf{Pf}}]) \to \mathsf{Pf}\,\Gamma\,(A \supset B)$
$\mathsf{elim}_{\supset} : \mathsf{Pf}\,\Gamma\,(A \supset B) \to \mathsf{Pf}\,\Gamma\,A \to \mathsf{Pf}\,\Gamma\,B$
$\mathsf{intro}_{\forall} : \mathsf{Pf}\,(\Gamma \triangleright_{\mathsf{Tm}})\,A \to \mathsf{Pf}\,\Gamma\,(\forall\,A)$
$\mathsf{elim}_{\forall} : \mathsf{Pf}\,\Gamma\,(\forall\,A) \to (t : \mathsf{Tm}\,\Gamma) \to \mathsf{Pf}\,\Gamma\,(A[\mathsf{id} ,_{\mathsf{Tm}} t])$
$\mathsf{intro}_{\mathsf{Eq}} : (t : \mathsf{Tm}\,\Gamma) \to \mathsf{Pf}\,\Gamma\,(\mathsf{Eq}\,t\,t)$
$\mathsf{elim}_{\mathsf{Eq}} : \mathsf{Pf}\,\Gamma\,(A[\mathsf{id}, t]) \to \mathsf{Pf}\,(\mathsf{Eq}\,t\,t') \to \mathsf{Pf}\,\Gamma\,(A[\mathsf{id}, t'])$

The syntax of monoids over A is definable without quotients, see Exercise 38. The syntax of STLC is definable without quotients using normal forms and proving normalisation via hereditary substitution [KA10]. Definable quotients were studied by Nuo Li [Li15].

Exercise 110 (long)**.** *Show that the syntax of Definition 109 can be defined in Agda/Coq without quotients; we need* Pf *to be in the universe of propositions to justify the only equation in (the second-order version of) this theory.*

Quotients not definable via normal forms are e.g. the syntax of untyped combinator calculus or lambda calculus. The simplest example is unordered pairs of a given type A. There are theories where it is open whether they are definable without quotients:

Food for thought 111. *First-order logic is a theory without equations, and its first-order syntax is definable. In objective type theory [vdBdB21] (also called weak type theory [BW19]) there are also no equations, the equations are expressed as elements of the* Id *type. Although the second-order version of this theory does not have any equations, it is open whether its first-order syntax is definable without quotients.*

System F is interesting because there are two different kind of variables: type and term variables.

Definition 112 (First-order model of System F, see Definition 76)**.** *We extend CwF (Definition 102) with the following components.*

$-\triangleright_{\mathsf{Ty}} : \mathsf{Con} \to \mathsf{Con}$
$- ,_{\mathsf{Ty}} - : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright_{\mathsf{Ty}})$
$\mathsf{p}_{\mathsf{Ty}} : \mathsf{Sub}\,(\Gamma \triangleright_{\mathsf{Ty}})\,\Gamma$
$\mathsf{q}_{\mathsf{Ty}} : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Ty}})\,(A[\mathsf{p}_{\mathsf{Ty}}])$
$\triangleright_{\mathsf{Ty}}\beta_1 : \mathsf{p}_{\mathsf{Ty}} \circ (\gamma ,_{\mathsf{Ty}} A) = \gamma$
$\triangleright_{\mathsf{Ty}}\beta_2 : \mathsf{q}_{\mathsf{Ty}}[\gamma ,_{\mathsf{Ty}} A] = A$
$\triangleright_{\mathsf{Ty}}\eta : \sigma = (\mathsf{p}_{\mathsf{Ty}} \circ \sigma ,_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}[\sigma])$
$\forall : \mathsf{Ty}\,(\Gamma \triangleright_{\mathsf{Ty}}) \to \mathsf{Ty}\,\Gamma$
$\forall[] : (\forall\,A)[\gamma] = \forall\,(A[\gamma \circ \mathsf{p}_{\mathsf{Ty}} ,_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}])$
$\mathsf{Lam} : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Ty}})\,A \to \mathsf{Tm}\,\Gamma\,(\forall\,A)$
$\mathsf{Lam}[] : (\mathsf{Lam}\,a)[\gamma] = \mathsf{Lam}\,(a[\gamma \circ \mathsf{p}_{\mathsf{Ty}} ,_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}])$
$- \bullet - : \mathsf{Tm}\,\Gamma\,(\forall\,A) \to (B : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,(A[\mathsf{id} ,_{\mathsf{Ty}} B])$
$\bullet[] : (t \bullet B)[\gamma] = (t[\gamma]) \bullet (B[\gamma])$
$\forall\beta : (\mathsf{Lam}\,a) \bullet B = a[\mathsf{id} ,_{\mathsf{Ty}} B]$
$\forall\eta : t = \mathsf{Lam}\,(t[\mathsf{p}_{\mathsf{Ty}}] \bullet \mathsf{q}_{\mathsf{Ty}})$

In System F_ω, we still have two extensions, but type variables are now indexed with their kinds, so their context extension operation is more interesting. We could optimise the following definition such that Kind does not depend on Con. We write the universal properties in concise form.

Definition 113 (First-order model of System F_ω, see Definition 77)**.**

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
- \circ - & : \mathsf{Sub}\ \Delta\ \Gamma \to \mathsf{Sub}\ \Theta\ \Delta \to \mathsf{Sub}\ \Theta\ \Gamma \\
\mathsf{ass} & : (\gamma \circ \delta) \circ \theta = \gamma \circ (\delta \circ \theta) \\
\mathsf{id} & : \mathsf{Sub}\ \Gamma\ \Gamma \\
\mathsf{idl} & : \mathsf{id} \circ \gamma = \gamma \\
\mathsf{idr} & : \gamma \circ \mathsf{id} = \gamma \\
\diamond & : \mathsf{Con} \\
\epsilon & : \mathsf{Sub}\ \Gamma\ \diamond \\
\diamond\eta & : (\sigma : \mathsf{Sub}\ \Gamma\ \diamond) \to \sigma = \epsilon \\
\mathsf{Kind} & : \mathsf{Con} \to \mathsf{Set} \\
-[-] & : \mathsf{Kind}\ \Gamma \to \mathsf{Sub}\ \Delta\ \Gamma \to \mathsf{Kind}\ \Delta \\
[\circ] & : K[\gamma \circ \delta] = K[\gamma][\delta] \\
[\mathsf{id}] & : K[\mathsf{id}] = K \\
\mathsf{Ty} & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Set} \\
-[-] & : \mathsf{Ty}\ \Gamma\ K \to (\gamma : \mathsf{Sub}\ \Delta\ \Gamma) \to \mathsf{Ty}\ \Delta\ (K[\gamma]) \\
[\circ] & : A[\gamma \circ \delta] = A[\gamma][\delta] \\
[\mathsf{id}] & : A[\mathsf{id}] = A \\
-\triangleright_{\mathsf{Ty}} - & (\Gamma : \mathsf{Con}) \to \mathsf{Kind}\ \Gamma \to \mathsf{Con} \\
-,_{\mathsf{Ty}} - & : (\gamma : \mathsf{Sub}\ \Delta\ \Gamma) \times \mathsf{Ty}\ \Delta\ (K[\gamma]) \cong \mathsf{Sub}\ \Delta\ (\Gamma \triangleright_{\mathsf{Ty}} K) : (\mathsf{p}_{\mathsf{Ty}} \circ -, \mathsf{q}_{\mathsf{Ty}}[-]) \\
- \Rrightarrow - & : \mathsf{Kind}\ \Gamma \to \mathsf{Kind}\ \Gamma \to \mathsf{Kind}\ \Gamma \\
\Rrightarrow[] & : (K \Rrightarrow L)[\gamma] = (K[\gamma]) \Rrightarrow (L[\gamma]) \\
\mathsf{LAM} & : \mathsf{Ty}\ (\Gamma \triangleright_{\mathsf{Ty}} K)\ L \to \mathsf{Ty}\ \Gamma\ (K \Rrightarrow L) \\
\mathsf{LAM}[] & : (\mathsf{LAM}\ A)[\gamma] = \mathsf{LAM}\ (A[\gamma \circ \mathsf{p}_{\mathsf{Ty}} ,_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}]) \\
- \bullet - & : \mathsf{Ty}\ \Gamma\ (K \Rrightarrow L) \to \mathsf{Ty}\ \Gamma\ K \to \mathsf{Ty}\ \Gamma\ L \\
\bullet[] & : (F \bullet B)[\gamma] = (F[\gamma]) \bullet (B[\gamma]) \\
\Rrightarrow\beta & : \mathsf{LAM}\ A \bullet B = A[\mathsf{id} ,_{\mathsf{Ty}} B] \\
\Rrightarrow\eta & : F = \mathsf{LAM}\ (F[\mathsf{p}_{\mathsf{Ty}}] \bullet \mathsf{q}_{\mathsf{Ty}}) \\
* & : \mathsf{Kind}\ \Gamma \\
*[] & : *[\gamma] = * \\
\mathsf{Tm} & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma * \to \mathsf{Set} \\
-[-] & : \mathsf{Tm}\ \Gamma\ A \to (\gamma : \mathsf{Sub}\ \Delta\ \Gamma) \to \mathsf{Tm}\ \Delta\ (A[\gamma]) \\
[\circ] & : a[\gamma \circ \delta] = a[\gamma][\delta] \\
[\mathsf{id}] & : a[\mathsf{id}] = a \\
-,_{\mathsf{Tm}} - & : (\gamma : \mathsf{Sub}\ \Delta\ \Gamma) \times \mathsf{Tm}\ \Delta\ (A[\gamma]) \cong \mathsf{Sub}\ \Delta\ (\Gamma \triangleright_{\mathsf{Tm}} A) : (\mathsf{p}_{\mathsf{Tm}} \circ -, \mathsf{q}_{\mathsf{Tm}}[-]) \\
\forall & : \mathsf{Ty}\ (\Gamma \triangleright_{\mathsf{Ty}} K) * \to \mathsf{Ty}\ \Gamma * \\
\forall[] & : (\forall\ A)[\gamma] = \forall\ (A[\gamma \circ \mathsf{p}_{\mathsf{Ty}} \circ_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}]) \\
\mathsf{Lam} & : \mathsf{Tm}\ (\Gamma \triangleright_{\mathsf{Ty}} K)\ A \to \mathsf{Tm}\ \Gamma\ (\forall\ A) \\
\mathsf{Lam}[] & : (\mathsf{Lam}\ a)[\gamma] = \mathsf{Lam}\ (a[\gamma \circ \mathsf{p}_{\mathsf{Ty}} ,_{\mathsf{Ty}} \mathsf{q}_{\mathsf{Ty}}]) \\
- \bullet - & : \mathsf{Tm}\ \Gamma\ (\forall\ A) \to (B : \mathsf{Ty}\ \Gamma\ K) \to \mathsf{Tm}\ \Gamma\ (A[\mathsf{id} ,_{\mathsf{Ty}} B]) \\
\bullet[] & : (t \bullet A)[\gamma] = (t[\gamma]) \bullet (A[\gamma]) \\
\forall\beta & : (\mathsf{Lam}\ a) \bullet B = a[\mathsf{id} ,_{\mathsf{Ty}} B] \\
\forall\eta & : t = \mathsf{Lam}\ (t[\mathsf{p}_{\mathsf{Ty}}] \bullet \mathsf{q}_{\mathsf{Ty}}) \\
- \Rightarrow - & : \mathsf{Ty}\ \Gamma * \to \mathsf{Ty}\ \Gamma * \to \mathsf{Ty}\ \Gamma * \\
\Rightarrow[] & : (A \Rightarrow B)[\gamma] = (A[\gamma]) \Rightarrow (B[\gamma]) \\
\mathsf{lam} & : \mathsf{Tm}\ (\Gamma \triangleright_{\mathsf{Tm}} A)\ B \to \mathsf{Tm}\ \Gamma (A \Rightarrow B) \\
\mathsf{lam}[] & : (\mathsf{lam}\ b)[\gamma] = \mathsf{lam}\ (b[\gamma \circ \mathsf{p}_{\mathsf{Tm}} ,_{\mathsf{Tm}} \mathsf{q}_{\mathsf{Tm}}]) \\
- \cdot - & : \mathsf{Tm}\ \Gamma\ (A \Rightarrow B) \to \mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \Gamma\ B \\
\cdot[] & : (t \cdot a)[\gamma] = (t[\gamma]) \cdot (a[\gamma]) \\
\Rightarrow\beta & : (\mathsf{lam}\ b) \cdot a = b[\mathsf{id}, a] \\
\Rightarrow\eta & : t = \mathsf{lam}\ (t[\mathsf{p}_{\mathsf{Tm}}] \cdot \mathsf{q}_{\mathsf{Tm}})
\end{array}
$$

Exercise 114. *Check that instantiation for terms make sense (is well-typed in the metatheory).*

5 Admissibility in SOGATs

It is not a bad position to define languages abstractly as SOGATs, and then when proving properties of the language, we have to work with its more verbose translated GAT-version. It is nice that all the substitution rules are generated automatically, and we don't have to worry that we missed one or wrote down one incorrectly.

However, we can do better. Some proofs about first-order models can be factored through a generic sconing construction.

In this section, we fix the SOGAT to mini-MLTT: its second-order model is Definition 79, its first-order model is Definition 106.

5.1 Proofs about closed syntactic terms

I denotes the first-order model of mini-MLTT.

Definition 115 (Second-order dependent model)**.**

$$
\begin{aligned}
\mathsf{Ty} &: \mathsf{Ty}_\mathsf{I}\,\diamond_\mathsf{I} \to \mathsf{Set}\\
\mathsf{Tm} &: \{A_\mathsf{I} : \mathsf{Ty}_\mathsf{I}\,\diamond_\mathsf{I}\} \to \mathsf{Ty}\,A_\mathsf{I} \to \mathsf{Tm}_\mathsf{I}\,\diamond_\mathsf{I} A_\mathsf{I} \to \mathsf{Set}\\
\iota &: \mathsf{Ty}\,(\iota_\mathsf{I}\,\{\diamond_\mathsf{I}\})\\
\Pi &: \{A_\mathsf{I} : \mathsf{Ty}_\mathsf{I}\,\diamond_\mathsf{I}\}(A : \mathsf{Ty}\,A_\mathsf{I})\{B_\mathsf{I} : \mathsf{Ty}_\mathsf{I}\,(\diamond_\mathsf{I} \triangleright_\mathsf{I} A_\mathsf{I})\} \to (\{a_\mathsf{I} : \mathsf{Tm}_\mathsf{I}\,\diamond_\mathsf{I} A_\mathsf{I}\} \to \mathsf{Tm}\,A_\mathsf{I}\,a_\mathsf{I} \to \mathsf{Ty}\,(B_\mathsf{I}[\epsilon_\mathsf{I} ,_\mathsf{I} a_\mathsf{I}]_\mathsf{I})) \to \mathsf{Ty}\,(\Pi_\mathsf{I}\,A_\mathsf{I}\,B_\mathsf{I})\\
\mathsf{lam} &: \big((a : \mathsf{Tm}\,A\,a_\mathsf{I}) \to \mathsf{Tm}\,(B\,a)\,(b_\mathsf{I}[\epsilon_\mathsf{I} ,_\mathsf{I} a_\mathsf{I}]_\mathsf{I})\big) \to \mathsf{Tm}\,(\Pi\,A\,B)\,(\mathsf{lam}_\mathsf{I}\,b_\mathsf{I})\\
- \cdot - &: \mathsf{Tm}\,(\Pi\,A\,B)\,f_\mathsf{I} \to (a : \mathsf{Tm}\,A\,a_\mathsf{I}) \to \mathsf{Tm}\,(B\,a)\,(f_\mathsf{I} \cdot_\mathsf{I} a_\mathsf{I})\\
\beta &: \mathsf{lam}\,b \cdot a = b\,a\\
\eta &: f = \mathsf{lam}\,\lambda a.\, f \cdot a
\end{aligned}
$$

The left hand side of β has type $\mathsf{Tm}\,(B\,a)\,(\mathsf{lam}_\mathsf{I}\,b_\mathsf{I} \cdot_\mathsf{I} a_\mathsf{I})$, the right hand side has type $\mathsf{Tm}\,(B\,a)\,(b_\mathsf{I}[\epsilon_\mathsf{I} ,_\mathsf{I} a_\mathsf{I}]_\mathsf{I})$, which are equal by β_I.

Exercise 116. *Similarly check that the two sides of η are in the same set.*

The above dependent model only says something about closed types and closed terms. We have a general construction to make it work for any type and term:

Definition 117 (Scone-contextualisation)**.** *Assuming a second-order dependent model, we obtain the following first-order dependent model.*

$$
\begin{aligned}
\mathsf{Con}\,\Gamma_\mathsf{I} &:= \mathsf{Sub}_\mathsf{I}\,\diamond_\mathsf{I}\,\Gamma_\mathsf{I} \to \mathsf{Set}\\
\mathsf{Sub}\,\Delta\,\Gamma\,\gamma_\mathsf{I} &:= \Delta\,\delta_\mathsf{I} \to \Gamma\,(\gamma_\mathsf{I} \circ_\mathsf{I} \delta_\mathsf{I})\\
\delta \circ \gamma &:= \lambda\theta_*.\ \delta\,(\gamma\,\theta_*)\\
\mathsf{id} &:= \lambda\gamma_*.\ \gamma_*\\
\diamond &:= \lambda\_.\ \mathbb{1}\\
\mathsf{Ty}\,\Gamma\,A_\mathsf{I} &:= (\gamma_* : \Gamma\,\gamma_\mathsf{I}) \to \mathsf{Ty}\,(A_\mathsf{I}[\gamma_\mathsf{I}]_\mathsf{I})\\
A[\gamma] &:= \lambda\delta_*.\ A\,(\gamma\,\delta_*)\\
\mathsf{Tm}\,\Gamma\,A\,a_\mathsf{I} &:= (\gamma_* : \Gamma\,\gamma_\mathsf{I}) \to \mathsf{Tm}\,(A\,\gamma_*)\,(a_\mathsf{I}[\gamma_\mathsf{I}]_\mathsf{I})\\
a[\gamma] &:= \lambda\delta_*.\ a\,(\gamma\,\delta_*)\\
\Gamma \triangleright A &:= \lambda(\gamma_\mathsf{I} ,_\mathsf{I} a_\mathsf{I}).\ (\gamma_* : \Gamma\,\gamma_\mathsf{I}) \times \mathsf{Tm}\,(A\,\gamma_\mathsf{I})\,a_\mathsf{I}\\
\gamma, a &:= \lambda\delta_*.\ (\gamma\,\delta_*, a\,\delta_*)\\
\mathsf{p} &:= \lambda(\gamma_*, a_*).\ \gamma_*\\
\mathsf{q} &:= \lambda(\gamma_*, a_*).\ a_*\\
\Pi\,A\,B &:= \lambda\gamma_*.\ \Pi\,(A\,\gamma_*)\,(\lambda a_*.\ B\,(\gamma_*, a_*))\\
\mathsf{lam}\,b &:= \lambda\gamma_*.\ \mathsf{lam}\,(\lambda a_*.\ b\,(\gamma_*, a_*))\\
f \cdot a &:= \lambda\gamma_*.\ f\,\gamma_* \cdot a\,\gamma_*
\end{aligned}
$$

Example 118 (Canonicity for mini-MLTT)**.** *We define the following second-order dependent model:*

$$\begin{aligned}
\mathsf{Ty}\, A_\mathsf{I} &:= \mathsf{Tm}_\mathsf{I} \diamond_\mathsf{I} A_\mathsf{I} \to \mathsf{Set}\\
\mathsf{Tm}\, A\, a_\mathsf{I} &:= A\, a_\mathsf{I}\\
\iota &:= \lambda t_\mathsf{I}.\, \mathbb{0}\\
\Pi\, A\, B &:= \lambda f_\mathsf{I}.\, \{a_\mathsf{I} : \mathsf{Tm}_\mathsf{I} \diamond_\mathsf{I} A_\mathsf{I}\}(a_* : A\, a_\mathsf{I}) \to B\, a_*\, (f_\mathsf{I} \cdot_\mathsf{I} a_\mathsf{I})\\
\mathsf{lam}\, b &:= b\\
f \cdot a &:= f\, a
\end{aligned}$$

This is the canonicity proof without boilerplate. From this, we get e.g. that the set $\mathsf{Tm}_\mathsf{I} \diamond_\mathsf{I} \iota_\mathsf{I}$ *is empty: applying (induction into the dependent scone-contextualisation of the canonicity second-order dependent model) on such a* t_I *we obtain an element of* $(\sigma_* : \diamond \sigma_\mathsf{I}) \to \mathsf{Tm}\, (\iota\, \sigma_*)\, (t_\mathsf{I}[\sigma_\mathsf{I}]_\mathsf{I}) = \mathbb{1} \to \mathbb{0}$*.*

5.2 Internal languages of presheaf models of MLTT

If we say that a model of a SOGAT is a model of its first-orderification, then did the derived things in Section 3 made sense?

How do we know that the SOGAT $\longrightarrow$ GAT translation is the correct one? One requirement is that they can be used to build the same things: if we build something assuming a second-order model, then the corresponding first-order thing should be also buildable using De Bruijn combinators and explicit weakenings etc. How do we make this sure?

The derivable operations and equations of Section 3 are also derivable in the result of the translations.

5.3 Proofs about open syntactic terms

Renamings.

Internal to presheaves over renamings, we have a first-order model which we also denote I.

References

[ACKS23] Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *Math. Struct. Comput. Sci.*, 33(8):688–743, 2023. URL: `https://doi.org/10.1017/s0960129523000130`, `doi:10.1017/S0960129523000130`.

[AKSV23] Thorsten Altenkirch, Ambrus Kaposi, Artjoms Sinkarovs, and Tamás Végh. Combinatory logic and lambda calculus are equal, algebraically. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: `https://doi.org/10.4230/LIPIcs.FSCD.2023.24`, `doi:10.4230/LIPICS.FSCD.2023.24`.

[Bar91] Henk Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):125–154, 1991. `doi:10.1017/S0956796800020025`.

[BKS23] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. For the metatheory of type theory, internal sconing is enough. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 18:1–18:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.FSCD.2023.18`.

[BW19] Simon Boulier and Théo Winterhalter. Weak type theory is rather strong. In Marc Bezem, editor, *25th International Conference on Types for Proofs and Programs, TYPES 2019*. Centre for Advanced Study at the Norwegian Academy of Science and Letters, 2019. URL: `https://www.ii.uib.no/~bezem/abstracts/TYPES_2019_paper_18`.

[HHP93] Robert Harper, Furio Honsell, and Gordon D. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, 1993. `doi:10.1145/138027.138060`.

[Hof99] Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 204–213. IEEE Computer Society, 1999. `doi:10.1109/LICS.1999.782616`.

[Hut23] Graham Hutton. Programming Language Semantics: It's Easy As 1,2,3. *Journal of Functional Programming*, 33, October 2023.

[KA10] Chantal Keller and Thorsten Altenkirch. Hereditary substitutions for simple types, formalized. In Venanzio Capretta and James Chapman, editors, *Proceedings of the 3rd ACM SIGPLAN Workshop on Mathematically Structured Functional Programming, MSFP@ICFP 2010, Baltimore, MD, USA, September 25, 2010*, pages 3–10. ACM, 2010. doi:10.1145/1863597.1863601.

[Kap19] Ambrus Kaposi. Reduction of indexed w-types to w-types, an agda formalisation. https://akaposi.github.io/IW.agda, 2019.

[KKA19] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, 2019. doi:10.1145/3290315.

[Kov22] András Kovács. *Type-Theoretic Signatures for Algebraic Theories and Inductive Types*. PhD thesis, Eötvös Loránd University, Hungary, 2022. URL: https://arxiv.org/pdf/2302.08837.pdf.

[KvR20] Ambrus Kaposi and Jakob von Raumer. A syntax for mutual inductive families. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 23:1–23:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL: https://doi.org/10.4230/LIPIcs.FSCD.2020.23, doi:10.4230/LIPICS.FSCD.2020.23.

[KX24] Ambrus Kaposi and Szumi Xie. Second-order generalised algebraic theories: Signatures and first-order semantics. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction, FSCD 2024, July 10-13, 2024, Tallinn, Estonia*, volume 299 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.FSCD.2024.10, doi:10.4230/LIPICS.FSCD.2024.10.

[Li15] Nuo Li. *Quotient types in type theory*. Thesis. University of Nottingham, Department of Computer Science, 2015. URL: http://eprints.nottingham.ac.uk/28941.

[Sch24] Moses Schönfinkel. Über die bausteine der mathematischen logik. *Mathematische Annalen*, 92(3):305–316, Sep 1924. doi:10.1007/BF01448013.

[Ste22] Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, Carnegie Mellon University, USA, 2022. URL: https://doi.org/10.1184/r1/19632681.v1, doi:10.1184/R1/19632681.V1.

[Tai67] William W. Tait. Intensional interpretations of functionals of finite type I. *J. Symb. Log.*, 32(2):198–212, 1967. doi:10.2307/2271658.

[vdBdB21] Benno van den Berg and Martijn den Besten. Quadratic type checking for objective type theory. *CoRR*, abs/2102.00905, 2021. URL: https://arxiv.org/abs/2102.00905, arXiv:2102.00905.