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.FSCD.2024.10

Supplementary Material  Software (Implementation and detailed constructions): https://bitbucket.org/akaposi/sogat, archived at swh:1:dir:0f9fefbd89f1151eedb46518bde3062c0bc5a3fe

Funding  Ambrus Kaposi: The author was supported by project no. TKP2021-NVA-29 which has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

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 [48, 49, 53]. If instead the syntax is defined using well-scoped syntax trees [34, 27, 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, 53] 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, 20], typechecking [35] 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) [39]. 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 [19]. Logical frameworks [29, 47] and higher-order abstract syntax (HOAS) [32] 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 \((\text{Tm} \rightarrow \text{Tm}) \rightarrow \text{Tm}\). The justification of HOAS is the type-theoretic internal language of presheaves over the category of contexts and syntactic substitutions [32]. 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 [23, 21, 2], SOGATS were first used by Uemura [52] 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 [40]. 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 [51] 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 [50, 16]. Substructural (e.g. linear or modal) type theories are not definable as SOGATS using the method described in this paper, but sometimes presheaves over a substructural theory provide a substructural internal language which can be used to describe the theory, as in the case of multi-modal type theory [26].

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 [18], and they can be described presentation-independently as contextual categories [18], categories with families (CwFs) or clans [24]. The “theory of signatures” (ToS) approach [39] 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 [43], while still working directly with signatures. SOGATS again can be defined syntactically [52] or presentation-independently as representable map categories [52] or CwFs with locally representable types [14]. The current paper contributes the ToS style definition of SOGATS (we leave the proof of equivalence with the former definitions as future work). 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 [42].
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, System F(ω), Martin-Löf type theory. We list more examples in Appendix A including the lambda cube. 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 [38] for a higher variant of GATs (higher inductive-inductive types), and was used to describe ordinary [39] and infinitary [42] GATs (quotient inductive-inductive types). The thesis of Kovács [43] 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 [36].

Direct precursors of our work are Hofmann’s analysis of higher-order abstract syntax (HOAS) [32] and Capriotti’s rule framework [17]. Syntactic definitions of SOGATs are given in Uemura’s thesis [52] and Harper’s equational logical framework [28]. A syntactic definition of type theories (SOGATs with two fixed sorts: types and terms) is described by Bauer and Haselwarter [30] based on earlier work [13]. Presentation-independent definitions of SOGATs are representable map categories by Uemura [52] and CwFs with a sort of locally representable types (CwF*) [15]. 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 C together with a CwF*-morphism from this CwF* to the CwF* of presheaves over 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. Synthetic Tait computability [50] and
internal sconing [16] 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 \( \text{Set}_i \), we usually
omit the level subscripts. We use infix \( \Sigma \) type notation using \( \times \), the single element of the
singleton type \( 1 \) is denoted \( \star \). 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].
2.2 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).

\[
\begin{align*}
\text{Ty} & : \text{Set} & K & : \text{Tm} (A \Rightarrow B \Rightarrow A) \\
\text{Tm} & : \text{Ty} \rightarrow \text{Set} & S & : \text{Tm} ((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C) \\
\iota & : \text{Ty} & K\beta & : K \cdot u \cdot f = u \\
\Rightarrow & : \text{Ty} \rightarrow \text{Ty} \rightarrow \text{Ty} & S\beta & : S \cdot f \cdot g \cdot u = f \cdot u \cdot (g \cdot u) \\
\cdot & : \text{Tm} (A \Rightarrow B) \rightarrow \text{Tm} A \rightarrow \text{Tm} B
\end{align*}
\]

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 \(\text{Tm}\) arguments, but they still take two and three \(\text{Ty}\) 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 [43]. In particular, each GAT has a syntax given by a quotient inductive-inductive type [39], we have free models [43] and cofree models [45].

If the language has variables or binders, we will define it as a second-order theory.

2.3 Second-order algebraic theories

The SOAT of lambda calculus is the following.

Definition 3 (Lambda calculus).

\[
\begin{align*}
\text{Tm} & : \text{Set} & \text{lam} & : (\text{Tm} \rightarrow \text{Tm}) \rightarrow \text{Tm} & \cdot & : \text{Tm} \rightarrow \text{Tm} \rightarrow \text{Tm} & \beta & : \text{lam} \cdot f \cdot u = f \cdot u
\end{align*}
\]

The type of \(\text{lam}\) 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 \(\text{lam}\) satisfying the equation \(\beta\)). However, we do not have a usable notion of homomorphism between second-order models \(M\) and \(N\): this would be a function \(\alpha : \text{Tm}_M \rightarrow \text{Tm}_N\) such that \(\alpha (t \cdot_M u) = \alpha t \cdot_N \alpha u\) and \(\alpha (\text{lam}_M f) = \text{lam}_N (\alpha \circ f \circ ?)\), 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 \(\text{Tm}\) and all operations by contexts and then \(\text{lam}\) becomes a first order function taking a term in an extended context as input. The resulting GAT is the following.
We explain in more detail how we obtained the GAT of Definition 4 from the SOAT of Definition 4. By the syntax of lambda calculus, we mean the syntax for the GAT of Definition 4. 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 will work generically for any SO(G)AT, hence it does not necessarily give the most minimal GAT presentation.

By the syntax of lambda calculus, we mean the syntax for the GAT of Definition 4. We explain in more detail how we obtained the GAT of Definition 4 from the SOAT of Definition 4. By the syntax of lambda calculus, we mean the syntax for the GAT of Definition 4. 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 will work generically for any SO(G)AT, hence it does not necessarily give the most minimal GAT presentation.
A first-order model of the simply typed lambda calculus contains a category with a terminal object (\textbf{Con}, \textbf{Sub} and the empty context $\emptyset$), two sorts \textbf{Ty} and \textbf{Tm} which are both indexed by contexts, and there are context extension operations both for types and terms (we omit the types of some operations and equations which are the same as in Definition 4):

\[
\begin{align*}
A & \Rightarrow B & \text{lam} \cdot x \cdot b : A \Rightarrow B & f : A \Rightarrow B & a : A \\
& & (\text{lam} \cdot x \cdot b) \cdot a = b(x \mapsto a) & f \cdot a : B & f = \text{lam} \cdot x \cdot f \cdot x
\end{align*}
\]
Definition 6 (Simply typed lambda calculus as a GAT with both type and term variables).

\[
\begin{align*}
\text{Con, Sub}, \ - & \circ \ - , \text{id}, \text{ass}, \text{idl}, \text{idr}, \odot, \epsilon, \eta \\
\text{Ty} & : \text{Con} \to \text{Set} \\
[-]_{\text{Ty}} & : \text{Ty} \Gamma \to \text{Sub} \Delta \Gamma \to \text{Ty} \Delta \\
\circ \cdot \text{Ty}, \ [\text{id}]_{\text{Ty}} & : \text{Con} \to \text{Con} \\
p_{\text{Ty}} \circ \ - , q_{\text{Ty}} [-] & : \text{Sub} \Delta (\Gamma \to \text{Ty} \Gamma) \cong \text{Sub} \Delta \Gamma \times \text{Ty} \Delta : - \cdot \text{Ty} - \\
\text{Tm} & : (\Gamma : \text{Con}) \to \text{Ty} \Gamma \to \text{Set} \\
[-]_{\text{Tm}} & : \text{Tm} \Gamma \Delta \to (\gamma : \text{Sub} \Delta \Gamma) \to \text{Tm} \Delta (A[\gamma]_{\text{Ty}}) \\
\circ \cdot \text{Tm}, \ [\text{id}]_{\text{Tm}} & : (\Gamma : \text{Con}) \to \text{Ty} \Gamma \to \text{Con} \\
p_{\text{Tm}} \circ \ - , q_{\text{Tm}} [-] & : \text{Sub} \Delta (\Gamma \to \text{Tm} \Delta A) \cong (\gamma : \text{Sub} \Delta \Gamma) \times \text{Tm} \Delta (A[\gamma]_{\text{Tm}}) : - \cdot \text{Tm} - \\
\Rightarrow [\text{Ty}] & : (A \Rightarrow B)[\gamma] = (A[\gamma]) \Rightarrow (B[\gamma]) \\
\text{lam} & : \text{Tm} (\Gamma \to \text{Tm} \Delta A) (B[p_{\text{Tm}}]) \to \text{Tm} \Gamma (A \Rightarrow B) \\
\text{lam} [\text{}] & : (\text{lam } t)[\gamma] = \text{lam } (t[\gamma \circ p_{\text{Tm}} \cdot \text{Tm } q_{\text{Tm}}]) \\
\Rightarrow \beta & : \text{Tm} \Gamma (A \Rightarrow B) \to \text{Tm} \Gamma \Delta A \Rightarrow \text{Tm} \Gamma B \\
\Rightarrow [\text{}] & : (t \cdot u)[\gamma] = t[\gamma] \cdot (u[\gamma]) \\
\Rightarrow \eta & : t = \text{lam } (t[p_{\text{Tm}}] \cdot q_{\text{Tm}})
\end{align*}
\]

The context extension operations take as arguments the index of the corresponding sort: Ty is not indexed, so \(\cdot \text{Ty} \) does not take any arguments, \(\cdot \text{Tm} \) 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 \(\cdot \text{Ty} \) and the type variables \(q_{\text{Ty}}, q_{\text{Tm}}[p], q_{\text{Ty}}[p][p] \), and so on. In the formal version of signatures (Definition 13), 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 \(\text{Ty} : \text{Con} \to \text{Set} \) by \(\text{Ty} : \text{Set} \) and removing the \(\cdot \text{Ty} \) operation. The operations in the notion of first-order model are the typed versions of the operations in Definition 4. Lambda and application could have been presented by an isomorphism \(\text{lam} : \text{Tm} (\Gamma \to \text{Tm} \Delta A) (B[p_{\text{Tm}}]) \cong \text{Tm} \Gamma (A \Rightarrow B) : \text{app} \), using a unary app application operation instead of the binary \(\cdot \cdot \) . Our setup allows choosing between the two versions, see the discussion after Definition 13. 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 \(\Rightarrow \).
Definition 7 (Minimal intuitionistic first-order logic).

\[
\begin{align*}
\text{For} & : \text{Set} & \text{Pf} & : \text{For} \rightarrow \text{Prop} \\
\text{Ty} & : \text{Ty} & \text{Tm} & : \text{Set} \quad \text{intro}^\wedge : (\text{Pf} A \rightarrow \text{Pf} B) \rightarrow \text{Pf} (A \supset B) : \text{elim}^\wedge \\
\rightarrow \supset \rightarrow & : \text{For} \rightarrow \text{For} \rightarrow \text{For} & \text{intro}^\forall : ((t : \text{Tm}) \rightarrow \text{Pf} (A t)) \leftrightarrow \text{Pf} (\forall A) : \text{elim}^\forall \\
\forall & : (\text{Tm} \rightarrow \text{For}) \rightarrow \text{For} & \text{elim}^\text{Eq} : (A : \text{Tm} \rightarrow \text{For}) \rightarrow \text{Pf} (\text{Eq} t t') \rightarrow \\
\text{Eq} & : \text{Tm} \rightarrow \text{Tm} \rightarrow \text{For} & \text{Pf} (A t) \rightarrow \text{Pf} (A t') \\
\end{align*}
\]

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 Bruijn indices \( q_{\text{For}}, q_{\text{Tm}}, q_{\text{Pf}} \), nine different 1 De Bruijn indices \( q_{\text{For}}[\text{Pf}], q_{\text{For}}[\text{Tm}]_{\text{For}}, q_{\text{For}}[\text{Pp}]_{\text{For}}, \ldots, q_{\text{Pf}}[\text{Pp}]_{\text{Pf}} \). In general, De Bruijn index \( n \) has \( 3^n+1 \) variants. We list the types of the binders:

\[
\begin{align*}
\forall & : \text{For} (\Gamma \supset \text{Tm}) \rightarrow \text{For} \Gamma \\
\text{intro}^\wedge & : \text{Pf} (\Gamma \supset \text{Pf} A) (B[q_{\text{Pf}}]_{\text{For}}) \rightarrow \text{Pf} \Gamma (A \supset B) \\
\text{intro}^\forall & : \text{Pf} (\Gamma \supset \text{Tm}) A \rightarrow \text{Pf} \Gamma (\forall A) \\
\text{elim}^\text{Eq} & : (A : \text{For} (\Gamma \supset \text{Tm})) \rightarrow \text{Pf} \Gamma (\text{Eq} t t') \rightarrow \text{Pf} \Gamma (A[id, \text{Tm} t]_{\text{For}}) \rightarrow \text{Pf} \Gamma (A[id, \text{Tm} t']_{\text{For}}) \\
\end{align*}
\]

The GAT presentation of first-order logic can be simplified by removing \text{For} variables as no operations bind formulas. Another post-hoc simplification is separating the \text{Tm}-variable contexts and the \text{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 \text{Pf} where we use a full quotient which can be implemented by \text{SProp} of Agda or Coq [25]). 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 without computation rules.

Next we show the SOGAT definition of the polymorphic lambda calculus.

Definition 8 (System F).

\[
\begin{align*}
\text{Ty} & : \text{Set} \\
\text{Tm} & : \text{Ty} \rightarrow \text{Set} \\
\rightarrow \supset \rightarrow & : \text{Ty} \rightarrow \text{Ty} \rightarrow \text{Ty} \\
\text{lam} & : (\text{Tm} A \rightarrow \text{Tm} B) \equiv \text{Tm} (A \Rightarrow B) : \rightarrow \rightarrow \\
\forall & : (\text{Ty} \rightarrow \text{Ty}) \rightarrow \text{Ty} \\
\text{Lam} & : ((X : \text{Ty}) \rightarrow \text{Tm} (A X)) \equiv \text{Tm} (\forall A) : \rightarrow \rightarrow \\
\end{align*}
\]

The first order version is Definition 6 extended with the following operations and equations for \( \forall \). Now we really need both type and term variables. We use a unary application operation for \( \forall \), see discussion after Definition 13.

\[
\begin{align*}
\forall & : \text{Ty} (\Gamma \supset \text{Ty}) \rightarrow \text{Ty} \Gamma \\
\text{Lam} & : \text{Tm} (\Gamma \supset \text{Ty}) A \equiv \text{Tm} \Gamma (\forall A) : \text{App} \\
\forall[\gamma] & : (\forall A)[\gamma] = \forall (A[\gamma \circ p_{\text{Ty}} \supset \text{Ty} q_{\text{Ty}}]) \\
\text{Lam}[\gamma] & : (\text{Lam} t)[\gamma] = \text{Lam} (t[\gamma \circ p_{\text{Ty}} \supset \text{Ty} q_{\text{Ty}}]) \\
\end{align*}
\]

The next language is interesting because its sorts and operations are interleaved: the typing of the sort \text{Tm} depends on the operation \( * \).
10:10 Second-Order Generalised Algebraic Theories: Signatures and First-Order Semantics

Definition 9 (System $F_{\omega}$).

\[
\begin{align*}
\boxempty &: \text{Set} & \text{Tm} &: \text{Ty} \rightarrow \text{Set} \\
\text{Ty} &: \boxempty \rightarrow \text{Set} & \forall &: (\text{Ty} K \rightarrow \text{Ty} \ast) \rightarrow \text{Ty} \\
\Rightarrow &: \boxempty \rightarrow \boxempty \rightarrow \boxempty & \text{Lam} &: ((X : \text{Ty} K) \rightarrow \text{Tm} (A X)) \equiv \\
\text{LAM} &: (\text{Ty} K \rightarrow \text{Ty} L) \equiv & \text{Tm} (\forall A) &: \boxempty \rightarrow \boxempty \\
& & \text{Ty} (K \Rightarrow L) &: \boxempty \Rightarrow \boxempty \\
\ast &: \boxempty & \text{Lam} &: (\text{Tm} A \rightarrow \text{Tm} B) \equiv \text{Tm} (A \Rightarrow B) : \ldots
\end{align*}
\]

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

\[
\begin{align*}
\text{LAM} &: \text{Ty} (\Gamma \Rightarrow \text{Ty} K) L \rightarrow \text{Ty} \Gamma (K \Rightarrow L) & \text{Lam} &: \text{Tm} (\Gamma \Rightarrow \text{Ty} K) A \rightarrow \text{Tm} \Gamma (\forall A) \\
\forall &: \text{Ty} (\Gamma \Rightarrow \text{Ty} K) \ast \rightarrow \text{Ty} \Gamma \ast & \text{lamb} &: \text{Tm} (\Gamma \Rightarrow \text{Tm} A) (B[p_{\text{Tm}}]_{\text{Ty}}) \rightarrow \text{Tm} \Gamma (A \Rightarrow B)
\end{align*}
\]

Our next example is a theory with dependent types featuring $\Pi$ 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 $N$ as a new sort and $0$ and $1 + -$ as new operations).

Definition 10 (Minimal Martin-Löf type theory).

\[
\begin{align*}
\text{Ty} &: \text{N} \rightarrow \text{Set} & \text{U} &: (i : \text{N}) \rightarrow \text{Ty} (1 + i) \\
\text{Tm} &: \text{Ty} i \rightarrow \text{Set} & \text{c} &: \text{Ty} i \cong \text{Tm} (U i) : \text{El} \\
\Pi &: (A : \text{Ty} i) \rightarrow (\text{Tm} A \rightarrow \text{Ty} i) \rightarrow \text{Ty} i & \text{Lift} &: \text{Ty} i \rightarrow \text{Ty} (1 + i) \\
\text{lamb} &: ((a : \text{Tm} A) \rightarrow \text{Tm} (B a)) \cong \text{Tm} (\Pi A B) : \ldots & \text{mk} &: \text{Tm} A \cong \text{Tm} (\text{Lift} A) : \text{un}
\end{align*}
\]

The first-order translation of this theory results in a category with families (CwF [19]), more precisely, a category with $\text{N}$-many families equipped with familywise $\Pi$-types, universes and a one-step upwards lifting between the families. The sorts are $\text{Ty} : \text{Con} \rightarrow \text{N} \rightarrow \text{Set}$ and $\text{Tm} : (\Gamma : \text{Con}) \rightarrow \text{Ty} \Gamma i \rightarrow \text{Set}$, the $i$ argument is implicit in the latter.

Instead of a Coquand-universe with $c$ and $\text{El}$, we could have defined a Russell universe where we have a sort equality $\text{Ty} i = \text{Tm} (U i)$, and we also have the option to do this for lifting and $\Pi$ types. The first-order semantics of such a theory has the following equalities where the second one makes sense because of the first one: $\text{Ty} \Gamma i = \text{Tm} \Gamma (U i)$, $A [\gamma]_{\text{Ty}} = A [\gamma]_{\text{Tm}}$. Having strict $\Pi$ types means $\text{Tm} (\Gamma \Rightarrow A) B = \text{Tm} \Gamma (\Pi A B)$ and $t [\gamma] = t [\gamma \circ p, q]$ where the left hand side $t$ is in $\text{Tm} \Gamma (\Pi A B)$.

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 $\text{Srt}$ for the (single) sort, $\Sigma$ types, a $\Pi$ type with fixed $\text{Srt}$ domain, and an equality type. $\Pi$ types are equipped with application, but the $\Sigma$ and $\text{Eq}$ types don’t have constructors or destructors, because those are not needed when defining signatures.
A first-order model of this theory is a CwF with type formers \( \Sigma \), Srt, Eq and a term former \(-\cdot-\) : \( \text{Tm} \Gamma \) (\( \Pi \text{Srt} \) \( B \)) \( \rightarrow \) (\( x : \text{Tm} \Gamma \) \( \text{Srt} \)) \( \rightarrow \) \( \text{Tm} \) \( \Gamma \) (\( B \) [id, \( x \)]). An element of Ty in the syntax of this language is an AT signature. We introduce abbreviations Srt \( \Rightarrow \) \( A \) := \( \Pi \text{Srt} \) \( \lambda \_A \) and \( A \times B \) := \( \Sigma \) \( A \lambda \_B \). The signature for combinator calculus is the following Ty:

\[
\Sigma \text{(Srt} \Rightarrow \text{Srt}) \lambda \text{appp, Srt} \lambda K, \Sigma \text{Srt} \lambda S. \left( \text{II} \text{Srt} \lambda u. \text{II} \text{Srt} \lambda f. \text{Eq} \left( \text{app} \cdot \left( \text{app} \cdot S \cdot f \right) \cdot g \right) \cdot u \right) \\
\times \left( \text{II} \text{Srt} \lambda f. \text{II} \text{Srt} \lambda g. \text{II} \text{Srt} \lambda u. \text{Eq} \left( \text{app} \cdot \left( \text{app} \cdot f \right) \cdot \left( \text{app} \cdot g \right) \cdot u \right) \right)
\]

This can be seen as a more explicit version of Definition 1: we use \( \Sigma \) types instead of a newline-separated list, we use the metatheoretic \( \lambda \) binder to give names to operations, we use an explicit \( \cdot \) operation for application and write \( \text{Eq} \) instead of \( = \). Moreover, we don’t have infix operators or implicit arguments, the three arguments of equation \( K \beta \) and the four arguments of equation \( S \beta \) have to be introduced using \( \Pi \text{Srt} \) explicitly. Being more explicit is needed to make sure that we describe an algebraic theory: for example, the fact that the domain of \( \Pi \) 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 \( \Sigma \) types, equality with reflection, and a \( \Pi \) type with U-domain.

**Definition 12** (ToS: the theory of GAT signatures).

\[
\begin{align*}
\text{Ty} &: \text{Set} \\
\text{Tm} &: \text{Ty} \rightarrow \text{Set} \\
\text{U} &: \text{Ty} \\
\text{El} &: \text{Tm} \text{U} \rightarrow \text{Ty} \\
\text{T} &: \text{Ty} \\
\text{tt} &: 1 \cong \text{Tm} \top
\end{align*}
\]

\[
\begin{align*}
\Sigma &: (A : \text{Ty}) \rightarrow (\text{Tm} A \rightarrow \text{Ty}) \rightarrow \text{Ty} \\
(\cdot, \cdot) &: (a : \text{Tm} A) \times \text{Tm} (B a) \cong \text{Tm} (\Sigma A B) : \text{fst, snd} \\
\Pi &: (a : \text{Tm} U) \rightarrow (\text{Tm} (\text{El} a) \rightarrow \text{Ty}) \rightarrow \text{Ty} \\
\text{lam} &: ((x : \text{Tm} (\text{El} a)) \rightarrow \text{Tm} (B x)) \cong \text{Tm} (\Pi a B) : \cdot \cdot \cdot \\
\text{Eq} &: (A : \text{Ty}) \rightarrow \text{Tm} A \rightarrow \text{Tm} A \rightarrow \text{Ty} \\
\text{refl} &: (u = v) \cong \text{Tm} (\text{Eq} A u v) : \text{reflect}
\end{align*}
\]

The first-order version is Definition 14. A (presentation of a) GAT is defined as a closed type in the syntax of ToS. The base type \( \text{U} \) is for declaring sorts, so a signature has to start with a sort, and then we can declare elements of the sort using \( \text{El} \) 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 \( \beta \) rule for reasons of space.
\[ \Sigma U . \lambda Tm . \Sigma ((Tm \Rightarrow U) . \lambda Tm . \Sigma (Ty \Rightarrow \Sigma (Ty \Rightarrow \Sigma (Ty \Rightarrow El (Ty)) \lambda arr . \Sigma)
\]
\[(\Pi Ty . \lambda A . Ty . \lambda B . Tm . (arr \cdot A \cdot B) \Rightarrow Tm . A \Rightarrow El (Tm \cdot B)) \lambda app . \Sigma
\]
\[(\Pi Ty . \lambda A . Ty . \lambda B . El (Tm \cdot (arr \cdot A \cdot (arr \cdot B \cdot A)))) \lambda K . \Sigma
\]
\[(\Pi Ty . \lambda A . Ty . \lambda B . El (\Pi (Tm \cdot A) . \lambda . El (Tm \cdot B) \lambda f . Eq (El (Tm \cdot A)) (app \cdot (arr \cdot B \cdot A) \cdot B \cdot (app \cdot (arr \cdot A \cdot (arr \cdot B \cdot A)) \cdot A \cdot K \cdot u) \cdot f) \cdot u) \times \ldots
\]

This type is a very explicit version of Definition 2: we use \( \Sigma \), 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 \( \Pi \) type with \( U^* \)-domain and \( U \)-codomain.

▶ Definition 13 (ToS\( ^* \): the theory of SOGAT signatures). We extend ToS with the following.

\[ U^* : Ty \]
\[ \pi^* : (a^* : Tm U^*) \rightarrow (Tm (El (el^* a^*))) \rightarrow Tm U \rightarrow Tm U \]
\[ el^* : Tm U^* \rightarrow Tm U \]
\[ lam^* : (x : El (el^* a^*)) \rightarrow Tm (El (b \cdot x)) \equiv Tm (El (\pi^* a^* b)) : .^* -.^* \]

The first-order version is Definition 15. 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 \( Ty \).

\[ \Sigma U^* . \lambda Tm . \Sigma ((Tm \Rightarrow U) . \lambda Tm . \Sigma (Ty \Rightarrow El (Ty)) \lambda lam . \Sigma (el^* Tm \Rightarrow el^* Tm \Rightarrow El (el^* Tm)) \lambda app.
\]
\[(\Pi (Tm \Rightarrow el^* Tm) \lambda t . El (el^* Tm) . \lambda u . Eq (El (el^* Tm)) (app \cdot (lam \cdot t) \cdot u) (t \cdot \cdot t) u)
\]

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 \( \Rightarrow \cdot \cdot \cdot \) can appear on the left hand side of an \( \Rightarrow \). When stating the \( \beta \) 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 \( \beta \) law (last line in Definition 4, we write \( app \) instead of \( \cdot \) to avoid confusion): \( app (lam t) u = t [id, u] \).

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 \( el^* Tm \Rightarrow el^* Tm \Rightarrow El (el^* Tm) \), and this is translated to \( Tm \Gamma \rightarrow Tm \Gamma \rightarrow Tm \Gamma \) in the GAT version (see Definition 4). But we could have defined \( app \) as having type \( El (Tm \Rightarrow Tm \Rightarrow Tm) \). In this case the GAT version of \( app \) would be in \( Tm (\Gamma \Rightarrow \cdot \cdot \cdot \cdot) \). 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 \( \Rightarrow \cdot \cdot \cdot \) and \( \Rightarrow \cdot \cdot \cdot \cdot \), 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, \( \Pi \) types, and so on, so we directly use these when interpreting the type formers of \( \text{ToS}^+ \). We assume basic working knowledge of categories with families (\text{CwFs} [19]).

Definition 14 (First-order model of \( \text{ToS} \)). A first-order model of \( \text{ToS} \) is a \( \text{CwF} \) (sorts are denoted \( \text{Con}, \text{Sub}, \text{Ty}, \text{Tm} \), the empty context is \( \varnothing \), the empty substitution is \( \varepsilon : \text{Sub} \Delta \varepsilon \), context extension is \( - \to - : (\Gamma : \text{Con}) \to \text{Tm} \Gamma \to \text{Con} \) with \( p \circ -, q [-] : \text{Sub} \Delta (\Gamma \to A) \cong (y : \text{Sub} \Delta \Gamma) \times \text{Tm} \Delta (A[y]) : -, - \) equipped with:

- A universe given by \( U : \text{Tm} \Gamma \to \text{Tm} \Gamma \).
- A function space with domain in \( U \), that is \( \Pi : (a : \text{Tm} \Gamma U) \to \text{Tm} (\Gamma \to \text{El} a) \to \text{Tm} \Gamma \), with an isomorphism \( \lambda \text{m} : \text{Tm} (\Gamma \to \text{El} a) B = \text{Tm} \Gamma (\Pi a B) \) : \text{app}.
- A strict equality type \( \text{Eq} \) with reflection and uniqueness of identity proofs.
- All the operations listed above are natural in \( \Gamma \).

Definition 15 (First-order model of \( \text{ToS}^+ \)). A first-order model of \( \text{ToS}^+ \) is a first-order model of \( \text{ToS} \) equipped with:

- Another universe \( U^+ : \text{Tm} \Gamma \) that is a subuniverse of \( U \) i.e. el\( ^+ \) : \( \text{Tm} \Gamma U \to \text{Tm} \Gamma U^+ \).
- \( U \) is closed under functions with \( U^\ast \)-domain, i.e. \( \pi^\ast : (a^\ast : \text{Tm} \Gamma U^+) \to \text{Tm} (\Gamma \to \text{El} a^\ast) \) : \( \text{Tm} \Gamma U \) with \( \lambda \text{m} \) such that \( \text{Tm} (\Gamma \to \text{El} a^\ast) (\text{El} b) = \text{Tm} \Gamma (\pi^\ast a^\ast b) \) : \text{app}.
- All the operations listed above are natural in \( \Gamma \).

Problem 16 (PSh). Presheaves over a category \( C \) form a \( \text{CwF} \) equipped with \( \tau, \Sigma \) types, an equality type with reflection, \( \Pi \) type and a Coquand-universe \( U \) with \( c : \text{Tm} \Gamma \) : \( \text{Tm} \Gamma U \) : \text{El}.

Unlike in Definition 10, we omit writing universe indices for readability.

Construction. We recall the main parts of the construction [31] for fixing notations. \( \Gamma : \text{Con} \) is a presheaf, that is a family of sets \( \Gamma : C \to \text{Set} \) with reindexing \( \gamma_I(f)_I = f : \Gamma I \to \Gamma J \) for \( \gamma_I : \Gamma I \) and \( f : C(J, I) \) such that \( \gamma_I(f \circ g)_I = \gamma_I(f)_I \gamma_J(g)_I = \gamma_J \). A \( \sigma : \text{Sub} \Delta \Gamma \) is a function \( \sigma : \Delta I \to \Gamma I \) such that \( (\sigma \delta) f = (\delta \sigma) f \Delta \). A type \( A : \text{Tm} \Gamma \) is a dependent presheaf containing a family \( A : (I : C) \to \Gamma I \to \text{Set} \) with reindexing \( a_I(f)_A = \text{A}(\gamma_I(f))_I \) for \( a_I : A \) satisfying functoriality. Type substitution is \( \text{A}(y) I \delta_I : = a I(\gamma_I f)_I \). A term \( a : \text{Tm} I A \) is a function \( a : (\gamma_I : \Gamma I) \to A I \gamma_I \) such that \( (a \gamma_I)_I = a_I(\gamma_I f)_I \). Term substitution is \( a(y) I \delta_I : = a(y) \delta_I \). The empty context is constant unit: \( \varnothing I = 1 \).

Context extension is pointwise: \( (\Gamma \to A) I = (\gamma_I : \Gamma I) \times A I \gamma_I \), its universal property is given by projections and pairing for metatheoretic \( \Sigma \) types, \( \tau, \Sigma \) and \( \text{Eq} \) are pointwise. We have the functor Yoneda \( y : C \to \text{PSh}(C) \) defined by \( y I J : = C(J, I) \), and we use this to define the universe by \( U I \gamma_I := y I \gamma_I \). We observe that \( \gamma_I[-] : \text{Sub} (y I) \Gamma \) (forward part of Yoneda lemma), and define \( \Pi A B I \gamma_I := \text{Tm} (y I \to A[\gamma_I[-] I]) (B[\gamma_I[-] I]_I, p, q) \).

Problem 17 (Locally representable types). The \( \text{CwF} \) of presheaves can be extended to a \( \text{CwF}^+ \), which means a \( \text{CwF} \) with a subset of \( \text{Ty} \) called \( \text{Ty}^+ \) and a \( \Pi^+ \) type with domain in \( \text{Ty}^+ \), i.e. \( \Pi^+ : (A : \text{Ty}^+ \Gamma) \to \text{Tm} (\Gamma \to A) \to \text{Tm} \Gamma \) with \( \lambda \text{m} \) : \( \text{Tm} (\Gamma \to A) B = \text{Tm} \Gamma (\Pi^+ A B) \) : \text{app}., natural in \( \Gamma \). \( \text{Ty}^+ \) is classified by the Coquand universe \( U^+ \).
Construction. An element \( A: Ty^* \Gamma \) is an \( A: Ty \Gamma \) together with \( \pi_A: (I: C) \rightarrow I \rightarrow C \) and an isomorphism \( p_A \circ \cdot, q_A \circ [-]_A: C(J, I \rightarrow A \gamma J) \equiv (f: C(J, I)) \times A J (\gamma J[f] J) \rightarrow \rightarrow A \rightarrow - \) natural in \( J \). So \( p_A: C(I \rightarrow A \gamma J, I, I) \) and \( q_A: A(I \rightarrow A \gamma J) (\gamma J[p_A] J) \). Substitution is given by \( I \rightarrow A \gamma J_\delta : I \rightarrow A \gamma J \) and we have \( C(J, I \rightarrow A \gamma J_\delta) = C(J, I \rightarrow A \gamma J) \delta_\delta = (f: C(J, I)) \times A J (\gamma J[f] J) = (f: C(J, I)) \times A J (\delta_\delta[f] J) \). We define \( A^* \) using the \( \pi_A \) operator which comes with \( A \), i.e. \( A^* AB I \gamma J_\delta := B(I \rightarrow A \gamma J_\delta) (\gamma J[p_A] J, r, q_A), b \rightarrow f[I \rightarrow A \gamma B] := b_\gamma (J, I \rightarrow A \gamma J, r, q_A) \)\( = (\gamma J[I][id_{I \rightarrow A \gamma} a] B) \). Like \( U, U^* I \gamma J := Ty^* (y I) \).

**Definition 18** (Naive semantics). Given a category \( C \), \( PSh(C) \) is a model of \( ToS^* \) choosing \( U := U, El := El a, \Pi a B := \Pi (El a) B, U^* := U^*, El^* a^* := c(El^* a^*), \pi^* a^* b := c(\Omega^* (El^* a^*)) (El b) \). Recall that a SOGAT signature \( \Omega \) is an element of \( Ty^* \) in the syntax of \( ToS^* \). A naive model of \( \Omega \) is a category with a terminal object \( \odot \) together with the interpretation of \( \Omega \) in presheaves over this category, i.e. \( (C: Cat^*) \times Tm_{PSh(C)} \rightarrow [\Omega]_{PSh(C)} \).

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 \( Tm: U, \lambda: (Tm \rightarrow Tm) \rightarrow Tm, - \rightarrow - : Tm \rightarrow Tm \rightarrow Tm, \) the second-order formal version is \( \Sigma U^* \lambda Tm.((Tm \rightarrow \rightarrow El^* Tm)) \rightarrow El (El^* Tm) \times (El^* Tm \rightarrow El^* Tm) \rightarrow El (El^* Tm)) \), and we interpret the first-order version of this. We assume a \( C: Cat^* \), write \( D := PSh(C) \), and use \( Tm_{D^*} \times [\Omega]_D \rightarrow [\Omega]_D \circ C \).

\[
\begin{align*}
\left[ \Sigma U^* \left( (\{ q \rightarrow \rightarrow El^* q \} \rightarrow El (El^* q)) \times \{ El^* q \rightarrow \rightarrow El (El^* q)\} \right) \right]_{D^*} \circ C \odot \star = \\
(Tm: Ty^*_D (y \circ) \times Tm_D (y \circ (Tm \rightarrow \rightarrow Tm) (Tm[p])) \times Tm_D (y \circ Tm) (Tm \Rightarrow Tm[p]) = \\
(Tm: (I: C) \rightarrow C(I, o) \rightarrow Set) \times (\{ Tm I \epsilon \rightarrow C(J, I) \rightarrow Tm J \epsilon \} \times \ldots \times \\
(\{ \rightarrow Tm: (I: C) \rightarrow C(I, o) \rightarrow C) \times \ldots \times (\lambda: C(I, o) \times Tm (I \rightarrow Tm) \epsilon \rightarrow Tm I \epsilon \} \times (\ldots \times \\
(\{ app: C(I, o) \times Tm I \epsilon \rightarrow ((I: C) \rightarrow C(J, I) \times Tm J \epsilon \rightarrow Tm J \epsilon \} \times \ldots \times \ldots) \}
\end{align*}
\]

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 uncurred, have several extra \( C(I, o) \) arguments (which can be all filled by \( \epsilon \)), and the type of \( app \) quantifies over another object of \( C \) for each argument. This is the result of using the usual presheaf universe and function space for interpreting \( U \) and \( \Pi \). 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 presheaves 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 19** (Presheaves over a CwF). If \( C \) is a CwF, then \( PSh(C) \) models \( ToS \) without using the usual presheaf \( U \) and \( \Pi \).

Construction. We interpret \( T, \Sigma, Eq \) as in Problem 16, but define \( U, El \) and \( \Pi \) by \( Ty_C, Tm_C \) and \( \rightarrow_C \), respectively: \( U I \gamma I := Ty_C I \), \( El a I \gamma I := Tm_C I (a) \gamma I \), \( \Pi a B I \gamma I := B(I \rightarrow C a) (\gamma I, [p_C J], r, q_C) \) with \( \lambda m b I \gamma I := b_\gamma (I \rightarrow C) \), \( \lambda app I \gamma I := t \gamma I [id_{I \rightarrow C} a] B \).
Problem 20 (Presheaves over a CwF*). If the category C is a CwF*, then the previous model extends to a model of ToS* (Definition 15).

Construction. We interpret U*, el* and π* by TyC, identity and ΠC, respectively: 
U* I γI := TyC I, el* a γI := a γI, π* a b γI := ΠC (a γI) b (γI[pC] r, qC), lam* r γI := lamC r (γI[pC] r, qC), app* t (γI, aI) := appC t (γI, idr . aI |τm).

Definition 21 (Direct semantics). A direct model of a SOGAT signature Ω is a category C with a terminal object together with the interpretation of Ω in presheaves over presheaves over C, evaluated at the terminal presheaf: (C : Cat*) × PSh(PSh(C)) * PSh(C) *. Note that this makes sense because PSh(C) : CwF*, hence PSh(PSh(C)) is a model of ToS*.

We revisit the example from the end of the previous section. We again assume a C : Cat* and write D := PSh(C) and E := PSh(D).

\[ \sum U^* \left( \left( (q \mapsto el^* q) \times (el^* q \mapsto el^* q) \right) \right) \circ D * = \]

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 1 components and app is uncurried.

Theorem 22. For any signature, the naive and direct semantics result in isomorphic notions of models.

Proof. We fix a C : Cat*, and denote D := PSh(C) and E := PSh(D). D is a model of ToS* via Definition 18 and E is a model via Definition 21, and Yoneda navigates between them (it is not only a functor, but a CwF pseudomorphism [37]). By induction on the syntax of ToS*, we define α for contexts, substitutions, types and terms: αT := SubE \[ T \] \[ E \], αA := A \[ E \] \[ E \], αA := A \[ id, a \] \[ E \] = y \[ a \] \[ D \] \[ a \] \[ E \]. For a signature Ω : Ty, we thus obtain (Ω) \[ E \] \[ D \] * \[ E \] \[ D \] = y (Ω) \[ D \] \[ a \] \[ E \] = TmD \[ E \] \[ D \] \[ E \].

Note that there is no size issue when stating the isomorphism because even if E is one level up compared to D, we only use small components from E when evaluating into it.

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 Ty : Set, Tm : Ty → Set, ..., refl : (u = v) ≅ Tm (Eq A u v) : reflect of Definition 12 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 23** (Curried \( \Pi \)). By induction-recursion, we define the \( \Sigma \)-closure of \( U \).

\[
\begin{align*}
U^* &: \text{Set} & \text{El}^* &: U^* \to Ty \\
\Gamma^* &: U^* & \text{El}^* \Gamma^* &=: \Gamma \\
\Sigma^* &: (as : U^*) \to (\text{Tm} (\text{El}^* as) \to \text{Tm} U) \to U^* & \text{El}^* (\Sigma^* as b) &=: \Sigma (\text{El}^* as) \lambda x. \text{El} (b x)
\end{align*}
\]

By induction on \( U^* \), we define the carried function space with \( U^* \) domain.

\[
\begin{align*}
\Pi^* &: (as : U^*) \to (\text{Tm} (\text{El}^* as) \to Ty) & \to Ty \\
\Pi^* \Gamma^* &\; B := B tt \\
\Pi^* (\Sigma^* as c) B &:= \Pi^* as (\lambda xs. \Pi (c xs) \lambda y. B (xs, y))
\end{align*}
\]

\( \Pi^* \) comes with \( \text{lam}^* \), \( \cdot^* \), and \( \beta, \eta \) laws all defined by induction on \( U^* \) providing the following isomorphism.

\[
\text{lam}^* : ((xs : \text{Tm} (\text{El}^* as)) \to \text{Tm} (B xs)) \cong \text{Tm} (\Pi^* as B) =: \cdot^* -
\]

We define the signature for categories with a terminal object by \( \text{Cat}^* : Ty := \Sigma \ U \lambda Ob. \Sigma (Ob \Rightarrow Ob \Rightarrow U) \lambda H o m \ldots \). We assume a \( C : \text{Tm Cat}^* \), we refer to its components by \( Ob, \text{Hom} \ldots \).

**Problem 24** (A CwF\(^+\) \( \mathcal{D} \) of presheaves over \( C \)). 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 \( C \).

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

\[
\begin{align*}
\Pi \text{Ob} \lambda I. \Pi (\Gamma I) \lambda y. A : I \cdot \gamma I &\Rightarrow \Pi \text{Ob} \lambda J. \Pi (\text{Hom} J I) \lambda f. E (A : J \cdot \gamma (\gamma_1 [f] I)) \\
\end{align*}
\]

We define \( \text{Ty}_\mathcal{D} (\Gamma : \text{Con}_\mathcal{D}) := \text{Tm} (\text{Ty}_\mathcal{D} \Gamma) \to Ty \) as \( \text{Tm}_\mathcal{D} \Delta \Gamma := \Sigma (\Pi \text{Ob} \lambda I. \Pi^* (\Gamma I) \lambda y. E (A : I \cdot \gamma I)) \) . Context extension \( \cdot^*_\mathcal{D} \) is \( \Sigma^* \), \( \text{Ty}_\mathcal{D}^* \) is the same as \( \text{Ty}_\mathcal{D} \) extended with an \( \cdot^*_A \) operator in \( \Pi \text{Ob} \lambda I. \Gamma I 
\Rightarrow^* E (\text{El} Ob) \) and its universal property. We define the first component of \( \Pi^*_\mathcal{D} := (\Lambda : \text{Tm} (\text{Ty}_\mathcal{D} \Gamma)) \to \text{Tm} (\text{Ty}_\mathcal{D} (I \cdot^*_\mathcal{D} A)) \to \text{Tm} (\text{Ty}_\mathcal{D} \Gamma) \) by \( \pi^*_\mathcal{D} (A : B : I : \cdot^*_A \gamma I := B \cdot (\cdot^*_A I \cdot^*_A \gamma I) : \cdot^*_A (\gamma_1 [p_A] I, q_A) \) where \( \cdot^*_A, p_A \) and \( q_A \) are components in the input A. Note the careful distinguishing of metatheoretic function application, \( \cdot \) and \( \cdot^*_A \)s. The full details are given as Supplementary Material.

**Problem 25** (\( \mathcal{E} := \text{PSh}(\mathcal{D}) \)). The Ty-valued presheaves over \( \mathcal{D} \) are a first-order model of ToS\(^+\). We name this model \( \mathcal{E} \).

**Proof.** \( \text{Con}_\mathcal{E} \) is defined as \( (\Psi : \text{Con}_\mathcal{D} \to Ty) \times (\cdot)_\Psi : \text{Tm} (\Psi \Gamma) \to \text{Sub}_\mathcal{D} \Delta \Gamma \to \text{Tm} (\text{Ty} \Delta) \times (\text{functoriality}) \). Types are Ty-valued dependent presheaves, terms are sections, context extension \( \cdot^*_\mathcal{E} \) and \( \Sigma_\mathcal{E} \) are given by \( \Sigma \ U_\mathcal{E}, E_\mathcal{E}, \Pi_\mathcal{E} \) are given by \( \text{Ty}_\mathcal{D}, \text{Tm}_\mathcal{D}, \cdot^*_\mathcal{D} \), respectively. \( \Sigma^*_\mathcal{E} \) is pointwise \( \Sigma^*_\mathcal{E} \), its restriction operation and \( \text{reflect}_\mathcal{E} \) use reflect. \( U^*_\mathcal{E}, E^*_\mathcal{E}, \Pi^*_\mathcal{E} \) are defined by \( \text{Ty}^*_\mathcal{D}, \text{id}^*_\mathcal{E}, \) and \( \cdot^*_\mathcal{D} \), respectively.
Given an \( \Omega : \text{Ty} \circ \) in the first-order syntax of ToS\(^\ast \), its GAT translation is \( \Sigma \text{Cat}^\ast \lambda C. \left[ \Omega \right]_{E(C)} \circ \left[ D(C) \right] \) tt where we explicitly marked that \( D \) and \( E \) depend on \( C \).

Now we can reuse the semantics of GATs [43, 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 [45].

Our running example assuming \( C : \text{Tm}^\ast \) (its first two components named \( \text{Ob}, \text{Hom} \)):

\[
\begin{align*}
\sum U^+ \left( \begin{array}{c}
\frac{\left( \begin{array}{c}
\left( q \Rightarrow^+ \text{el}^+ q \Rightarrow \text{El}(\text{el}^+ q) \times (\text{el}^+ q \Rightarrow \text{el}^+ q \Rightarrow \text{El}(\text{el}^+ q))
\end{array} \right) \right)}{D} \equiv \text{tt} \\
\sum (\text{Ty}^+ \circ \gamma) \lambda \text{Tm}, \text{Tm}^\ast \left( \begin{array}{c}
\left( \text{Ob} \Rightarrow \text{U} \right) \lambda \text{Tm}, C(\Pi \text{Ob} \lambda I. \text{Tm} \cdot I \Rightarrow \Pi \text{Ob} \lambda J. \text{Hom} \cdot J \cdot I \Rightarrow \text{El}(\text{Tm} \cdot J) \right)
\end{array} \right)
\end{align*}
\]

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

We implemented the SOGAT \( \rightarrow \) GAT translation in Agda using partial deep embeddings of ToS and ToS\(^\ast \). It computes the expected GAT signatures for a number of SOGAT examples. It is available as Supplementary Material.

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 \( \text{Con} = \text{Set}, \text{Ty} \lambda I = \lambda I \rightarrow \text{Set}, \text{Tm} \lambda I A = (\lambda I : I) \rightarrow A \lambda I \) 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 27. For any SOGAT signature, the direct semantics and the GAT semantics over the 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 \( \text{Ty} \) are Russell-universes and are closed under type formers \( \Sigma, \Pi, \text{Eq} \) without the restrictions we have in the syntax of ToS. We reformulate Definition 21 in this internal language: the category \( C \) becomes an element of \( \text{Tm} \Sigma^\ast \), the \( \mathcal{D}' := \text{PSh}(C) \) is a \( \text{CwF}^\ast \) with \( \text{Ty} \)-valued types and terms. We compare this \( \mathcal{D}' \) and the \( \mathcal{D} \) given by Problem 24: we define \( \alpha : \mathcal{D} \rightarrow \mathcal{D}' \) as a strict \( \text{CwF}^\ast \)-morphism which is bijective on \( \text{Ty}, \text{Ty}^\ast \) and \( \text{Tm} \).

The content of \( \alpha \) is mapping in and out of the inductive-recursive universe \( U^\ast \). We denote \( E := \text{PSh}(D) \) and \( E' := \text{PSh}(D') \). Precomposition with \( \alpha \) is \( \alpha^* : \text{PSh}(D') \rightarrow \text{PSh}(D) \) which is a strict \( \text{CwF} \)-morphism. Now, by induction on the syntax of ToS\(^\ast \), we define \( \beta \) for contexts, substitutions, types and terms:

\[
\begin{align*}
\beta_I : \text{Sub}_{E^\ast} \left( a^* \right) \left( I \right)_{E'}, \beta_{\gamma} : \beta_I \circ \left( \gamma \right)_{E'} = a^* \left( \gamma \right)_{E'} \circ \beta_D, \\
\beta_A : \left( A \right)_{E'} \cong a^* \left( A \right)_{E'} \circ \beta_I, \beta_a : \beta_A(\text{id}, a)_{E'} = a^* \left( a \right)_{E'} \circ \beta_I.
\end{align*}
\]

Now for a signature \( \Omega : \text{Ty} \circ \), from \( \beta_\Omega \) we have \( \left[ \Omega \right]_{E^\ast} \circ \Omega = a^* \left[ \Omega \right]_{E'} \circ \beta_D \circ \Omega = \left[ \Omega \right]_{E'} \circ \Omega^\ast. \)

### Corollary 28. By combining the isomorphisms of Theorems 22 and 27: for any SOGAT signature, in presheaves over any of its first-order models, a second-order model is available.

This corollary formalises the diagonal internalisation arrow \( S \hookrightarrow S \) in [16, page 3].
7 Extensions and variants

In this section, we sketch some extensions of ToS\(^+\) and the alternative single substitution calculus semantics, see the Supplementary Material for details.

Open and infinitary SOGATs. The SOGAT→GAT translation also works in the case when signatures are open (can refer to external types like \(\mathbb{N}\) in Definition 10) or infinitary. 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 \(\text{Set}^\circ\) of inner types [43, Chapter 3]. The theory of possibly open signatures includes a type former \(\Pi : (A : \text{Set}) \rightarrow (A \rightarrow Ty) \rightarrow Ty\), with the universal property \((a : A) \rightarrow \text{Tm}(B a)) \cong \text{Tm}(\Pi A B)\). For example, Definition 10 is formalised as \(\Sigma (\mathbb{N} \Rightarrow U)\lambda Ty.\Sigma (\hat{\Pi} N \lambda T y : i \Rightarrow U)\lambda T m \ldots\) where \(\mathbb{N} : \text{Set}^\circ\). Similarly, for infinitary signatures, we have a type former \(\hat{\pi} : (A : \text{Set}) \rightarrow (A \rightarrow \text{Tm} U) \rightarrow \text{Tm} U\) with the universal property \((a : A) \rightarrow \text{Tm}(\text{El}(b a)) \cong \text{Tm}(\text{El}(\hat{\pi} A b))\). 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 [43, Chapter 5].

Semantics using single substitution calculus. 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 SSC translation of a GAT. We only change the construction for Problem 24: \(\pi\) is not a category, just a graph with a vertex \(\circ\); \(\text{Con}_{\Gamma\Theta}\) and \(\text{Ty}_{\Gamma\Theta}\) do not include functoriality equations; \(A : \text{Ty}_{\Theta}^\circ\) includes \(\pi A\), but not the usual universal property; instead we have \(\pi A, \pi A, \langle\cdots\rangle A, -^{\pi A}\) 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.

---

1 We can restrict any CwF to be only an SSC: we build a new sort of substitutions out of a single term or a single weakening inductively. These substitutions do not compose, so they do not form a category, but they form a model of the above described SSC.
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 \cite{46}. 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 \cite{16}), and so on.

References


10:20  Second-Order Generalised Algebraic Theories: Signatures and First-Order Semantics


41 Ambrus Kaposi and Szumi Xie. Type theory in type theory using single substitutions. In *30th International Conference on Types for Proofs and Programs*. IT University of Copenhagen, 2024.


More examples of languages as SOGATs

Definition 29 (Hindley–Milner type system).

\[
\begin{align*}
\text{MTy} & : \text{Set} & \rightarrow & : \text{MTy} \rightarrow \text{MTy} \rightarrow \text{MTy} \\
\text{Ty} & : \text{Set} & \text{lam} & : (\text{Tm}(iA) \rightarrow \text{Tm}(iB)) \equiv \text{Tm}(i(A \Rightarrow B)) : \cdot \cdot \\
i & : \text{MTy} \rightarrow \text{Ty} & \forall & : (\text{MTy} \rightarrow \text{Ty}) \rightarrow \text{Ty} \\
\text{Tm} & : \text{Ty} \rightarrow \text{Set} & \text{Lam} & : ((A : \text{MTy}) \rightarrow \text{Tm}(B A)) \equiv \text{Tm}(\forall B) : \cdot \cdot
\end{align*}
\]

The language of fine-grain call by value is to Freyd categories [44] 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 30 (Fine-grain call by value).

\[
\begin{align*}
&\text{Ty} : \text{Set} & &\Rightarrow - : \text{Ty} \to \text{Ty} \\
&\text{Val} : \text{Ty} \to \text{Set} & &\text{lam} : (\text{Val} \ A \to \text{Tm} \ B) \to \text{Val} \ (A \Rightarrow B) \\
&\text{Tm} : \text{Ty} \to \text{Set} & &\text{val} : (\text{Val} \ (A \Rightarrow B) \to \text{Val} \ A \to \text{Tm} \ B) \\
&\text{return} : \text{Val} \ A \to \text{Tm} \ A & &\Rightarrow \beta : \text{lam} \ f \cdot a = f \ a \\
&\text{idl} : \text{return} \ a \Rightarrow f = f \ a & &\text{idr} : m \Rightarrow \ \text{return} = m \\
&\text{ass} : (m \Rightarrow f) \Rightarrow g = & &\text{ass} : (m \Rightarrow (\lambda\ a. f \ a \Rightarrow g)) \\
&\text{T} : \text{Tm} \ A \Rightarrow \text{Val} \ (T \ A) & &\text{fix} : (\text{Val} \ (T \ A) \Rightarrow \text{Tm} \ A) \\
&\text{thunk} : \text{Tm} \ A \Rightarrow \text{Val} \ (T \ A) : \text{force} & &\text{fix} \beta : \text{fix} \ f = f \ (\text{thunk} \ (\text{fix} \ f))
\end{align*}
\]

The following definition shows that all the languages in the lambda cube [12] can be given as SOGATs. The simply typed lambda calculus (STLC) only includes \( \Pi_{\text{fr}} \), and the edges in each dimension add one of the other three \( \Pi \) types, respectively. The calculus of constructions (CC) includes all four \( \Pi \) types.

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

Definition 31 (CC).

\[
\begin{align*}
&\text{Box} : \text{Set} \\
&\text{Ty} : \Box \Rightarrow \text{Set} & &\Rightarrow - : \text{Ty} \Rightarrow \text{Ty} \\
&\ast : \Box & &\Box \\
&\text{Tm} : \text{Ty} \Rightarrow \text{Set} & &\Rightarrow - : \text{Tm} \Rightarrow \text{Tm} \\
&\Pi_{\text{fr}} : (A : \text{Ty} \Rightarrow) \Rightarrow (\text{Tm} \ A \Rightarrow \text{Ty} \Rightarrow) \Rightarrow \text{Ty} \Rightarrow & &\Rightarrow (a : \text{Tm} \ A) \Rightarrow \text{Tm} \ (B \ a) \\
&\Pi_{\Box} : (A : \text{Ty} \Rightarrow) \Rightarrow (\text{Tm} \ A \Rightarrow \Box) \Rightarrow \Box & &\Rightarrow (a : \text{Tm} \ A) \Rightarrow \text{Tm} \ (L \ a) \\
&\Pi_{\ast} : (K : \Box) \Rightarrow (\text{Tm} \ K \Rightarrow \text{Ty} \Rightarrow) \Rightarrow \text{Ty} \Rightarrow & &\Rightarrow (a : \text{Tm} \ K) \Rightarrow \text{Tm} \ (B \ A) \\
&\Pi_{\Box} : (K : \Box) \Rightarrow (\text{Tm} \ K \Rightarrow \Box) \Rightarrow \Box & &\Rightarrow (a : \text{Tm} \ K) \Rightarrow \text{Tm} \ (L \ A)
\end{align*}
\]

The next definition adds \( \Sigma \), 0, 1, 2 and W-types to minimal Martin-Löf type theory, which is enough to encode all inductive types [33].

Definition 32 (Minimal Martin-Löf type theory with inductive types). We extend Definition 10 with the following.
\begin{align*}
\Sigma & : (A : Ty \rightarrow (Tm A \rightarrow Ty) \rightarrow Ty) \\
(\sim, \sim) & : (a : Tm A) \times Tm (B a) \equiv Tm (\Sigma A B) : \text{fst, snd} \\
\bot & : Ty 0 \\
\text{exfalso} & : Tm \bot \rightarrow Tm A \\
\top & : Ty 0 \\
\text{Bool} & : Ty 0 \\
\text{true} & : Tm \text{Bool} \\
\text{false} & : Tm \text{Bool} \\
\text{indBool} & : (C : Tm \text{Bool} \rightarrow Ty) \rightarrow Tm (C \text{true}) \rightarrow Tm (C \text{false}) \rightarrow Tm (C b) \\
\text{Bool}^\beta_1 & : \text{indBool} t f \text{true} = t \\
\text{Bool}^\beta_2 & : \text{indBool} t f \text{false} = f \\
\text{Id} & : (A : Ty) \rightarrow Tm A \rightarrow Tm A \rightarrow Ty \\
\text{refl} & : (a : Tm A) \rightarrow Tm (\text{Id} a a) \\
\text{J} & : (C : (x : Tm A) \rightarrow Tm (\text{Id} A a x) \rightarrow Ty) \rightarrow \\
& \quad Tm (C a (\text{refl} a)) \rightarrow (x : Tm A)(e : Tm (\text{Id} A a x)) \rightarrow Tm (C x e) \\
\text{Id}^\beta & : C w a (\text{refl} a) = w \\
\text{W} & : (S : Ty) \rightarrow (Tm S \rightarrow Ty) \rightarrow Ty \\
\text{sup} & : (s : Tm S) \rightarrow (Tm (P s) \rightarrow Tm (W S P)) \rightarrow Tm (W S P) \\
\text{indW} & : (C : Tm (W S P) \rightarrow Ty) \rightarrow \\
& \quad \left(\left(\left(p : Tm (P s) \rightarrow Tm (C (f p))\right) \rightarrow Tm (C (\text{sup} s f))\right)\right) \rightarrow \\
& \quad (w : Tm (W S P)) \rightarrow Tm (C w) \\
\text{W}^\beta & : \text{indW} C h (\text{sup} s f) = h (A p. \text{indW} C h (f p))
\end{align*}

In the following example, we add a new sort of telescopes to type theory. This can also be seen as an inductive-recursive definition internally to presheaves over the syntax (or any model; it shows that any CwF with \(T\) and \(\Sigma\) can be extended with telescopes).

➤ **Definition 33** (Telescopes in Martin-Löf type theory). *We extend Definition 32 with the following.*

\[
\begin{array}{ll}
\text{Tys} & : \text{Set} \\
\circ & : \text{Tys} \\
\rightarrow & : (\overline{\text{A}} : \text{Tys}) \rightarrow (\text{Tms} \overline{\text{A}} \rightarrow Ty) \rightarrow \text{Tys} \\
\vdash \neg & : \text{Tys} \rightarrow Ty \\
\circ \beta & : \neg\neg = \top \\
\star \beta & : (\overline{\text{A}} \rightarrow \text{Tys}) \rightarrow (\text{Tms} \overline{\text{A}} \rightarrow Ty) \\
\end{array}
\]

The next SOGAT has both telescopes and teleopic terms, it does not rely on the presence of \(\Sigma\) types.

➤ **Definition 34** (Telescopes and telescopic terms in Martin-Löf type theory). *We extend Definition 10 or Definition 32 with the following.*

\[
\begin{array}{ll}
\text{Tys} & : \text{Set} \\
\circ & : \text{Tys} \\
\rightarrow & : (\overline{\text{A}} : \text{Tys}) \rightarrow (\text{Tms} \overline{\text{A}} \rightarrow Ty) \rightarrow \text{Tys} \\
\text{Tms} & : \text{Tys} \rightarrow \text{Set} \\
\star & : \text{Tms} \circ \equiv 1 : \varepsilon \\
\pi_1, \pi_2 & : \text{Tms} (\overline{\text{A}} \rightarrow \text{Tys}) \equiv (\overline{\overline{\text{A}}} : \text{Tys}) \times Tm (\overline{\text{A} \overline{\text{A}}}) \\
\end{array}
\]

We can turn the above two isomorphisms into equalities if \(U\) in the theory of SOGAT signatures was closed under unit and \(\Sigma\), and had sort equations at the same time. Note that these are not featured at the same time in the semantics of GATs [43].