Second-order generalised algebraic theories: signatures and first-order semantics

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary

Szumi Xie
Eötvös Loránd University, Budapest, Hungary

Abstract

Programming languages can be defined from the concrete to the abstract by abstract syntax trees, well-scoped syntax, well-typed (intrinsic) syntax, algebraic syntax (well-typed syntax quotiented by conversion). Another aspect is the representation of binding structure for which nominal approaches, De Bruijn indices/levels and higher order abstract syntax (HOAS) are available. In HOAS, binders are given by the function space of an internal language of presheaves. In this paper, we show how to combine the algebraic approach with the HOAS approach: following Uemura, we define languages as second-order generalised algebraic theories (SOGATs). Through a series of examples we show that non-substructural languages can be naturally defined as SOGATs. We give a formal definition of SOGAT signatures (using the syntax of a particular SOGAT) and define two translations from SOGAT signatures to GAT signatures (signatures for quotient inductive-inductive types), based on parallel and single substitutions, respectively.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Type theory, universal algebra, inductive types, quotient inductive types, higher-order abstract syntax, logical framework

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

The traditional way of defining a programming language comprises of a BNF-style description of abstract syntax trees, a typing relation and a reduction or conversion relation [46, 47, 51]. If instead the syntax is defined using well-scoped syntax trees [33, 26, 3], bound names do not matter: for example, one cannot distinguish $\lambda x.x$ and $\lambda y.y$ anymore. A higher level representation is given by intrinsic (well-typed) terms [9, 51] where one merges the syntax and the typing relation: non well-typed terms are not expressable in such a representation. The next level of abstraction is when well-typed terms are quotiented by the conversion relation: this is especially convenient for dependently typed languages where typing depends on conversion [7]. Here one can only define functions on the syntax that preserve conversion: a simple printing function is not definable, but normalisation [6, 19], typechecking [34] or parametricity [7] preserve conversion, so they can be defined on the well-typed quotiented syntax. The well-typed quotiented syntax is also concordant with the semantics: there is no reason to have a separate definition of syntax and a different notion of semantics, but the syntax can be simply defined as the initial model, which always exists for any generalised algebraic theory (GAT) [38]. Thus, abstractly, a language is simply a GAT.

Another aspect of the definition of a language is the treatment of bindings and variables: one can use De Bruijn indices to make sure that choices of names do not matter, but then substitution has to be part of the syntax, for example in the form of a category with families [18]. Logical frameworks [28, 45] and higher-order abstract syntax (HOAS) [31] provide another way to implement bindings and variables: they use the function space of the metatheory. For example, the type of the lambda operation in the pure lambda calculus is simply the second-order function space $(\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm}$. The justification of HOAS is the type-theoretic internal language of presheaves over the category of contexts and syntactic substitutions [31]. In this internal language, lambda indeed has the above type. This internal language viewpoint can also be used to *define* languages: in this case a language with bindings is not a GAT, but a second-order generalised algebraic theory (SOGAT), which allows second-order (but not general higher-order) operations. While untyped or simply typed languages were defined as second-order theories before [22, 20, 2], SOGATs were first used by Uemura [50] for defining languages with bindings. The step from second-order algebraic theories to SOGATs is a big one: it is analogous to the step from inductive types to inductive-inductive types [39], which is difficult, e.g. inductive-inductive types are still not supported by Coq. The SOGAT definition of a language can be even more abstract than the well-typed quotiented definition: the SOGAT does not mention contexts or substitutions: these can be seen as boilerplate that should be automatically generated. SOGATs are not well-behaved algebraic theories, for example, there is no meaningful notion of homomorphism of second-order models. To describe first order models, homomorphisms or the notion of syntax for a SOGAT, we turn it into a GAT. In this process we introduce new sorts for contexts and substitutions, we index every operation with its context, and the second-order function spaces become first order using this context indexing. The thus obtained GAT has some "correctness by construction" properties, for example, every operation automatically preserves substitution. For complicated theories, this property is not trivial if we do not start from a SOGAT, but try to work with the lower level GAT presentation directly. Cubical type theory [49] and a type theory with internal parametricity [5] have been presented as SOGATs, and methods were developed to prove properties of type theories at the SOGAT level of abstraction [48, 15]. Substructural (e.g. linear or modal) type theories are not definable as SOGATs using the method described in this paper, but sometimes the internal language of presheaves over a substructural theory provides a substructural internal language which can be used to describe the theory as in the case of multi-modal type theory [25].

Simple algebraic theories can be presented using signatures and equations, or presentation-independently as Lawvere theories. GATs have syntactic signatures defined using preterms and well-formedness relations [17], and they can be described presentation-independently as contextual categories [17], categories with families (CwFs) or clans [23]. The "theory of signatures" (ToS) approach [38] is halfway between the syntactic and presentation-independent approaches: here signatures are defined by the syntax of a particular GAT, which is a domain-specific type theory designed for defining signatures. Signatures look exactly as we write inductive datatype definitions in a proof assistant like Agda: a list (telescope) of the curried types of sorts and constructors. A signature in the ToS is a concrete presentation of a theory, but it is given at the level of abstraction of well-typed quotiented syntax. This allows elegant semantic constructions [41], while still working directly with signatures. SOGATs again can be defined syntactically [50] or presentation-independently as representable map categories [50] or CwFs with locally representable types [13], and the current paper contributes the ToS style definition of SOGATs. In fact, the theory of SOGAT signatures is itself a SOGAT which can describe itself. Circularity is avoided because we bootstrap the theory of SOGAT signatures by first defining it as a GAT, and the theory of GAT signatures (which is the syntax of a GAT) can itself be bootstrapped using a Church-encoding [40].

Contributions. The main takeaway of this paper is that structural languages are SOGATs. We justify this claim through several examples. Our technical contributions are the following:

- The theory of SOGAT signatures (ToS^{+}), a domain-specific type theory in which every closed type is a SOGAT signature. As it is a structural type theory, it can be defined as a SOGAT itself. Signatures can be formalised in ToS^+ without encoding overhead.

- A translation from SOGAT signatures to GAT signatures based on a parallel substitution calculus. Thus, for every SOGAT, we obtain all of the semantics of GATs: a category of models with an initial object, (co)free models, notions of displayed models and sections, the fact that induction is equivalent to initiality, and so on. The GAT descriptions that we obtain are readable, do not contain occurrences of Yoneda as in usual presheaf function spaces. Correctness of the translation is showed by proving that internally to presheaves over a model of the GAT, a second-order model of the SOGAT is available.
- We define an alternative translation producing a single substitution calculus.

Structure of the paper. In Section 2, we walk through examples of languages defined as second-order algebraic theories (SOGATs) including (simply typed) combinator calculus, (simply typed) lambda calculus, first-order logic, Martin-Löf type theory. We list more examples in Appendix A. We explain what the SOGAT → GAT translation will give for each example. In Section 3, we define languages for describing algebraic theories, culminating in the theory of SOGAT signatures (ToS^+). A SOGAT is simply a closed type in the syntax of ToS^+. Then we define the SOGAT → GAT translation in three iterations: Section 4 presents a naive notion of model which is obviously correct, but has lots of encoding overhead. Section 5 defines an isomorphic notion of model with less encoding overhead. The final translation is defined in Section 6. Section 7 discusses open and infinitary signatures, and explains the single substitution calculus variant. Section 8 concludes.

Related work. The "theory of signatures" (ToS) approach was introduced by Kaposi and Kovács [37] for a higher variant of GATs (higher inductive-inductive types), and was used to describe ordinary [38] and infinitary [40] GATs (quotient inductive-inductive types). The thesis of Kovács [41] summarises and generalises these results, in particular, it provides semantics internal to any category with families (CwF) using the semantic setting of two-level type theory [4, 10]. The current paper extends this work with second-order operations. The ToS that we use differs from the one in Kovács' thesis by including Σ types and being presented as a SOGAT itself. This has the advantage that we do not have to deal with De Bruijn indices when giving formal signatures. A version of ToS^+ with two fixed sorts of types and terms was given in the HoTTeST talk by Kaposi [35].

Direct precursors of our work are Hofmann's analysis of higher-order abstract syntax (HOAS) [31] and Capriotti's rule framework [16]. Syntactic definitions of SOGATs are given in Uemura's thesis [50] and Harper's equational logical framework [27]. A syntactic definition of type theories (SOGATs with two fixed sorts: types and terms) is described by Bauer and Haselwater [29] based on earlier work [12]. Presentation-independent definitions of SOGATs are representable map categories by Uemura [50] and CwFs with a sort of locally representable types (CwF^+) [14]. The presentation-independent ways define models using functorial semantics, while the ToS approach defines semantics of GATs by induction on the signature. Functorial semantics for our SOGAT signatures is as follows: every SOGAT signature Ω gives rise to the free CwF^+ over Ω (the slice of the theory of SOGAT signatures over Ω). Now a model is a category $\mathcal{C}$ together with a CwF^+-morphism from this CwF^+ to the CwF^+ of presheaves over $\mathcal{C}$.

Our two different ways of translating SOGATs to GATs roughly correspond to Voevodsky's two different descriptions of the substitution calculus for dependent type theory: B-systems correspond to single substitutions, C-systems to parallel substitutions. B-systems and C-systems are equivalent [1], however our single substitution calculus is more minimalistic, and has more models than the parallel substitution calculus.

In this paper we explain how to define languages as SOGATs and then translate them into GATs. Then, the induction principle of the GAT can be used to prove properties of the syntax. However, certain metatheoretic proofs can be described at the level of SOGATs avoiding mentioning contexts or substitutions. Synthethic Tait computability [48] and internal sconing [15] are techniques for this. We leave adapting them to ToS^{+} as future work.

Metatheory and notation. Our metatheory is extensional type theory with uniqueness of identity proofs, we use Agda-like notation with implicit arguments sometimes omitted. We write function application as juxtaposition, the universe of types is denoted Set_i, we usually omit the level subscripts. We use infix Σ type notation using $\times$, the single element of the singleton type $\mathbb{1}$ is denoted $*$. Sometimes we work in the internal language of a presheaf category using the same notations, in the style of two-level type theory [4, 10].

2 Classes of algebraic theories through examples

In this section, we walk through examples of logic and programming languages defined as algebraic theories: we define a single-sorted algebraic theory (AT), a generalised algebraic theory (GAT), a second-order algebraic theory (SOAT) and multiple second-order generalised algebraic theories (SOGATs). GATs include typing information compared to ATs, SOATs include binders, while SOGATs combine these two aspects.

Algebraic theories. Combinator calculus is an algebraic theory (AT) with a single sort of terms, one binary, two nullary operations and two equations. We denote its signature as follows (unlike usual presentations of algebraic theories, we include the equations in the notion of signature, because for generalised algebraic theories separation is not possible).

▶ **Definition 1** (Schönfinkel's combinator calculus).

$$\begin{array}{llllll}
\mathsf{Tm} & : \mathsf{Set} & \mathsf{K} & : \mathsf{Tm} & \mathsf{K}\beta & : \mathsf{K} \cdot u \cdot f = u \\
- \cdot - & : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm} & \mathsf{S} & : \mathsf{Tm} & \mathsf{S}\beta & : \mathsf{S} \cdot f \cdot g \cdot u = f \cdot u \cdot (g \cdot u)
\end{array}$$

The notion of algebra/model is evident from this signature. The quotiented syntax of combinator calculus is the initial model, which always exists. Notions of homomorphism, displayed/dependent model, induction, products and coproducts of models, free models, and so on, are derivable from the signature, as described in any book on universal algebra. The initial algebra of an AT is called a quotient inductive type [21].

Single-sorted algebraic theories from logic are classical (or intuitionistic) propositional logic defined as the theory of Boolean algebras (or Heyting algebras). Examples from algebra are monoids, groups, rings, lattices, and so on.

Generalised algebraic theories. Generalised algebraic theories (GATs) allow sorts indexed by other sorts. Examples are typed combinator calculus and propositional logic with Hilbert-style proof theory, theories of graphs, preorders, categories, and so on.

▶ **Definition 2** (Typed combinator calculus).

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

We have a sort of types, and for each type, a separate sort of terms of that type. Now the K and S operations are nullary only in the sense that they don't take Tm arguments, but they still take two and three Ty arguments, respectively. For readability, these are given implicitly. Similarly, application $- \cdot -$ takes the arguments A and B implicitly.

The above mentioned universal algebraic features of ATs generalise to GATs [41]. In particular, each GAT has a syntax given as a quotient inductive-inductive type [38], we have free models [41] and cofree models [43]. If the language has variables or binders, we will define it as a second-order theory.

Second-order algebraic theories. The SOAT of lambda calculus is the following.

▶ **Definition 3** (Lambda calculus).

$$\mathsf{Tm} : \mathsf{Set} \qquad \mathsf{lam} : (\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm} \qquad - \cdot - : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm} \qquad \beta : \mathsf{lam}\, f \cdot u = f\, u$$

The type of lam is not first-order (not strictly positive), hence this is not an algebraic theory anymore. It is clear what a second-order model is (a set with a binary operation and a second-order function with the type of lam satisfying the equation β). However, we do not have a usable notion of homomorphism between second-order models M and N: this would be a function $\alpha : \mathsf{Tm}_M \to \mathsf{Tm}_N$ such that $\alpha\,(t \cdot_M u) = \alpha\, t \cdot_N \alpha\, u$ and $\alpha\,(\mathsf{lam}_M\, f) = \mathsf{lam}_N\,(\alpha \circ f \circ ?)$, but we don't know what to put in place of the ?. To talk about homomorphisms or the syntax, we translate the SOAT to a first-order GAT: we add contexts, substitutions, index Tm and all operations by contexts and then lam becomes a first order function taking a term in an extended context as input. The resulting GAT is the following.

▶ **Definition 4** (Lambda calculus as a first-order GAT).

$$\begin{array}{llll}
\mathsf{Con} & : \mathsf{Set} & [\mathsf{id}] & : t[\mathsf{id}] = t \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} & - \triangleright & : \mathsf{Con} \to \mathsf{Con} \\
- \circ - & : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma & -,- & : \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright) \\
\mathsf{ass} & : (\gamma \circ \delta) \circ \theta = \gamma \circ (\delta \circ \theta) & \mathsf{p} & : \mathsf{Sub}\,(\Gamma \triangleright)\,\Gamma \\
\mathsf{id} & : \mathsf{Sub}\,\Gamma\,\Gamma & \mathsf{q} & : \mathsf{Tm}\,(\Gamma \triangleright) \\
\mathsf{idl} & : \mathsf{id} \circ \gamma = \gamma & \triangleright\beta_1 & : \mathsf{p} \circ (\gamma, t) = \gamma \\
\mathsf{idr} & : \gamma \circ \mathsf{id} = \gamma & \triangleright\beta_2 & : \mathsf{q}[\gamma, t] = t \\
\diamond & : \mathsf{Con} & \triangleright\eta & : \sigma = (\mathsf{p} \circ \sigma, \mathsf{q}[\sigma]) \\
\epsilon & : \mathsf{Sub}\,\Gamma\,\diamond & - \cdot - & : \mathsf{Tm}\,\Gamma \to \mathsf{Tm}\,\Gamma \to \mathsf{Tm}\,\Gamma \\
\diamond\eta & : (\sigma : \mathsf{Sub}\,\Gamma\,\diamond) \to \sigma = \epsilon & \cdot[] & : (t \cdot u)[\gamma] = t[\gamma] \cdot (u[\gamma]) \\
\mathsf{Tm} & : \mathsf{Con} \to \mathsf{Set} & \mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright) \to \mathsf{Tm}\,\Gamma \\
-[-] & : \mathsf{Tm}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta & \mathsf{lam}[] & : (\mathsf{lam}\,t)[\gamma] = \mathsf{lam}\,(t[\gamma \circ \mathsf{p}, \mathsf{q}]) \\
[\circ] & : t[\gamma \circ \delta] = t[\gamma][\delta] & \beta & : \mathsf{lam}\,t \cdot u = t[\mathsf{id}, u]
\end{array}$$

In more detail: the GAT starts with a category with a terminal object ($\mathsf{Con}, \ldots, \diamond\eta$), then we have the sort Tm which is now indexed by Con and comes with an instantiation operation $-[-]$ which is functorial. There is a context extension $-\triangleright$ which makes contexts a natural number algebra (with zero $\diamond$ and successor $-\triangleright$). Substitutions are lists of terms, this is expressed by the isomorphism $\mathsf{p} \circ -, \mathsf{q}[-] : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright) \cong \mathsf{Sub}\,\Delta\,\Gamma \times \mathsf{Tm}\,\Delta : -,-$. Now variables are definable as De Bruijn indices: $0 = \mathsf{q}$, $1 = \mathsf{q}[\mathsf{p}]$, $2 = \mathsf{q}[\mathsf{p}][\mathsf{p}]$, and so on. The operations $- \cdot -$ and lam are also (implicitly) indexed by contexts and come equipped with substitution laws ($\cdot[]$ and $\mathsf{lam}[]$). The function in the input of the SOAT presentation of lam

becomes a Tm in an extended context. In $\mathsf{lam}[]$, the substitution $(\gamma \circ \mathsf{p}, \mathsf{q}) : \mathsf{Sub}\,(\Delta \triangleright)\,(\Gamma \triangleright)$ is the lifting of $\gamma : \mathsf{Sub}\,\Delta\,\Gamma$ which does not touch the last variable bound by lam. Finally, the metatheoretic function application on the right hand side of the β law in the SOAT presentation becomes an instantiation of the last variable by $(\mathsf{id}, u) : \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright)$.

In the special case of the lambda calculus, there are equivalent simpler GATs, but this is the one which is generated by the translation of Section 6. Our translation works generically, hence it does not necessarily give the most minimal GAT presentation.

By the syntax of lambda calculus, we mean the syntax for the above GAT. However, we still prefer to define lambda calculus as a SOGAT: it is a shorter definition, does not include boilerplate, and ensures that once translated to its first-order version, all operations respect substitution by construction. Also, we can do programming using the second-order representation in the style of logical frameworks. This means that using the second-order presentation, we can define *derivable* operations and prove derivable equations as opposed to *admissible* ones for which we would need induction. An example of a derivable operation is the Y combinator: we assume a second-order model of the lambda calculus given by Tm, lam, $- \cdot -$, β, and define $\mathsf{Y} := \mathsf{lam}\,\lambda f.\big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big) \cdot \big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big)$. We prove that this is indeed a fixpoint combinator as follows.

$$\begin{aligned}
\mathsf{Y} \cdot f = {} & \Big(\mathsf{lam}\,\lambda f.\big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big) \cdot \big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big)\Big) \cdot f && = (\beta) \\
& \big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big) \cdot \big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big) && = (\beta) \\
& f \cdot \Big(\big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big) \cdot \big(\mathsf{lam}\,\lambda x.f \cdot (x \cdot x)\big)\Big) && = f \cdot (\mathsf{Y} \cdot f)
\end{aligned}$$

This kind of reasoning makes sense for any second-order model, and any first-order model gives rise to a second-order model in the internal language of presheaves over the first-order model, see Corollary 25.

Second-order generalised algebraic theories. SOGATs combine the two previous classes: sorts can be indexed over previous sorts and second-order operations are allowed. In the following examples, we write $\mathsf{f} : \mathsf{A} \leftrightarrow \mathsf{B} : \mathsf{g}$ for $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ and $\mathsf{g} : \mathsf{B} \to \mathsf{A}$, we write $\mathsf{A} \cong \mathsf{B}$ for $\mathsf{A} \leftrightarrow \mathsf{B}$ with two equations $\beta : \mathsf{g}\,(\mathsf{f}\,a) = a$ and $\eta : \mathsf{f}\,(\mathsf{g}\,b) = b$. We write $\mathsf{A} : \mathsf{Prop}$ for $\mathsf{A} : \mathsf{Set}$ together with an equation $\mathsf{irr} : (a\,a' : \mathsf{A}) \to a = a'$. We list the theories as SOGATs, and discuss the interesting aspects of their first-order models.

▶ **Definition 5** (Simply typed lambda calculus)**.**

$$\begin{aligned}
&\mathsf{Ty} \quad\; : \mathsf{Set} && \mathsf{Tm} : \mathsf{Ty} \to \mathsf{Set} \\
&- \Rightarrow - : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} && \mathsf{lam} : (\mathsf{Tm}\,A \to \mathsf{Tm}\,B) \cong \mathsf{Tm}\,(A \Rightarrow B) : - \cdot -
\end{aligned}$$

An alternative popular description of simply typed lambda calculus is when we omit Ty and Tm, write a horizontal line or $\vdash$ for function space, give names to every input of a function (write $(a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,B$ instead of $\mathsf{Tm}\,A \to \mathsf{Tm}\,B$) and use named function application with square brackets (write $t[x \mapsto a]$ instead of $t\,a$, where $t : (x : A) \to B[x \mapsto a]$, where $B : (x : A) \to \mathsf{Set}$).

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

A first-order model of the simply typed lambda calculus contains a category with a terminal object (Con, Sub and the empty context $\diamond$), two sorts Ty and Tm which are both indexed by contexts, and there are context extension operations both for types and terms (we only list the relevant parts for reasons of space):

$$
\begin{aligned}
&\mathsf{Ty} && : \mathsf{Con} \to \mathsf{Set} \\
&-[-]_{\mathsf{Ty}} && : \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta \\
&- \triangleright_{\mathsf{Ty}} && : \mathsf{Con} \to \mathsf{Con} \\
&\mathsf{p}_{\mathsf{Ty}} \circ -, \mathsf{q}_{\mathsf{Ty}}[-] && : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright_{\mathsf{Ty}}) \cong \mathsf{Sub}\,\Delta\,\Gamma \times \mathsf{Ty}\,\Delta : - ,_{\mathsf{Ty}} - \\
&\mathsf{Tm} && : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
&-[-]_{\mathsf{Tm}} && : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]_{\mathsf{Ty}}) \\
&- \triangleright_{\mathsf{Tm}} - && : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con} \\
&\mathsf{p}_{\mathsf{Tm}} \circ -, \mathsf{q}_{\mathsf{Tm}}[-] && : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright_{\mathsf{Tm}} A) \cong (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \times \mathsf{Tm}\,\Delta\,(A[\gamma]_{\mathsf{Ty}}) : - ,_{\mathsf{Tm}} -
\end{aligned}
$$

The context extension operations take as arguments the index of the corresponding sort: Ty is not indexed, so $\triangleright_{\mathsf{Ty}}$ does not take any arguments, $\triangleright_{\mathsf{Tm}}$ takes a Ty argument. In simply typed lambda calculus, none of the operations (or sorts) use type variables, hence it is not necessary to include the operation $\triangleright_{\mathsf{Ty}}$ and the type variables q_{Ty}, $\mathsf{q}_{\mathsf{Ty}}[\mathsf{p}]$, $\mathsf{q}_{\mathsf{Ty}}[\mathsf{p}][\mathsf{p}]$, and so on. In the formal version of signatures (Definition 10), we will distinguish those sorts which have variables and those which do not, so this optimisation can be handled by our setup. The fact that all types are closed (don't depend on term variables, hence do not depend on the context at all) will not be handled by our translation, so the generated theory will include unnecessary dependencies, and a by hand optimisation step is needed to replace $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}$ by $\mathsf{Ty} : \mathsf{Set}$ and removing the $-[-]_{\mathsf{Ty}}$ operation. The operations in the notion of first-order model are the typed versions of the operations in Definition 4, for example $\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Tm}} A)\,B \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)$. This concludes the typed lambda calculus example.

The following definition of first-order logic has minimal amount of logical connectives, but illustrates the general idea. The proof theory that comes with it is natural deduction style, it can be also written following the above conventions using horizontal lines and $\vdash$.

▶ **Definition 6** (Minimal intuitionistic first-order logic).

$$
\begin{aligned}
&\mathsf{For} && : \mathsf{Set} \\
&\mathsf{Tm} && : \mathsf{Set} \\
&- \supset - && : \mathsf{For} \to \mathsf{For} \to \mathsf{For} \\
&\forall && : (\mathsf{Tm} \to \mathsf{For}) \to \mathsf{For} \\
&\mathsf{Eq} && : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{For}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Pf} && : \mathsf{For} \to \mathsf{Prop} \\
&\mathsf{intro}^{\supset} && : (\mathsf{Pf}\,A \to \mathsf{Pf}\,B) \leftrightarrow \mathsf{Pf}\,(A \supset B) \\
&\mathsf{intro}^{\forall} && : \big((t : \mathsf{Tm}) \to \mathsf{Pf}\,(A\,t)\big) \leftrightarrow \mathsf{Pf}\,(\forall A) \\
&\mathsf{intro}^{\mathsf{Eq}} && : \mathsf{Pf}\,(\mathsf{Eq}\,t\,t) \\
&\mathsf{elim}^{\mathsf{Eq}} && : (A : \mathsf{Tm} \to \mathsf{For}) \to \mathsf{Pf}\,(\mathsf{Eq}\,t\,t') \to \\
& && \quad \mathsf{Pf}\,(A\,t) \to \mathsf{Pf}\,(A\,t')
\end{aligned}
$$

A first-order model contains a category of contexts and substitutions equipped with three different kinds of context extension corresponding to three different kinds of variables. This means that there are three different 0 De Burijn indices ($\mathsf{q}_{\mathsf{For}}$, q_{Tm} q_{Pf}), nine different 1 De Bruijn indices ($\mathsf{q}_{\mathsf{For}}[\mathsf{p}_{\mathsf{For}}]_{\mathsf{For}}$, $\mathsf{q}_{\mathsf{For}}[\mathsf{p}_{\mathsf{Tm}}]_{\mathsf{For}}$, $\mathsf{q}_{\mathsf{For}}[\mathsf{p}_{\mathsf{Pf}}]_{\mathsf{For}}$, $\dots$, $\mathsf{q}_{\mathsf{Pf}}[\mathsf{p}_{\mathsf{Pf}}]_{\mathsf{Pf}}$). In general, De Bruijn index n has 3^{n+1} variants. We list the types of the binders:

$$
\begin{aligned}
&\forall && : \mathsf{For}\,(\Gamma \triangleright_{\mathsf{Tm}}) \to \mathsf{For}\,\Gamma \\
&\mathsf{intro}^{\supset} && : \mathsf{Pf}\,(\Gamma \triangleright_{\mathsf{Pf}} A)\,(B[\mathsf{p}_{\mathsf{Pf}}]_{\mathsf{For}}) \to \mathsf{Pf}\,\Gamma\,(A \supset B) \\
&\mathsf{intro}^{\forall} && : \mathsf{Pf}\,(\Gamma \triangleright_{\mathsf{Tm}})\,A \to \mathsf{Pf}\,\Gamma\,(\forall A) \\
&\mathsf{elim}^{\mathsf{Eq}} && : \big(A : \mathsf{For}\,(\Gamma \triangleright_{\mathsf{Tm}})\big) \to \mathsf{Pf}\,\Gamma\,(\mathsf{Eq}\,t\,t') \to \mathsf{Pf}\,\Gamma\,(A[\mathsf{id} ,_{\mathsf{Tm}} t]_{\mathsf{For}}) \to \mathsf{Pf}\,\Gamma\,(A[\mathsf{id} ,_{\mathsf{Tm}} t']_{\mathsf{For}})
\end{aligned}
$$

The GAT presentation of first-order logic can be simplified by removing For variables as no operations bind formulas. Another post-hoc simplification is separating the Tm-variable contexts and the Pf-variable contexts which depend on the former. After such a separation, it is possible to define [11] the syntax of first-order logic simply using inductive types and avoiding quotienting (with the exception of Pf where we use a full quotient which can be implemented by SProp of Agda or Coq [24]). One reason for being able to do this is that the above SOGAT does not have any equations, but this is not enough in general. For example, if we do not have quotients, it does not seem to be possible to define the syntax of a Martin-Löf type theory which does not have computation rules.

Our next example is a theory with dependent types featuring Π types, a Coquand-universe (which forces types to be indexed by levels) and a lifting operation. This is an open signature which means that it refers to some external types, in this case a natural number algebra (we can make it closed by adding $\mathbb{N}$ as a new sort and 0 and $1+-$ as new operations).

▶ **Definition 7** (Minimal Martin-Löf type theory).

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

The first-order translation of this theory results in a category with families (CwF [18]), more precisely, a category with $\mathbb{N}$-many families equipped with familywise Π-types, universes and a one-step upwards lifting between the families. The sorts are $\mathsf{Ty} : \mathsf{Con} \to \mathbb{N} \to \mathsf{Set}$ and $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Set}$, the i argument is implicit in the latter.

Instead of a Coquand-universe with c and El, we could have defined a Russell universe where we have a sort equality $\mathsf{Ty}\,i = \mathsf{Tm}\,(\mathsf{U}\,i)$, and we also have the option to do this for lifting and Π types. The first-order semantics of such a theory has the following equalities: $\mathsf{Ty}\,\Gamma\,i = \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i)$, and having strict Π types means $\mathsf{Tm}\,(\Gamma \triangleright A)\,B = \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$.

3 Theories of signatures as SOGATs

In this section we define three languages which describe signatures for ATs, GATs and SOGATs, respectively. All three languages are given as SOGATs.

The theory of signatures for ATs is a dependent type theory without a universe, it has one base type Srt for the (single) sort, Σ types, a Π type with fixed Srt domain, and an equality type. Π types are equipped with application, but the Σ and Eq types don't have constructors or destructors, because those are not needed when defining signatures.

▶ **Definition 8** (Signatures for single-sorted algebraic theories).

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

A first-order model of this theory is a CwF with type formers Σ, Srt, $\Pi\mathsf{Srt}$, Eq and a term former $-\cdot- : \mathsf{Tm}\,\Gamma\,(\Pi\mathsf{Srt}\,B) \to (x : \mathsf{Tm}\,\Gamma\,\mathsf{Srt}) \to \mathsf{Tm}\,\Gamma\,(B[\mathsf{id}, x])$. An element of Ty in the syntax of this language is an AT signature. We introduce abbreviations $\mathsf{Srt}\Rightarrow A := \Pi\mathsf{Srt}\,\lambda\_.A$ and $A \times B := \Sigma\,A\,\lambda\_.B$. The signature for combinator calculus is the following Ty:

$$\Sigma\,(\mathsf{Srt}\Rightarrow\mathsf{Srt}\Rightarrow\mathsf{Srt})\,\lambda ap.\Sigma\,\mathsf{Srt}\,\lambda K.\Sigma\,\mathsf{Srt}\,\lambda S.\Big(\Pi\mathsf{Srt}\,\lambda u.\Pi\mathsf{Srt}\,\lambda f.\mathsf{Eq}\,(ap\cdot(ap\cdot K\cdot u)\cdot f)\,u\Big)$$
$$\times\Big(\Pi\mathsf{Srt}\,\lambda f.\Pi\mathsf{Srt}\,\lambda g.\Pi\mathsf{Srt}\,\lambda u.\mathsf{Eq}\,\Big(ap\cdot(ap\cdot(ap\cdot S\cdot f)\cdot g)\cdot u\Big)\,\Big(ap\cdot(ap\cdot f\cdot u)\cdot(ap\cdot g\cdot u)\Big)\Big)$$

This can be seen as a more explicit version of Definition 1: we use Σ types instead of a newline-separated list, we use the metatheoretic λ binder to give names to operations, we use an explicit $\cdot$ operation for application and write Eq instead of $=$. Moreover, we don't have infix operators or implicit arguments, the three arguments of equation $\mathsf{K}\beta$ and the four arguments of equation $\mathsf{S}\beta$ have to be introduced using $\Pi\mathsf{Srt}$ explicitly. Being more explicit is needed to make sure that we describe an algebraic theory: for example, the fact that the domain of Π is fixed ensures strict positivity.

The theory of GAT signatures (ToS) is a type theory with an empty universe (a type and a family over it), $\top$ and Σ types, equality with reflection, and a Π type with U-domain.

▶ **Definition 9** (ToS: the theory of GAT signatures)**.**

$$\begin{array}{llll}
\mathsf{Ty} & : \mathsf{Set} & \Sigma & : (A:\mathsf{Ty})\to(\mathsf{Tm}\,A\to\mathsf{Ty})\to\mathsf{Ty}\\
\mathsf{Tm} & : \mathsf{Ty}\to\mathsf{Set} & (-,-) & : (a:\mathsf{Tm}\,A)\times\mathsf{Tm}\,(B\,a)\cong\mathsf{Tm}\,(\Sigma\,A\,B) : \mathsf{fst},\mathsf{snd}\\
\mathsf{U} & : \mathsf{Ty} & \Pi & : (a:\mathsf{Tm}\,\mathsf{U})\to(\mathsf{Tm}\,(\mathsf{El}\,a)\to\mathsf{Ty})\to\mathsf{Ty}\\
\mathsf{El} & : \mathsf{Tm}\,\mathsf{U}\to\mathsf{Ty} & \mathsf{lam} & : \big((x:\mathsf{Tm}\,(\mathsf{El}\,a))\to\mathsf{Tm}\,(B\,x)\big)\cong\mathsf{Tm}\,(\Pi\,a\,B) : -\cdot-\\
\top & : \mathsf{Ty} & \mathsf{Eq} & : (A:\mathsf{Ty})\to\mathsf{Tm}\,A\to\mathsf{Tm}\,A\to\mathsf{Ty}\\
\mathsf{tt} & : \mathbb{1}\cong\mathsf{Tm}\,\Gamma\,\top & \mathsf{refl} & : (u=v)\cong\mathsf{Tm}\,(\mathsf{Eq}\,A\,u\,v) : \mathsf{reflect}
\end{array}$$

The first-order version is Definition 11. A (presentation of a) GAT is defined as a closed type in the syntax of ToS. The only base type U is for declaring sorts, so a signature has to start with a sort, and then we can declare elements of the sort using El or functions where the input is a sort. For example, part of typed combinator calculus (Definition 2) is given by the following signature. We use the abbreviations $a\Rightarrow B:=\Pi\,a\,\lambda\_.b$ and $A\times B=\Sigma\,A\,\lambda\_.B$. We left out the S combinator and its β rule for reasons of space.

$$\Sigma\,\mathsf{U}\,\lambda Ty.\Sigma\,(Ty\Rightarrow\mathsf{U})\,\lambda Tm.\mathsf{El}\,Ty\times\Sigma\,(Ty\Rightarrow Ty\Rightarrow\mathsf{El}\,Ty)\,\lambda arr.\Sigma$$
$$(\Pi\,Ty\,\lambda A.\Pi\,Ty\,\lambda B.Tm\cdot(arr\cdot A\cdot B)\Rightarrow Tm\cdot A\Rightarrow\mathsf{El}\,(Tm\cdot B))\,\lambda app.\Sigma$$
$$\Big(\Pi\,Ty\,\lambda A.\Pi\,Ty\,\lambda B.\mathsf{El}\,\Big(Tm\cdot(arr\cdot A\cdot(arr\cdot B\cdot A))\Big)\Big)\,\lambda K.\Sigma$$
$$\Big(\Pi\,Ty\,\lambda A.\Pi\,Ty\,\lambda B.\Pi\,(Tm\cdot A)\,\lambda u.\Pi\,(Tm\cdot B)\,\lambda f.\mathsf{Eq}\,(\mathsf{El}\,(Tm\cdot A))$$
$$(app\cdot(arr\cdot B\cdot A)\cdot B\cdot(app\cdot(arr\cdot A\cdot(arr\cdot B\cdot A))\cdot A\cdot K\cdot u)\cdot f)\,u\Big)\times\dots$$

This type is a very explicit version of Definition 2: we use Σ, explicit application $\cdot$, no infix operators, no implicit arguments, and explicit El turning terms in U into types. We expect that an elaboration algorithm can turn Definition 2 into such an explicit version.

For the theory of SOGAT signatures (ToS$^+$), we add a new universe U^+ of sorts for which variables are allowed: with the help of these we can write second order functions. U^+ is a subuniverse of U (witnessed by el^+) and has a Π type with U^+-domain and U-codomain.

▶ **Definition 10** (ToS$^+$: the theory of SOGAT signatures)**.** *We extend ToS with the following.*

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

The first-order version is Definition 12. A (presentation of a) GAT is defined as a closed type in the syntax of ToS^+. The signature for lambda calculus (Definition 3) is the following element of Ty.

$$\Sigma\,\mathsf{U}^+\,\lambda Tm.\Sigma\left((Tm \Rightarrow^+ \mathsf{el}^+\,Tm) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,Tm)\right)\lambda lam.\Sigma\left(\mathsf{el}^+\,Tm \Rightarrow \mathsf{el}^+\,Tm \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,Tm)\right)\lambda app.$$
$$\Pi\,(Tm \Rightarrow^+ \mathsf{el}^+\,Tm)\,\lambda t.\Pi\,(\mathsf{el}^+\,Tm)\,\lambda u.\mathsf{Eq}\left(\mathsf{El}\,(\mathsf{el}^+\,Tm)\right)\left(app\cdot(lam\cdot t)\cdot u\right)(t\cdot^+ u)$$

We have one sort Tm for which variables are allowed, application app uses ordinary function space $\Rightarrow$ where Tm has to be lifted by el^+ from U^+ to U. Lambda lam is defined as a second-order function where $\Rightarrow^+$ can appear on the left hand side of an $\Rightarrow$. When stating the β equation, note the two different application operators ($\cdot$ vs. $\cdot^+$): $\cdot^+$ is used when giving value to a variable. This becomes clear if we look at the first-order presentation of the β law (last line in Definition 4, we write app instead of $\cdot$ to avoid confusion): $app\,(\mathsf{lam}\,t)\,u = t[\mathsf{id},u]$. So the semantics of $\cdot$ should be simply function application, while the semantics of $\cdot^+$ is instantiation with a substitution. We give another illustration of this difference: in the above signature, the type of app is $\mathsf{el}^+\,Tm \Rightarrow \mathsf{el}^+\,Tm \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,Tm)$, and this is translated to $Tm\,\Gamma \to Tm\,\Gamma \to Tm\,\Gamma$ in the GAT version (see Definition 4). But we could have defined app as having type $\mathsf{El}\,(Tm \Rightarrow^+ Tm \Rightarrow^+ Tm)$. In this case the GAT version of app would be in $Tm\,(\Gamma \triangleright \triangleright)$. Both variants are meaningful, and ToS^+ allows the user to make a choice if she wants an operation with arguments, or an operation returning in an extended context. Note that both function spaces in the type of lam are forced to be $\Rightarrow^+$ and $\Rightarrow$, respectively.

Analogously, all SOGATs in Sections 2, 3 and Appendix A can be reified into SOGAT signatures (with the exception of Martin-Löf type theory which is an open signature, but we will rectify this in Section 7). This includes ToS^+ itself.

4 Naive semantics of SOGAT signatures

In this section, for any SOGAT signature, we define a notion of first-order model. The idea is that a model is a category together with the presheaf interpretation of the signature over that category: the category of presheaves supports a universe, Π types, and so on, so we directly use these when interpreting the type formers of ToS^+. We assume basic working knowledge of categories with families (CwFs [18]).

▶ **Definition 11** (First-order model of ToS). *A first-order model of ToS is a CwF (sorts are denoted* Con, Sub, Ty, Tm, *context extension is* $-\triangleright-$ *with* $\mathsf{p}\circ-,\mathsf{q}[-] : \mathsf{Sub}\,\Delta\,(\Gamma\triangleright A) \cong (\gamma : \mathsf{Sub}\,\Delta\,\Gamma)\times\mathsf{Tm}\,\Delta\,(A[\gamma]) : -,-$*) equipped with:*

- $\top$ *and* Σ *types given by isomorphisms* $\mathsf{tt} : \mathbb{1} \cong \mathsf{Tm}\,\Gamma\,\top,\qquad (-,-) : (a : \mathsf{Tm}\,\Gamma\,A)\times\mathsf{Tm}\,\Gamma\,(B[\mathsf{id},a]) \cong \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B) : \mathsf{fst},\mathsf{snd}.$
- *A universe given by* $\mathsf{U} : \mathsf{Ty}\,\Gamma$ *and* $\mathsf{El} : \mathsf{Tm}\,\Gamma\,\mathsf{U}$.
- *A function space with domain in* U, *that is* $\Pi : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma\triangleright\mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma$, *with an isomorphism* $\mathsf{lam} : \mathsf{Tm}\,(\Gamma\triangleright\mathsf{El}\,a)\,B \cong \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) : \mathsf{app}$.
- *A strict equality type* Eq *with reflection and uniqueness of identity proofs.*
- *All the operations listed above are natural in* Γ.

▶ **Definition 12** (First-order model of ToS^+). *A first-order model of* ToS^+ *is a first-order model of ToS equipped with:*

- *Another universe* $\mathsf{U}^+ : \mathsf{Ty}\,\Gamma$ *that is a subuniverse of* U *i.e.* $\mathsf{el}^+ : \mathsf{Tm}\,\Gamma\,\mathsf{U}^+ \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$.
- U *is closed under functions with* U^+*-domain, i.e.* $\pi^+ : (a^+ : \mathsf{Tm}\,\Gamma\,\mathsf{U}^+) \to \mathsf{Tm}\,\left(\Gamma\triangleright\mathsf{El}\,(\mathsf{el}^+\,a^+)\right)\mathsf{U} \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$ *with* $\mathsf{lam}^+ : \mathsf{Tm}\,\left(\Gamma\triangleright\mathsf{El}\,(\mathsf{el}^+\,a)\right)(\mathsf{El}\,b) \cong \mathsf{Tm}\,\Gamma\,\left(\mathsf{El}\,(\pi^+\,a^+\,b)\right) : \mathsf{app}^+$.
- *All the operations listed above are natural in* Γ.

▶ **Problem 13** (PSh($\mathcal{C}$)). *Presheaves over $\mathcal{C}$ form a CwF equipped with $\top$, Σ types, an equality type with reflection, Π types and a Coquand-universe* U *with* $\mathsf{c} : \mathsf{Ty}\,\Gamma \cong \mathsf{Tm}\,\Gamma\,\mathsf{U} : \mathsf{El}$. *Unlike in Definition 7, we omit writing universe indices for readibility.*

Construction. We recall the main parts of the construction [30] for fixing notations. $\Gamma : \mathsf{Con}$ is a presheaf, that is a family of sets $\Gamma : \mathcal{C} \to \mathsf{Set}$ with reindexing $\gamma_I[f]_\Gamma : \Gamma\,J$ for $\gamma_I : \Gamma\,I$ and $f : \mathcal{C}(J,I)$ such that $\gamma_I[f \circ g]_\Gamma = \gamma_I[f]_\Gamma[g]_\Gamma$ and $\gamma_I[\mathsf{id}]_\Gamma = \gamma_I$. A $\sigma : \mathsf{Sub}\,\Delta\,\Gamma$ is a function $\sigma : \Delta\,I \to \Gamma\,I$ such that $(\sigma\,\delta_I)[f]_\Gamma = \sigma\,(\delta_I[f]_\Delta)$. A type $A : \mathsf{Ty}\,\Gamma$ is a dependent presheaf containing a family $A : (I : \mathcal{C}) \to \Gamma\,I \to \mathsf{Set}$ with reindexing $a_I[f]_A : A\,J\,(\gamma_I[f]_\Gamma)$ for $a_I : A\,I\,\gamma_I$ and $f : \mathcal{C}(J,I)$ satisfying functoriality. Type substitution is $A[\gamma]\,I\,\delta_I := A\,I\,(\gamma\,\delta_I)$. A term $a : \mathsf{Tm}\,\Gamma\,A$ is a function $a : (\gamma_I : \Gamma\,I) \to A\,I\,\gamma_I$ such that $(a\,\gamma_I)[f]_A = a\,(\gamma_I[f]_\Gamma)$. Term substitution is $a[\gamma]\,\delta_I := a\,(\gamma\,\delta_I)$. The empty context is constant unit: $\diamond\,I := \mathbb{1}$. Context extension is pointwise: $(\Gamma \triangleright A)\,I := (\gamma_I : \Gamma\,I) \times A\,I\,\gamma_I$, its universal property is given by projections and pairing for metatheoretic Σ types. $\top$, Σ and Eq are pointwise. We have the functor Yoneda $\mathsf{y} : \mathcal{C} \to \mathsf{PSh}(\mathcal{C})$ defined by $\mathsf{y}\,I\,J := \mathcal{C}(J,I)$, and we use this to define the universe by $\mathsf{U}\,I\,\gamma_I := \mathsf{Ty}\,(\mathsf{y}\,I)$. We observe that $\gamma_I[-]_\Gamma : \mathsf{Sub}\,(\mathsf{y}\,I)\,\Gamma$ (forward part of Yoneda lemma), and define $\Pi\,A\,B\,I\,\gamma_I := \mathsf{Tm}\,(\mathsf{y}\,I \triangleright A[\gamma_I[-]_\Gamma])\,(B[\gamma_I[-]_\Gamma \circ \mathsf{p}, \mathsf{q}])$. ◀

▶ **Problem 14** (Locally representable types). *The CwF of presheaves can be extended to a CwF*$^+$*, which means a CwF with a subsort of* Ty *called* Ty^+ *and a* Π^+ *type with domain in* Ty^+*, i.e.* $\Pi^+ : (A : \mathsf{Ty}^+\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$ *with* $\mathsf{lam}^+ : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \cong \mathsf{Tm}\,\Gamma\,(\Pi^+\,A\,B) : \mathsf{app}^+$*, natural in* Γ*.* Ty^+ *is classified by the Coquand universe* U^+*.*

Construction. An element $A : \mathsf{Ty}^+\,\Gamma$ is an $A : \mathsf{Ty}\,\Gamma$ together with $- \triangleright_A - : (I : \mathcal{C}) \to \Gamma\,I \to \mathcal{C}$ and an isomorphism $\mathsf{p}_A \circ -, \mathsf{q}_A[-]_A : \mathcal{C}(J, I \triangleright_A \gamma_I) \cong (f : \mathcal{C}(J,I)) \times A\,J\,(\gamma_I[f]_\Gamma) : -,_A-$ natural in J. So $\mathsf{p}_A : \mathcal{C}(I \triangleright_A \gamma_I, I)$ and $\mathsf{q}_A : A\,(I \triangleright_A \gamma_I)\,(\gamma_I[\mathsf{p}_A]_\Gamma)$. Substitution is given by $I \triangleright_{A[\gamma]} \delta_I := I \triangleright_A \gamma\,\delta_I$ and we have $\mathcal{C}(J, I \triangleright_{A[\gamma]} \delta_I) = \mathcal{C}(J, I \triangleright_A \gamma\,\delta_I) \cong (f : \mathcal{C}(J,I)) \times A\,J\,(\gamma\,\delta_I[f]_\Gamma) = (f : \mathcal{C}(J,I)) \times A[\gamma]\,J\,(\delta_I[f]_\Delta)$. We define Π^+ using the $\triangleright_A$ operator which comes with A, i.e. $\Pi^+\,A\,B\,I\,\gamma_I := B\,(I \triangleright_A \gamma_I)\,(\gamma_I[\mathsf{p}_A]_\Gamma, \mathsf{q}_A)$, $b_{I'}[f]_{\Pi^+ A\,B} := b_{I'}[f \circ \mathsf{p}_A \,,_A\, \mathsf{q}_A]$, $\mathsf{lam}^+\,b\,\gamma_I := b\,(\gamma_I[\mathsf{p}_A]_\Gamma, \mathsf{q}_A)$ and $\mathsf{app}^+\,t\,(\gamma_I, a_I) := (t\,\gamma_I)[\mathsf{id}_I \,,_A\, a_I]_B$. Like U, $\mathsf{U}^+\,I\,\gamma_I := \mathsf{Ty}^+\,(\mathsf{y}\,I)$. ◀

▶ **Definition 15** (Naive semantics). *Given a category* $\mathcal{C}$*,* $\mathsf{PSh}(\mathcal{C})$ *is a model of ToS*$^+$ *choosing* $\mathsf{U} := \mathsf{U}$*,* $\mathsf{El}\,a := \mathsf{El}\,a$*,* $\Pi\,a\,B := \Pi\,(\mathsf{El}\,a)\,B$*,* $\mathsf{U}^+ := \mathsf{U}^+$*,* $\mathsf{el}^+\,a^+ := \mathsf{c}\,(\mathsf{El}^+\,a^+)$*,* $\pi^+\,a^+\,b := \mathsf{c}\,\big(\Pi^+\,(\mathsf{El}^+\,a^+)\,(\mathsf{El}\,b)\big)$*. Recall that a SOGAT signature* Ω *is an element of* $\mathsf{Ty}\,\diamond$ *in the syntax of ToS*$^+$*. A naive model of* Ω *is a category with a terminal object together with the interpetation of* Ω *in presheaves over this category, i.e.* $(\mathcal{C} : \mathsf{Cat}^\diamond) \times \mathsf{Tm}_{\mathsf{PSh}(\mathcal{C})}\,\diamond\,[\![\Omega]\!]_{\mathsf{PSh}(\mathcal{C})}$*.*

This definition immediately implies that internally to presheaves over a naive first-order model, we have a second order model.

For illustration, we compute the naive semantics for the signature of untyped lambda calculus without the equations. The informal signature is $\mathsf{Tm} : \mathsf{U}, \mathsf{lam} : (\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm}, -\cdot- : \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm}$, the second-order formal version is $\Sigma\,\mathsf{U}^+\,\lambda Tm.\big((Tm \Rightarrow^+ \mathsf{el}^+\,Tm) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,Tm)\big) \times (\mathsf{el}^+\,Tm \Rightarrow \mathsf{el}^+\,Tm \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,Tm))$, and we interpret the first-order version of this. We assume a $\mathcal{C} : \mathsf{Cat}^\diamond$, write $\mathcal{D} := \mathsf{PSh}(\mathcal{C})$, and use $\mathsf{Tm}_\mathcal{D}\,\diamond\,[\![\Omega]\!]_\mathcal{D} \cong [\![\Omega]\!]_\mathcal{D}\,\diamond_\mathcal{C}\,*$.

$$
\begin{aligned}
&\Big[\!\!\Big[\Sigma\,\mathsf{U}^+\,\Big(\big((\mathsf{q} \Rightarrow^+ \mathsf{el}^+\,\mathsf{q}) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})\big) \times \big(\mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})\big)\Big)\Big]\!\!\Big]_\mathcal{D}\,\diamond_\mathcal{C}\,* = \\
&(Tm : \mathsf{Ty}^+_\mathcal{D}\,(\mathsf{y}\,\diamond)) \times \mathsf{Tm}_\mathcal{D}\,\big(\mathsf{y}\,\diamond \triangleright (Tm \Rightarrow^+_\mathcal{D} Tm)\big)\,(Tm[\mathsf{p}]) \times \mathsf{Tm}_\mathcal{D}\,(\mathsf{y}\,\diamond \triangleright Tm)\,(Tm \Rightarrow Tm[\mathsf{p}]) = \\
&(Tm : (I : \mathcal{C}) \to \mathcal{C}(I, \diamond) \to \mathsf{Set}) \times (-[-]_{Tm} : Tm\,I\,\epsilon \to \mathcal{C}(J,I) \to Tm\,J\,\epsilon) \times \ldots \times \\
&\quad (- \triangleright_{Tm} - : (I : \mathcal{C}) \to \mathcal{C}(I,\diamond) \to \mathcal{C}) \times \cdots \times \big(lam : \mathcal{C}(I,\diamond) \times Tm\,(I \triangleright_{Tm} \epsilon) \to Tm\,I\,\epsilon\big) \times \cdots \times \\
&\quad (app : \mathcal{C}(I,\diamond) \times Tm\,I\,\epsilon \to (\{J : \mathcal{C}\} \to \mathcal{C}(J,I) \times Tm\,J\,\epsilon \to Tm\,J\,\epsilon) \times \ldots) \times \ldots
\end{aligned}
$$

As we can see, the naive semantics produces some encoding overhead: the above definition differs from Definition 4 in the following ways: the operations are uncurried, have several extra $\mathcal{C}(I, \diamond)$ arguments (which can be all filled by ϵ), and the type of app quantifies over another object of $\mathcal{C}$ for each argument. This is the result of using the usual presheaf universe and function space for interpreting U and Π. We will rectify this in the next section.

5 Direct semantics of SOGAT signatures

In this section, we define first-order models of SOGATs using a more careful version of the presheaf model. We make sure that no Yoneda-encodings are present in the semantics using the idea of two-level type theory [4, 10] where preheaves over a CwF include a universe of "inner types" coming from the CwF. We extend two-level type theory with a separate function space where the domain is an inner type. This function space is isomorphic to the usual presheaf function space, but has a simpler semantics.

▶ **Problem 16** (Presheaves over a CwF). *If $\mathcal{C}$ is a CwF, then $\mathsf{PSh}(\mathcal{C})$ models ToS without using the usual presheaf U and Π.*

Construction. We interpret $\top$, Σ, Eq as in Problem 13, but define U, El and Π by $\mathsf{Ty}_{\mathcal{C}}$, $\mathsf{Tm}_{\mathcal{C}}$ and $\triangleright_{\mathcal{C}}$, respectively: $\mathsf{U}\, I\, \gamma_I := \mathsf{Ty}_{\mathcal{C}}\, I$, $\mathsf{El}\, a\, I\, \gamma_I := \mathsf{Tm}_{\mathcal{C}}\, I\, (a\, \gamma_I)$, $\Pi\, a\, B\, I\, \gamma_I := B\, (I \triangleright_{\mathcal{C}} a\, \gamma_I)\, (\gamma_I[\mathsf{p}_{\mathcal{C}}]_\Gamma, \mathsf{q}_{\mathcal{C}})$ with $\mathsf{lam}\, b\, \gamma_I := b\, (\gamma_I[\mathsf{p}_{\mathcal{C}}]_\Gamma, \mathsf{q}_{\mathcal{C}})$ and $\mathsf{app}\, t\, (\gamma_I, a_I) := t\, \gamma_I[\mathsf{id}_I\, ,_{\mathcal{C}}\, a_I]_B$. ◀

▶ **Problem 17** (Presheaves over a CwF⁺). *If the category $\mathcal{C}$ is a CwF⁺, then the previous model extends to a model of ToS⁺ (Definition 12).*

Construction. We interpret U^+, el^+ and π^+ by $\mathsf{Ty}^+_{\mathcal{C}}$, identity and $\Pi^+_{\mathcal{C}}$, respectively: $\mathsf{U}^+\, I\, \gamma_I := \mathsf{Ty}^+_{\mathcal{C}}\, I$, $\mathsf{el}^+\, a\, \gamma_I := a\, \gamma_I$, $\pi^+\, a\, b\, \gamma_I := \Pi^+\, (a\, \gamma_I)\, \big(b\, (\gamma_I[\mathsf{p}_{\mathcal{C}}]_\Gamma, \mathsf{q}_{\mathcal{C}})\big)$, $\mathsf{lam}^+\, t\, \gamma_I := \mathsf{lam}^+_{\mathcal{C}}\, \big(t\, (\gamma_I[\mathsf{p}_{\mathcal{C}}]_\Gamma, \mathsf{q}_{\mathcal{C}})\big)$, $\mathsf{app}^+\, t\, (\gamma_I, a_I) := \mathsf{app}^+_{\mathcal{C}}\, (t\, \gamma_I)[\mathsf{id}_I\, ,_{\mathcal{C}}\, a_I]_{\mathsf{Tm}_{\mathcal{C}}}$. ◀

▶ **Definition 18** (Direct semantics). *A direct model of a SOGAT signature Ω is a category with a terminal object $\mathcal{C}$ together with the interpretation of Ω in presheaves over presheaves over $\mathcal{C}$, evaluated at the terminal presheaf: $(\mathcal{C} : \mathsf{Cat}^\diamond) \times [\![\Omega]\!]_{\mathsf{PSh}(\mathsf{PSh}(\mathcal{C}))}\, \diamond_{\mathsf{PSh}(\mathcal{C})}\, *$. Note that this makes sense because $\mathsf{PSh}(\mathcal{C}) : \mathsf{CwF}^+$, hence $\mathsf{PSh}\big(\mathsf{PSh}(\mathcal{C})\big)$ is a model of ToS⁺.*

We revisit the example from the end of the previous section. We again assume a $\mathcal{C} : \mathsf{Cat}^\diamond$ and write $\mathcal{D} := \mathsf{PSh}(\mathcal{C})$ and $\mathcal{E} := \mathsf{PSh}(\mathcal{D})$.

$$
\begin{aligned}
&\Big[\!\!\Big[\Sigma\, \mathsf{U}^+\, \Big(\big((\mathsf{q} \Rightarrow^+ \mathsf{el}^+\, \mathsf{q}) \Rightarrow \mathsf{El}\, (\mathsf{el}^+\, \mathsf{q})\big) \times \big(\mathsf{el}^+\, \mathsf{q} \Rightarrow \mathsf{el}^+\, \mathsf{q} \Rightarrow \mathsf{El}\, (\mathsf{el}^+\, \mathsf{q})\big)\Big)\Big]\!\!\Big]_{\mathcal{E}}\, \diamond_{\mathcal{D}}\, * = \\
&(Tm : \mathsf{Ty}^+_{\mathcal{D}}\, \diamond_{\mathcal{D}}) \times \mathsf{Tm}_{\mathcal{D}}\, \big(\diamond \triangleright (Tm \Rightarrow^+_{\mathcal{D}} Tm)\big)\, (Tm[\mathsf{p}]) \times \mathsf{Tm}_{\mathcal{D}}\, (\diamond \triangleright Tm \triangleright Tm[\mathsf{p}])\, (Tm[\mathsf{p}][\mathsf{p}]) = \\
&(Tm : \mathcal{C} \to \mathbb{1} \to \mathsf{Set}) \times (-[-]_{Tm} : Tm\, I\, * \to \mathcal{C}(J, I) \to Tm\, J\, *) \times \ldots \times \\
&\quad (- \triangleright_{Tm} - : \mathcal{C} \to \mathbb{1} \to \mathcal{C}) \times \cdots \times \big(lam : \mathbb{1} \times Tm\, (I \triangleright_{Tm} *) \to Tm\, I\, *\big) \times \cdots \times \\
&\quad (app : \mathbb{1} \times Tm\, I\, * \times Tm\, I\, * \to Tm\, I\, *) \times \ldots
\end{aligned}
$$

This translation is closer to computing Definition 4 from Definition 3: the only remaining noise is that the types of Tm, lam and app include extra $\mathbb{1}$ components and app is uncurried. In the next section, we will remove the extra $\mathbb{1}$s and make the type of application curried.

▶ **Theorem 19.** *For any signature, the naive and direct semantics result in isomorphic notions of models.*

Proof. We fix a $\mathcal{C} : \mathsf{Cat}^{\diamond}$, and denote $\mathcal{D} := \mathsf{PSh}(\mathcal{C})$ and $\mathcal{E} := \mathsf{PSh}(\mathcal{D})$. $\mathcal{D}$ is a model of ToS⁺ via Definition 15 and $\mathcal{E}$ is a model via Definition 18, and Yoneda navigates between them (it is not only a functor, but a CwF pseudomorphism [36]). By induction on the syntax of ToS⁺, we define α for contexts, substitutions, types and terms: $\alpha_\Gamma : \mathsf{Sub}_{\mathcal{E}}\,[\![\Gamma]\!]_{\mathcal{E}}\,(\mathsf{y}\,[\![\Gamma]\!]_{\mathcal{D}})$, $\alpha_\gamma : \alpha_\Gamma \circ [\![\gamma]\!]_{\mathcal{E}} = \mathsf{y}\,[\![\gamma]\!]_{\mathcal{D}} \circ \alpha_\Delta$, $\alpha_A : [\![A]\!]_{\mathcal{E}} \cong \mathsf{y}\,[\![A]\!]_{\mathcal{D}}[\alpha_\Gamma]$, $\alpha_a : \alpha_A[\mathsf{id}, [\![a]\!]_{\mathcal{E}}] = \mathsf{y}\,[\![a]\!]_{\mathcal{D}}[\alpha_\Gamma]$. For a signature $\Omega : \mathsf{Ty}\,\diamond$, we thus obtain $[\![\Omega]\!]_{\mathcal{E}} \diamond_{\mathcal{D}} * \cong \mathsf{y}\,[\![\Omega]\!]_{\mathcal{D}}[\alpha_\Gamma] \diamond_{\mathcal{D}} * = \mathsf{Tm}_{\mathcal{D}} \diamond_{\mathcal{D}} [\![\Omega]\!]_{\mathcal{D}}$. ◀

6 GAT signature semantics of SOGAT signatures

In this section we translate SOGAT signatures into GAT signatures. The idea is the same as in the previous two sections: the GAT signature will start with a category with terminal object and then contain the presheaf interpretation of the SOGAT signature over that category. However now the presheaf model is not expressed in the metatheory, but internally to the theory of GAT signatures. This is challenging because this language is quite limited: there are no higher-order functions, no real universe, and so on.

In this section we work internally to presheaves over the syntax of ToS. Another way to say this is that we work in two-level type theory where the inner model is the syntax of ToS. Hence, we have the components $\mathsf{Ty} : \mathsf{Set}$, $\mathsf{Tm} : \mathsf{Ty} \to \mathsf{Set}$, …, $\mathsf{refl} : (u = v) \cong \mathsf{Tm}\,(\mathsf{Eq}\,A\,u\,v) : \mathsf{reflect}$ of Definition 9 available (these are the inner types and type formers). We will build a first-order model of ToS⁺, and the final result of the translation will be an element of Ty.

▶ **Construction 20** (Curreid Π)**.** *By induction-recursion, we define the Σ-closure of* U*.*

$$
\begin{aligned}
&\mathsf{U}^* : \mathsf{Set} && \mathsf{El}^* : \mathsf{U}^* \to \mathsf{Ty}\\
&\top^* : \mathsf{U}^* && \mathsf{El}^*\,\top^* := \top\\
&\Sigma^* : (as : \mathsf{U}^*) \to \big(\mathsf{Tm}\,(\mathsf{El}^*\,as) \to \mathsf{Tm}\,\mathsf{U}\big) \to \mathsf{U}^* && \mathsf{El}^*\,(\Sigma^*\,as\,b) := \Sigma\,(\mathsf{El}^*\,as)\,\lambda x.\mathsf{El}\,(b\,x)
\end{aligned}
$$

By induction on U^**, we define the curried function space with* U^* *domain. We call it* $\Pi^* : (as : \mathsf{U}^*) \to (\mathsf{Tm}\,(\mathsf{El}^*\,as) \to \mathsf{Ty}) \to \mathsf{Ty}$ *and is defined by* $\Pi^*\,\top^*\,B := B\,\mathsf{tt}$ *and* $\Pi^*(\Sigma^*\,as\,c)\,B := \Pi^*\,as\,\big(\lambda xs.\Pi\,(c\,xs)\,\lambda y.B\,(xs,y)\big)$*. It comes with* lam^**,* $\cdot^*$*, and* β*,* η *laws all defined by induction on* U^* *resulting in* $\mathsf{lam}^* : \big((xs : \mathsf{Tm}\,(\mathsf{El}^*\,as)) \to \mathsf{Tm}\,(B\,xs)\big) \cong \mathsf{Tm}\,(\Pi^*\,as\,B) : - \cdot^* -$*.*

We define the signature for category with a terminal object by $\mathsf{Cat}^{\diamond} : \mathsf{Ty} := \Sigma\,\mathsf{U}\,\lambda Ob.\Sigma\,(Ob \Rightarrow Ob \Rightarrow \mathsf{U})\,\lambda Hom\ldots$ We assume a $\mathcal{C} : \mathsf{Tm}\,\mathsf{Cat}^{\diamond}$, we refer to its components by Ob, Hom, …

▶ **Problem 21** (A CwF⁺ $\mathcal{D}$ of presheaves over $\mathcal{C}$)**.** *There is a notion of CwF⁺ where the sorts of types and terms are* Ty*-valued. We construct such a CwF⁺ $\mathcal{D}$ of presheaves over $\mathcal{C}$.*

Construction. The category part is given by U^*-valued presheaves and natural transformations where $\mathsf{Con}_{\mathcal{D}} := \big(\Gamma : \mathsf{Tm}\,(\mathsf{El}\,Ob) \to \mathsf{U}^*\big) \times \Big(-[-]_\Gamma : \mathsf{Tm}\,\big(\mathsf{El}^*\,(\Gamma\,I)\big) \to \mathsf{Tm}\,\big(\mathsf{El}\,(Hom \cdot J \cdot I)\big) \to \mathsf{Tm}\,\big(\mathsf{El}^*\,(\Gamma\,J)\big)\Big) \times$(functoriality) and $\mathsf{Sub}_{\mathcal{D}}\,\Delta\,\Gamma := \Big(\gamma : \mathsf{Tm}\,\big(\mathsf{El}^*\,(\Delta\,I)\big) \to \mathsf{Tm}\,\big(\mathsf{El}^*\,(\Gamma\,I)\big)\Big) \times$ (naturality). We make sure that Ty, Tm have enough structure to define U-valued presheaves. For example, we define $\mathsf{Ty}_{\mathcal{D}} : \mathsf{Con}_{\mathcal{D}} \to \mathsf{Ty}$ by

$$
\begin{aligned}
&\mathsf{Ty}_{\mathcal{D}}\,\Gamma := \Sigma\,(\Pi\,Ob\,\lambda I.\Gamma\,I \Rightarrow^* \mathsf{U})\,\lambda A.\Sigma\\
&\Big(\Pi\,Ob\,\lambda I.\Pi^*\,(\Gamma\,I)\,\lambda\gamma_I.A \cdot I \cdot^* \gamma_I \Rightarrow \Pi\,Ob\,\lambda J.\Pi\,(Hom \cdot J \cdot I)\,\lambda f.\mathsf{El}\,\big(A \cdot J \cdot^* (\gamma_I[f]_\Gamma)\big)\Big)\ldots
\end{aligned}
$$

$\mathsf{Tm}_{\mathcal{D}}$ is defined using Π^*-functions, context extension $\triangleright_{\mathcal{D}}$ is Σ^*, $\mathsf{Ty}^+_{\mathcal{D}}$ is the same as $\mathsf{Ty}_{\mathcal{D}}$ extended with an $\triangleright_A$ operator in $\Pi\,Ob\,\lambda I.\Gamma\,I \Rightarrow^* \mathsf{El}\,Ob$, and its universal property. We define the first component of $\Pi^+_{\mathcal{D}} : \big(A : \mathsf{Tm}\,(\mathsf{Ty}^+_{\mathcal{D}}\,\Gamma)\big) \to \mathsf{Tm}\,\big(\mathsf{Ty}_{\mathcal{D}}\,(\Gamma \triangleright_{\mathcal{D}} A)\big) \to \mathsf{Tm}\,(\mathsf{Ty}_{\mathcal{D}}\,\Gamma)$ by $\Pi^+_{\mathcal{D}}\,A\,B \cdot I \cdot^* \gamma_I := B \cdot (\triangleright_A \cdot I \cdot^* \gamma_I) \cdot^* (\gamma_I[\mathsf{p}_A]_\Gamma, \mathsf{q}_A)$ where $\triangleright_A$, p_A and q_A are components in the input A. Note the careful distinguishing of metatheoretic function application, ·s and ·*s. ◀

▶ **Problem 22** ($\mathcal{E} := \mathsf{PSh}(\mathcal{D})$). *The* Ty*-valued presheaves over* $\mathcal{D}$ *are a first-order model of ToS*$^+$*. We name this model* $\mathcal{E}$*.*

Proof. $\mathsf{Con}_{\mathcal{E}}$ is defined as $(\Psi : \mathsf{Con}_{\mathcal{D}} \to \mathsf{Ty}) \times (-[-]_\Psi : \mathsf{Tm}\,(\Psi\,\Gamma) \to \mathsf{Sub}_{\mathcal{D}}\,\Delta\,\Gamma \to \mathsf{Tm}\,(\Psi\,\Delta)) \times (\text{functoriality})$. Types are Ty-valued dependent presheaves, terms are sections, context extension $\triangleright_{\mathcal{E}}$ and $\Sigma_{\mathcal{E}}$ are given by Σ. $\mathsf{U}_{\mathcal{E}}$, $\mathsf{El}_{\mathcal{E}}$, $\Pi_{\mathcal{E}}$ are given by $\mathsf{Ty}_{\mathcal{D}}$, $\mathsf{Tm}_{\mathcal{D}}$, $\triangleright_{\mathcal{D}}$, respectively. $\mathsf{Eq}_{\mathcal{E}}$ is pointwise Eq, its restriction operation and $\mathsf{reflect}_{\mathcal{E}}$ use $\mathsf{reflect}$. $\mathsf{U}^+_{\mathcal{E}}$, $\mathsf{el}^+_{\mathcal{E}}$, $\pi^+_{\mathcal{E}}$ are defined by $\mathsf{Ty}^+_{\mathcal{D}}$, identity and $\Pi^+_{\mathcal{D}}$, respectively. ◀

▶ **Construction 23** (SOGAT → GAT translation). *Given an* $\Omega : \mathsf{Ty}\,\diamond$ *in the first-order syntax of ToS*$^+$*, its GAT translation is* $\Sigma\,\mathsf{Cat}^\diamond\,\lambda C.[\![\Omega]\!]_{\mathcal{E}(C)}\,\diamond_{\mathcal{D}(C)}\,\mathsf{tt}$ *where we explicitly marked that* $\mathcal{D}$ *and* $\mathcal{E}$ *depend on* C*.*

Now we can reuse the semantics of GATs [41, Chapter 4] for any SOGAT, e.g. there is a category of models with an initial object, notions of dependent/displayed models, sections, induction is equivalent to initiality, free models, cofree models [43].

Our running example assuming $C : \mathsf{Tm}\,\mathsf{Cat}^\diamond$ (its first two components named Ob, Hom):

$$
\begin{aligned}
&\Big[\!\!\Big[\Sigma\,\mathsf{U}^+\Big(\big((\mathsf{q} \Rightarrow^+ \mathsf{el}^+\,\mathsf{q}) \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})\big) \times \big(\mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{el}^+\,\mathsf{q} \Rightarrow \mathsf{El}\,(\mathsf{el}^+\,\mathsf{q})\big)\Big)\Big]\!\!\Big]_{\mathcal{E}}\,\diamond_{\mathcal{D}}\ \mathsf{tt} = \\
&\Sigma\,(\mathsf{Ty}^+_{\mathcal{D}}\,\diamond_{\mathcal{D}})\,\lambda Tm.\mathsf{Tm}_{\mathcal{D}}\,\big(\diamond \triangleright (Tm \Rightarrow^+_{\mathcal{D}} Tm)\big)\,(Tm[\mathsf{p}]) \times \mathsf{Tm}_{\mathcal{D}}\,(\diamond \triangleright Tm \triangleright Tm[\mathsf{p}])\,(Tm[\mathsf{p}][\mathsf{p}]) = \\
&\Sigma\Big(\Sigma\,(Ob \Rightarrow \mathsf{U})\,\lambda Tm.\Sigma\,\big(\Pi\,Ob\,\lambda I.Tm \cdot I \Rightarrow \Pi\,Ob\,\lambda J.Hom \cdot J \cdot I \Rightarrow \mathsf{El}\,(Tm \cdot J)\big)\ldots \\
&\quad \Sigma\,(Ob \Rightarrow \mathsf{El}\,Ob)\ \ldots\Big)\,\lambda(Tm,\ldots,\triangleright_{Tm},\ldots).\Sigma\,\big(\Sigma\,(\Pi\,Ob\,\lambda I.Tm \cdot (\triangleright \cdot I) \Rightarrow \mathsf{El}\,(Tm \cdot I)\big)\ \ldots) \\
&\qquad \lambda lam.\Sigma\big(\Pi\,Ob\,\lambda I.Tm \cdot I \Rightarrow Tm \cdot I \Rightarrow \mathsf{El}\,(Tm \cdot I)\big)\ldots
\end{aligned}
$$

The second line is the same as for the direct semantics, but now $\mathcal{D}$ is defined using the curried function space, which removes the extra $\mathbb{1}$s and makes application curried when we unfold even more. As we now compute a formal signature in Ty, we do not use implicit arguments, and use λ for binders. The only difference from Definition 4 is that the components for $\mathsf{Cat}^\diamond$, Tm and lam are separate (flat) Σ types, rather than one flat iterated Σ.

We implemented the SOGAT → GAT translation in Agda using partial deep embeddings of ToS and ToS$^+$. It computes the expected GAT signatures for a number of SOGAT examples. It is available on the first author's website with readable versions of the examples.

The GAT semantics was defined relative to the syntax of ToS. However, it works for any model of ToS: if we use the standard model of ToS (set model, metacircular interpretation where $\mathsf{Con} = \mathsf{Set}$, $\mathsf{Ty}\,\Gamma = \Gamma \to \mathsf{Set}$, $\mathsf{Tm}\,\Gamma\,A = (\gamma : \Gamma) \to A\,\gamma$) instead of the syntax, we obtain another notion of model for each SOGAT signature. We show that this notion of model is isomorphic to the direct semantics from the previous section.

▶ **Theorem 24.** *For any SOGAT signature, the direct semantics and the GAT semantics over the standard model yield isomorphic notions of models.*

Proof. We work in presheaves over the standard model of ToS. We observe that in this model U and Ty are Russell-universes and are closed under type formers Σ, Π, Eq without the restrictions we have in the syntax of ToS. We reformulate Definition 18 in this internal language: the category C becomes an element of $\mathsf{Tm}\,\mathsf{Cat}^\diamond$, the $\mathcal{D}' := \mathsf{PSh}(C)$ is a CwF$^+$ with Ty-valued types and terms. We compare this $\mathcal{D}'$ and the $\mathcal{D}$ given by Problem 21: we define $\alpha : \mathcal{D} \to \mathcal{D}'$ as a strict CwF$^+$-morphism which is bijective on Ty, Ty^+ and Tm. The content of α is mapping in and out of the inductive-recursive universe U^*. We denote $\mathcal{E} := \mathsf{PSh}(\mathcal{D})$ and $\mathcal{E}' := \mathsf{PSh}(\mathcal{D}')$. Precomposition with α is $\alpha^* : \mathsf{PSh}(\mathcal{D}') \to \mathsf{PSh}(\mathcal{D})$ which is a strict CwF-morphism. Now, by induction on the syntax of ToS⁺, we define β for contexts, substitutions, types and terms: $\beta_\Gamma : \mathsf{Sub}_{\mathcal{E}}\,[\![\Gamma]\!]_{\mathcal{E}}\,(\alpha^*\,[\![\Gamma]\!]_{\mathcal{E}'})$, $\beta_\gamma : \beta_\Gamma \circ [\![\gamma]\!]_{\mathcal{E}} = \alpha^*\,[\![\gamma]\!]_{\mathcal{E}'} \circ \beta_\Delta$, $\beta_A : [\![A]\!]_{\mathcal{E}} \cong \alpha^*\,[\![A]\!]_{\mathcal{E}'}[\beta_\Gamma]$, $\beta_a : \beta_A[\mathsf{id}, [\![a]\!]_{\mathcal{E}}] = \alpha^*\,[\![a]\!]_{\mathcal{E}'}[\beta_\Gamma]$. Now for a signature $\Omega : \mathsf{Ty}\,\diamond$, from β_Ω we have $[\![\Omega]\!]_{\mathcal{E}}\ \diamond_{\mathcal{D}}\ * \cong \alpha^*\,[\![\Omega]\!]_{\mathcal{E}'}[\beta_\diamond]\ \diamond_{\mathcal{D}}\ * = [\![\Omega]\!]_{\mathcal{E}'}\ \diamond_{\mathcal{D}'}\ *$. ◀

▶ **Corollary 25.** *By combining the isomorphisms of Theorems 19 and 24: for any SOGAT signature, in presheaves over any of its first-order models, a second-order model is available.*

7 Extensions and variants

The translation also works in the case when signatures are open (can refer to external types like $\mathbb{N}$ in Definition 7). In this case the theory of signatures is defined in the outer layer of a two-level type theory where the inner layer is any chosen CwF, and signatures can refer to the universe Set° of inner types [41, Chapter 3]. The theory of possibly open signatures includes a type former $\hat{\Pi} : (A : \mathsf{Set}^\circ) \to (A \to \mathsf{Ty}) \to \mathsf{Ty}$. Similarly, for infinitary signatures, we have a type former $\tilde{\pi} : (A : \mathsf{Set}^\circ) \to (A \to \mathsf{Tm}\,\mathsf{U}) \to \mathsf{Tm}\,\mathsf{U}$. When supporting infinitary operations, we have to replace the general Eq type by an equality of types in U. This is because the semantics of infinitary GATs is not compatible with sort equations [41, Chapter 5].

Our translation from SOGAT to GAT is not canonical: for example, we could have used semicategories instead of categories. There is also a minimalistic version of the translation which results in a single substitution calculus (SSC), which does not involve a category (single substitutions are not composable). For the SOGAT $\Sigma\,\mathsf{U}\,\lambda Ty.Ty \Rightarrow \mathsf{U}^+$, the parallel translation results in the GAT known as CwF. The SSC translation for the same SOGAT gives a smaller theory: there is no composition or identity substitution, no empty substitution ϵ and no $-,-$ operator for building substitutions into extended contexts. We have $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$, $\langle - \rangle : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A)$ and $-^+ : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A)$. There are four equations for types: $A[\mathsf{p}][\gamma^+] = A[\gamma][\mathsf{p}]$, $A[\mathsf{p}][\langle b \rangle] = A$, $A[\langle b \rangle][\gamma] = A[\gamma^+][\langle b[\gamma] \rangle]$, $A[\mathsf{p}^+][\langle \mathsf{q} \rangle] = A$ and four equations for terms: $\mathsf{q}[\langle b \rangle] = b$, $\mathsf{q}[\gamma^+] = \mathsf{q}$, $b[\mathsf{p}][\gamma^+] = b[\gamma][\mathsf{p}]$, $b[\mathsf{p}][\langle a \rangle] = b$. The resulting theory is a minimalistic variant of B systems [1]. CwFs are models of the resulting theory, but not the other way. The syntaxes are however equivalent (the situation is analogous to the relationship of lambda calculus and combinatory logic [8]).

With small modifications, the translation described in Section 6 can be used to obtain the SSC translation of a GAT. We only change the construction for Problem 21: $\mathcal{C}$ is not a category, just a graph with a vertex $\diamond$; $\mathsf{Con}_{\mathcal{D}}$ and $\mathsf{Ty}_{\mathcal{D}}$ do not include functoriality equations; $A : \mathsf{Ty}^+_{\mathcal{D}}\,\Gamma$ includes $\triangleright_A$, but not the usual universal property; instead we have p_A, q_A, $\langle - \rangle_A$, $-^{+_A}$ operations and the above described 8 equations.

8 Conclusions and further work

In this paper we described SOGAT signatures and translations from SOGAT signatures to GAT signatures. Correctness of our parallel substitution-based translation was shown by constructing an isomorphism with the naive semantics, and was validated by several examples. In the future we would like to show equivalence with Uemura's semantic definition of SOGATs. We would like to computer check our constructions possibly using strict presheaves [44]. It would be interesting to understand the exact relationship between our parallel and single substitution calculi: we conjecture that for any SOGAT, they yield equivalent syntaxes.

We hope that our paper makes a step towards proof assistants with SOGAT support. In such a system, the user could specify the signature for a SOGAT using a built-in ToS⁺, and would automatically obtain notions of first-order and second-order models, morphisms, iterators, induction principles (also for second-order displayed models [15]), and so on.

References

1. Benedikt Ahrens, Jacopo Emmenegger, Paige Randall North, and Egbert Rijke. B-systems and C-systems are equivalent. *The Journal of Symbolic Logic*, page 1–9, 2023. doi:10.1017/jsl.2023.41.
2. Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.FSCD.2019.6, doi:10.4230/LIPICS.FSCD.2019.6.
3. Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type- and scope-safe universe of syntaxes with binding: their semantics and proofs. *J. Funct. Program.*, 31:e22, 2021. doi:10.1017/S0956796820000076.
4. Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending homotopy type theory with strict equality. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. URL: https://doi.org/10.4230/LIPIcs.CSL.2016.21, doi:10.4230/LIPICS.CSL.2016.21.
5. Thorsten Altenkirch, Yorgo Chamoun, Ambrus Kaposi, and Michael Shulman. Internal parametricity, without an interval. *Proc. ACM Program. Lang.*, 8(POPL):2340–2369, 2024. doi:10.1145/3632920.
6. Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for dependent types. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. URL: https://doi.org/10.4230/LIPIcs.FSCD.2016.6, doi:10.4230/LIPICS.FSCD.2016.6.
7. Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.
8. 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.
9. Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.
10. Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *Mathematical Structures in Computer Science*, 33(8):688–743, 2023. doi:10.1017/S0960129523000130.
11. Samy Avrillon. Logic as a second-order generalized algebraic theory, 2023. Report on the 3-month research internship at the Faculty of Informatics of ELTE. URL: https://github.com/MysaaJava/m1-internship/releases/download/project-report/Avrillon-02.pdf.
12. Andrej Bauer, Philipp G. Haselwarter, and Peter LeFanu Lumsdaine. A general definition of dependent type theories. *CoRR*, abs/2009.05539, 2020. URL: https://arxiv.org/abs/2009.05539, arXiv:2009.05539.

13 Rafaël Bocquet. External univalence for second-order generalized algebraic theories. *CoRR*, abs/2211.07487, 2022. URL: `https://doi.org/10.48550/arXiv.2211.07487`, `arXiv:2211.07487`, `doi:10.48550/ARXIV.2211.07487`.

14 Rafaël Bocquet. Towards coherence theorems for equational extensions of type theories. *CoRR*, abs/2304.10343, 2023. URL: `https://doi.org/10.48550/arXiv.2304.10343`, `arXiv:2304.10343`, `doi:10.48550/ARXIV.2304.10343`.

15 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. URL: `https://doi.org/10.4230/LIPIcs.FSCD.2023.18`, `doi:10.4230/LIPICS.FSCD.2023.18`.

16 Paolo Capriotti. Notions of type formers. In Ambrus Kaposi, editor, *23rd International Conference on Types for Proofs and Programs, TYPES 2017*. Eötvös Loránd University, 2017. URL: `http://types2017.elte.hu/proc.pdf#page=77`.

17 John Cartmell. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Log.*, 32:209–243, 1986. `doi:10.1016/0168-0072(86)90053-9`.

18 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. URL: `http://arxiv.org/abs/1904.00827`, `arXiv:1904.00827`.

19 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019. URL: `https://doi.org/10.1016/j.tcs.2019.01.015`, `doi:10.1016/J.TCS.2019.01.015`.

20 Marcelo P. Fiore and Chung-Kil Hur. Second-order equational logic (extended abstract). In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010. `doi:10.1007/978-3-642-15205-4\_26`.

21 Marcelo P. Fiore, Andrew M. Pitts, and S. C. Steenkamp. Quotients, inductive types, and quotient inductive types. *Log. Methods Comput. Sci.*, 18(2), 2022. URL: `https://doi.org/10.46298/lmcs-18(2:15)2022`, `doi:10.46298/LMCS-18(2:15)2022`.

22 Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202. IEEE Computer Society, 1999. `doi:10.1109/LICS.1999.782615`.

23 Jonas Frey. Duality for clans: a refinement of gabriel-ulmer duality. *CoRR*, abs/2308.11967, 2023. URL: `https://doi.org/10.48550/arXiv.2308.11967`, `arXiv:2308.11967`, `doi:10.48550/ARXIV.2308.11967`.

24 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019. `doi:10.1145/3290316`.

25 Daniel Gratzer. Normalization for multimodal type theory. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 2:1–2:13. ACM, 2022. `doi:10.1145/3531130.3532398`.

26 Robert Harper. *Practical Foundations for Programming Languages (2nd. Ed.)*. Cambridge University Press, 2016. URL: `https://www.cs.cmu.edu/%7Erwh/pfpl/index.html`.

27 Robert Harper. An equational logical framework for type theories. *CoRR*, abs/2106.01484, 2021. URL: `https://arxiv.org/abs/2106.01484`, `arXiv:2106.01484`.

28 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`.

29 Philipp G. Haselwarter and Andrej Bauer. Finitary type theories with and without contexts. *J. Autom. Reason.*, 67(4):36, 2023. URL: `https://doi.org/10.1007/s10817-023-09678-y`, `doi:10.1007/S10817-023-09678-Y`.

30 Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

31 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.

32 Jasper Hugunin. Why not W? In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*, volume 188 of *LIPIcs*, pages 8:1–8:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL: https://doi.org/10.4230/LIPIcs.TYPES.2020.8, doi:10.4230/LIPICS.TYPES.2020.8.

33 Jonas Kaiser, Steven Schäfer, and Kathrin Stark. Binder aware recursion over well-scoped de Bruijn syntax. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 293–306. ACM, 2018. doi:10.1145/3167098.

34 Ambrus Kaposi. Formalisation of type checking into algebraic syntax. https://bitbucket.org/akaposi/tt-in-tt/src/master/Typecheck.agda, 2018.

35 Ambrus Kaposi. Quotient inductive-inductive types and higher friends. Talk given at the Homotopy Type Theory Electronic Seminar Talks (HoTTEST), October 2020. URL: https://akaposi.github.io/pres_hottest.pdf.

36 Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for type theory. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.FSCD.2019.25, doi:10.4230/LIPICS.FSCD.2019.25.

37 Ambrus Kaposi and András Kovács. A syntax for higher inductive-inductive types. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. URL: https://doi.org/10.4230/LIPIcs.FSCD.2018.20, doi:10.4230/LIPICS.FSCD.2018.20.

38 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.

39 Ambrus Kaposi, András Kovács, and Ambroise Lafont. For finitary induction-induction, induction is enough. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, volume 175 of *LIPIcs*, pages 6:1–6:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.TYPES.2019.6, doi:10.4230/LIPICS.TYPES.2019.6.

40 András Kovács and Ambrus Kaposi. Large and infinitary quotient inductive-inductive types. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 648–661. ACM, 2020. doi:10.1145/3373718.3394770.

41 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.

42 Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003. doi:10.1016/S0890-5401(03)00088-9.

43 Hugo Moeneclaey. Parametricity and semi-cubical types. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–11. IEEE, 2021. doi:10.1109/LICS52264.2021.9470728.

44 Pierre-Marie Pédrot. Russian constructivism in a prefascist theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE*

Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020, pages 782–794. ACM, 2020. `doi:10.1145/3373718.3394740`.

45 Brigitte Pientka and Jana Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning*, pages 15–21, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

46 Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

47 Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, Andrew Tolmach, and Brent Yorgey. *Programming Language Foundations*, volume 2 of *Software Foundations*. Electronic textbook, 2024. Version 6.5, `http://softwarefoundations.cis.upenn.edu`.

48 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`.

49 Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–15. IEEE, 2021. `doi:10.1109/LICS52264.2021.9470719`.

50 Taichi Uemura. *Abstract and Concrete Type Theories*. PhD thesis, University of Amsterdam, 2021.

51 Philip Wadler, Wen Kokke, and Jeremy G. Siek. *Programming Language Foundations in Agda*. August 2022. URL: `https://plfa.inf.ed.ac.uk/22.08/`.

A More examples of languages as SOGATs

▶ **Definition 26** (Hindley–Milner type system).

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

▶ **Definition 27** (System F).

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

The following language is interesting because its sorts and operations are interleaved: the typing of the sort Tm depends on the operation $*$.

▶ **Definition 28** (System F_ω).

$$
\begin{array}{llll}
\mathsf{Kind} & : \mathsf{Set} & \qquad \mathsf{Tm} & : \mathsf{Ty}\,* \to \mathsf{Set}
\end{array}
$$

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

In the first-order version (minimised by removing Kind variables), we have sorts $\mathsf{Kind} : \mathsf{Set}$, $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Kind} \to \mathsf{Set}$, an operation $* : \mathsf{Kind}$, and a sort $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,* \to \mathsf{Set}$. We have three operations binding Ty-variables and one operation binding a term-variable:

$$
\begin{array}{ll}
\mathsf{LAM} : \mathsf{Ty}\,(\Gamma \triangleright_{\mathsf{Ty}} K)\,L \to \mathsf{Ty}\,\Gamma\,(K \Rrightarrow L) & \mathsf{Lam} : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Ty}} K)\,A \to \mathsf{Tm}\,\Gamma\,(\forall A) \\
\forall \quad : \mathsf{Ty}\,(\Gamma \triangleright_{\mathsf{Ty}} K)\,* \to \mathsf{Ty}\,\Gamma\,* & \mathsf{lam} \;\; : \mathsf{Tm}\,(\Gamma \triangleright_{\mathsf{Tm}} A)\,(B[\mathsf{p}_{\mathsf{Tm}}]_{\mathsf{Ty}}) \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)
\end{array}
$$

The language of fine-grain call by value is to Freyd categories [42] as simply typed lambda calculus is to cartesian closed categories. Here we add some type formers and a fixpoint operator for illustration. All variables are values (in Val).

► **Definition 29** (Fine-grain call by value)**.**

$$
\begin{array}{llll}
\mathsf{Ty} & : \mathsf{Set} & - \Rightarrow - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{Val} & : \mathsf{Ty} \to \mathsf{Set} & \mathsf{lam} & : (\mathsf{Val}\,A \to \mathsf{Tm}\,B) \to \mathsf{Val}\,(A \Rightarrow B) \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} & - \cdot - & : \mathsf{Val}\,(A \Rightarrow B) \to \mathsf{Val}\,A \to \mathsf{Tm}\,B \\
\mathsf{return} & : \mathsf{Val}\,A \to \mathsf{Tm}\,A & \Rightarrow\beta & : \mathsf{lam}\,f \cdot a = f\,a \\
- \gg\!= - & : \mathsf{Tm}\,A \to (\mathsf{Val}\,A \to \mathsf{Tm}\,B) \to \mathsf{Tm}\,B & - \times - & : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\mathsf{idl} & : \mathsf{return}\,a \gg\!= f = f\,a & -,- & : \mathsf{Val}\,A \to \mathsf{Val}\,B \to \mathsf{Val}\,(A \times B) \\
\mathsf{idr} & : m \gg\!= \mathsf{return} = m & \mathsf{case}\times & : \mathsf{Val}\,(A \times B) \to \\
\mathsf{ass} & : (m \gg\!= f) \gg\!= g = & & \quad (\mathsf{Val}\,A \to \mathsf{Val}\,B \to \mathsf{Tm}\,C) \to \mathsf{Tm}\,C \\
 & \quad m \gg\!= (\lambda a.f\,a \gg\!= g) & \times\beta & : \mathsf{case}\times\,(a, b)\,f = f\,a\,b \\
\mathsf{T} & : \mathsf{Ty} \to \mathsf{Ty} & \mathsf{fix} & : (\mathsf{Val}\,(\mathsf{T}\,A) \to \mathsf{Tm}\,A) \to \mathsf{Tm}\,A \\
\mathsf{thunk} & : \mathsf{Tm}\,A \cong \mathsf{Val}\,(\mathsf{T}\,A) : \mathsf{force} & \mathsf{fix}\beta & : \mathsf{fix}\,f = f\,\big(\mathsf{thunk}\,(\mathsf{fix}\,f)\big)
\end{array}
$$

The next definition adds Σ, 0, 1, 2 and W-types to minimal Martin-Löf type theory, which is enough to encode all inductive types [32].

► **Definition 30** (Martin-Löf type theory with inductive types)**.** *We extend Definition 7 with the following.*

$$
\begin{array}{ll}
\Sigma & : (A : \mathsf{Ty}\,i) \to (\mathsf{Tm}\,A \to \mathsf{Ty}\,i) \to \mathsf{Ty}\,i \\
(-,-) & : (a : \mathsf{Tm}\,A) \times \mathsf{Tm}\,(B\,a) \cong \mathsf{Tm}\,(\Sigma\,A\,B) : \mathsf{fst}, \mathsf{snd} \\
\bot & : \mathsf{Ty}\,0 \\
\mathsf{exfalso} & : \mathsf{Tm}\,\bot \to \mathsf{Tm}\,A \\
\top & : \mathsf{Ty}\,0 \\
\mathsf{tt} & : \top \cong \mathsf{Tm}\,\top \\
\mathsf{Bool} & : \mathsf{Ty}\,0 \\
\mathsf{true} & : \mathsf{Tm}\,\mathsf{Bool} \\
\mathsf{false} & : \mathsf{Tm}\,\mathsf{Bool}
\end{array}
$$

$$
\begin{aligned}
&\mathsf{indBool} : (C : \mathsf{Tm}\,\mathsf{Bool} \to \mathsf{Ty}\,i) \to \mathsf{Tm}\,(C\,\mathsf{true}) \to \mathsf{Tm}\,(C\,\mathsf{false}) \to \\
&\qquad\qquad (b : \mathsf{Tm}\,\mathsf{Bool}) \to \mathsf{Tm}\,(C\,b) \\
&\mathsf{Bool}\beta_1 \;\; : \mathsf{indBool}\,t\,f\,\mathsf{true} = t \\
&\mathsf{Bool}\beta_2 \;\; : \mathsf{indBool}\,t\,f\,\mathsf{false} = f \\
&\mathsf{Id} \qquad\; : (A : \mathsf{Ty}\,i) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,A \to \mathsf{Ty}\,i \\
&\mathsf{refl} \qquad : (a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(\mathsf{Id}\,a\,a) \\
&\mathsf{J} \qquad\;\; : \big(C : (x : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(\mathsf{Id}\,A\,a\,x) \to \mathsf{Ty}\,i\big) \to \\
&\qquad\qquad \mathsf{Tm}\,\big(C\,a\,(\mathsf{refl}\,a)\big) \to (x : \mathsf{Tm}\,A)(e : \mathsf{Tm}\,(\mathsf{Id}\,A\,a\,x)) \to \mathsf{Tm}\,(C\,x\,e) \\
&\mathsf{Id}\beta \qquad : \mathsf{J}\,C\,w\,a\,(\mathsf{refl}\,a) = w \\
&\mathsf{W} \qquad\; : (S : \mathsf{Ty}\,i) \to (\mathsf{Tm}\,S \to \mathsf{Ty}\,i) \to \mathsf{Ty}\,i \\
&\mathsf{sup} \qquad : (s : \mathsf{Tm}\,S) \to (\mathsf{Tm}\,(P\,s) \to \mathsf{W}\,S\,P) \to \mathsf{W}\,S\,P \\
&\mathsf{indW} \quad\; : (C : \mathsf{Tm}\,(\mathsf{W}\,S\,P) \to \mathsf{Ty}\,i) \to \\
&\qquad\qquad \Big(\big((p : \mathsf{Tm}\,(P\,s)) \to \mathsf{Tm}\,(C\,(f\,p))\big) \to \mathsf{Tm}\,\big(C\,(\mathsf{sup}\,s\,f)\big)\Big) \to \\
&\qquad\qquad (w : \mathsf{Tm}\,(\mathsf{W}\,\mathsf{S}\,\mathsf{P})) \to \mathsf{Tm}\,\big(C\,w\big) \\
&\mathsf{W}\beta \qquad : \mathsf{indW}\,C\,h\,(\mathsf{sup}\,s\,f) = h\,(\lambda p.\mathsf{indW}\,C\,h\,(f\,p))
\end{aligned}
$$