Title TBA

András Kovács

2021 September

Contents

Chapter 1

Introduction

1.1 Specification and Semantics for Inductive Types

1.2 Overview of the Thesis and Contributions

1.3 Notation and Conventions

Chapter 2

Simple Inductive Signatures

In this chapter, we take a look at a very simple notion of inductive signature. The motivation for doing so is to present the basic ideas of this thesis in the easiest possible setting. We also include a complete Agda formalization of the contents of this chapter, in less than 150 lines. Hopefully this provides intuition for the later chapters, which are greatly generalized and expanded compared to the current chapter, and which are not feasible (and probably not that useful) to present in full formal detail.

potentially in intro

The mantra throughout this dissertation is the following: inductive types are specified by typing contexts in certain *theories of signatures*. For each class of inductive types, there is a corresponding theory of signatures, which is viewed as a proper type theory and comes equipped with an algebraic model theory. *Semantics* of signatures is given by interpreting them in certain models of the theory of signatures. Semantics should at least provide a notion of induction principle for each signature, but usually we will provide more than that.

2.1 Theory of Signatures

Generally, more expressive theories of signatures can describe a larger classes of inductive types. As we are aiming at minimalism right now, the current theory of signatures is as follows:

Definition 1. The *theory of signatures*, or ToS for short in the current chapter, is a simple type theory equipped with the following features:

- An empty base type ι.
- A *first-order function type* $\iota \to -$; this is a function whose domain is fixed to be ι. Moreover, first-order functions only have neutral terms: there is application, but no λ-abstraction.

We can specify the full syntax using the following Agda-like inductive definitions.

$$\begin{array}{ll} \mathsf{Ty} & : \mathsf{Set} \\ \iota & : \mathsf{Ty} \\ \iota \to - & : \mathsf{Ty} \to \mathsf{Ty} \end{array} \qquad \begin{array}{ll} \mathsf{Var} & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set} \\ \mathsf{vz} & : \mathsf{Var}\,(\Gamma \triangleright A)\,A \\ \mathsf{vs} & : \mathsf{Var}\,\Gamma\,A \to \mathsf{Var}\,(\Gamma \triangleright B)\,A \end{array}$$

$$\begin{array}{ll} \mathsf{Con} & : \mathsf{Set} \\ \bullet & : \mathsf{Con} \\ - \triangleright - & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Con} \end{array} \qquad \begin{array}{ll} \mathsf{Tm} & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set} \\ \mathsf{var} & : \mathsf{Var}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \\ \mathsf{app} & : \mathsf{Tm}\,\Gamma\,(\iota \to A) \to \mathsf{Tm}\,\Gamma\,\iota \to \mathsf{Tm}\,\Gamma\,A \end{array}$$

Here, Con contexts are lists of types, and Var specifies well-typed De Bruijn indices, where vz represents the zero index, and vs takes the successor of an index.

Notation 1. We use capital Greek letters starting from Γ to refer to contexts, A, B, C to refer to types, and t, u, v to refer to terms. In examples, we may use a nameful notation instead of De Bruijn indices. For example, we may write $x : \mathsf{Tm}\,(\bullet \triangleright (x : \iota) \triangleright (y : \iota))\,\iota$ instead of $\mathsf{var}\,(\mathsf{vs}\,\mathsf{vz}) : \mathsf{Tm}\,(\bullet \triangleright \iota \triangleright \iota)\,\iota$. Additionally, we may write $t\,u$ instead of $\mathsf{app}\,t\,u$ for t and u terms.

Definition 2. *Parallel substitutions* map variables to terms.

$$\begin{aligned} &\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\ &\mathsf{Sub}\,\Gamma\,\Delta \equiv \{A\} \to \mathsf{Var}\,\Delta\,A \to \mathsf{Tm}\,\Gamma\,A \end{aligned}$$

We use σ and δ to refer to substitutions. We also define the action of substitution on terms, by recursion on terms:

$$\begin{aligned} &-[-] : \mathsf{Tm}\,\Delta\,A \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Tm}\,\Gamma\,A \\ &(\mathsf{var}\,x)\quad [\sigma] \equiv \sigma\,x \\ &(\mathsf{app}\,t\,u)[\sigma] \equiv \mathsf{app}\,(t[\sigma])\,(u[\sigma]) \end{aligned}$$

The *identity substitution* is defined simply as $\mathsf{id} \equiv \mathsf{var}$. It is easy to see that $t[\mathsf{id}] = t$ for all t.

Example 1. We may write signatures for natural numbers and binary trees respectively as follows.

$$\mathsf{NatSig} \equiv \bullet \triangleright (\mathit{zero} : \iota) \triangleright (\mathit{suc} : \iota \to \iota)$$
$$\mathsf{TreeSig} \equiv \bullet \triangleright (\mathit{leaf} : \iota) \triangleright (\mathit{node} : \iota \to \iota \to \iota)$$

In short, the current ToS allows inductive types which are

- *Single-sorted*: this means that we have a single type constructior, corresponding to ι.
- *Closed*: signatures cannot refer to any externally existing type. For example, we cannot write a signature for "lists of natural number" in a direct fashion, since there is no way to refer to the type of natural numbers.
- *Finitary*: inductive types corresponding to signatures are always finitely branching trees. Being closed implies being finitary, since an infinitely branching node would require some external type to index subtrees with. For example, $\mathit{node} : (\mathbb{N} \to \iota) \to \iota$ would specify an infinite branching (if such type was allowed in ToS).

Remark. We omit λ-expressions from ToS for the sake of simplicity: this causes terms to be always in normal form (neutral, to be precise), and thus we can skip talking about conversion rules. Later, starting from Chapter 4 we include proper $\beta\eta$-rules in signature theories.

2.2 Semantics

For each signature, we need to know what it means for a type theory to support the corresponding inductive type. For this, we need at least a notion of *algebras*, which can be viewed as a bundle of all type and value constructors, and what it means for an algebra to support an *induction principle*. Additionally, we may want to know what it means to support a *recursion principle*, which can be viewed as a non-dependent variant of induction. In the following, we define these notions by induction on ToS syntax.

2.2.1 Algebras

First, we calculate types of algebras. This is simply a standard interpretation into the Set universe. We define the following operations by induction; the $-^A$ name is overloaded for Con, Ty and Tm.

$$
\begin{aligned}
&-^A : \mathsf{Ty} \to \mathsf{Set} \to \mathsf{Set}\\
&\iota^A \qquad\quad X \equiv X\\
&(\iota \to A)^A\, X \equiv X \to A^A\, X
\end{aligned}
$$

$$
\begin{aligned}
&-^A : \mathsf{Con} \to \mathsf{Set} \to \mathsf{Set}\\
&\Gamma^A\, X \equiv \{A : \mathsf{Ty}\} \to \mathsf{Var}\,\Gamma\, A \to A^A\, X
\end{aligned}
$$

$$
\begin{aligned}
&-^A : \mathsf{Tm}\,\Gamma\, A \to \{X : \mathsf{Set}\} \to \Gamma^A\, X \to A^A\, X\\
&(\mathsf{var}\, x)^A \quad \gamma \equiv \gamma\, x\\
&(\mathsf{app}\, t\, u)^A\, \gamma \equiv t^A\, \gamma\, (u^A\, \gamma)
\end{aligned}
$$

$$
\begin{aligned}
&-^A : \mathsf{Sub}\,\Gamma\,\Delta \to \{X : \mathsf{Set}\} \to \Gamma^A\, X \to \Delta^A\, X\\
&\sigma^A\, \gamma\, x \equiv (\sigma\, x)^A\, \gamma
\end{aligned}
$$

Here, types and contexts depend on some $X : \mathsf{Set}$, which serves as the interpretation of ι. We define Γ^A as a product: for each variable in the context, we get a semantic type. This trick, along with the definition of Sub, makes formalization a bit more compact. Terms and substitutions are interpreted as natural maps. Substitutions are interpreted by pointwise interpreting the contained terms.

Notation 2. We may write Γ^A using notation for Σ-types. For example, we may write $(\mathit{zero} : X) \times (\mathit{suc} : X \to X)$ for the result of computing $\mathsf{NatSig}^A\, X$.

Definition 3. We define *algebras* as follows.

$$
\begin{aligned}
&\mathsf{Alg} : \mathsf{Con} \to \mathsf{Set}_1\\
&\mathsf{Alg}\,\Gamma \equiv (X : \mathsf{Set}) \times \Gamma^A\, X
\end{aligned}
$$

Example 2. $\mathsf{Alg}\,\mathsf{NatSig}$ is computed to $(X : \mathsf{Set}) \times (\mathit{zero} : X) \times (\mathit{suc} : X \to X)$.

2.2.2 Morphisms

Now, we compute notions of morphisms of algebras. In this case, morphisms are functions between underlying sets which preserve all specified structure. The interpretation for calculating morphisms is a *proof-relevant logical relation interpretation* [**?**] over the $-^A$ interpretation. The key part is the interpretation of types:

$$
\begin{aligned}
&-^M : (A : \mathsf{Ty})\{X_0\, X_1 : \mathsf{Set}\}(X^M : X_0 \to X_1) \to A^A\, X_0 \to A^A\, X_1 \to \mathsf{Set}\\
&\iota^M \qquad\quad X^M\, \alpha_0\, \alpha_1 \equiv X^M\, \alpha_0 = \alpha_1\\
&(\iota \to A)^M\, X^M\, \alpha_0\, \alpha_1 \equiv (x : X_0) \to A^M\, X^M\, (\alpha_0\, x)\, (\alpha_1\, (X^M\, x))
\end{aligned}
$$

We again assume an interpretation for the base type ι, as X_0, X_1 and $X^M : X_0 \to X_1$. X^M is function between underlying sets of algebras, and A^M computes what it means that X^M preserves an operation with type A. At the base type, preservation is simply equality. At the first-order function type, preservation is a quantified statement over X_0. We define morphisms for Con pointwise:

$$
\begin{aligned}
&\mathsf{Con}^M : (\Gamma : \mathsf{Con})\{X_0\, X_1 : \mathsf{Set}\} \to (X_0 \to X_1) \to \Gamma^A\, X_0 \to \Gamma^A\, X_1 \to \mathsf{Set}\\
&\Gamma^M\, X^M\, \gamma_0\, \gamma_1 \equiv \{A : \mathsf{Ty}\}(x : \mathsf{Var}\, \Gamma\, A) \to A^M\, X^M\, (\gamma_0\, x)\, (\gamma_1\, x)
\end{aligned}
$$

For terms and substitutions, we get preservation statements, which are sometimes called *fundamental lemmas* in discussions of logical relations [**?**].

$$
\begin{aligned}
&-^M : (t : \mathsf{Tm}\, \Gamma\, A) \to \Gamma^M\, X^M\, \gamma_0\, \gamma_1 \to A^M\, X^M\, (t^A\, \gamma_0)\, (t^A\, \gamma_1)\\
&(\mathsf{var}\, x)^M \quad \gamma^M \equiv \gamma^M\, x\\
&(\mathsf{app}\, t\, u)^M \gamma^M \equiv t^M\, \gamma^M\, (u^A\, \gamma_0)
\end{aligned}
$$

$$
\begin{aligned}
&-^M : (\sigma : \mathsf{Sub}\, \Gamma\, \Delta) \to \Gamma^M\, X^M\, \gamma_0\, \gamma_1 \to \Delta^M\, X^M\, (\sigma^A\, \gamma_0)\, (\sigma^A\, \gamma_1)\\
&\sigma^M\, \gamma^M\, x = (\sigma\, x)^M\, \gamma^M
\end{aligned}
$$

The definition of $(\mathsf{app}\, t\, u)^M$ is well-typed by the induction hypothesis $u^M\, \gamma^M : X^M\, (u^A\, \gamma_0) = u^A\, \gamma_1$.

Definition 4. We again pack up Γ^M with the interpretation of ι, to get notions of *algebra morphisms*:

$$
\begin{aligned}
&\mathsf{Mor} : (\Gamma : \mathsf{Con}) \to \mathsf{Alg}\, \Gamma \to \mathsf{Alg}\, \Gamma \to \mathsf{Set}\\
&\mathsf{Mor}\, \Gamma\, (X_0,\, \gamma_0)\, (X_1,\, \gamma_1) \equiv (X^M : X_0 \to X_1) \times \Gamma^M\, X^M\, \gamma_0\, \gamma_1
\end{aligned}
$$

Example 3. We have the following computation:

$$\begin{aligned}&\mathsf{Mor}\,\mathsf{NatSig}\,(X_0,\ \mathit{zero}_0,\ \mathit{suc}_0)\,(X_0,\ \mathit{zero}_1,\ \mathit{suc}_1) \equiv\\ &\qquad (X^M : X_0 \to X_1)\\ &\times (X^M\ \mathit{zero}_0 = \mathit{zero}_1)\\ &\times ((x : X_0) \to X^M\,(\mathit{suc}_0\ x) = \mathit{suc}_1\,(X^M\ x))\end{aligned}$$

Definition 5. We state *initiality* as a predicate on algebras:

$$\begin{aligned}&\mathsf{Initial} : \mathsf{Alg}\,\Gamma \to \mathsf{Set}\\ &\mathsf{Initial}\,\gamma \equiv (\gamma' : \mathsf{Alg}\,\Gamma) \to \mathsf{isContr}\,(\mathsf{Mor}\,\Gamma\,\gamma\,\gamma')\end{aligned}$$

Here isContr refers to unique existence. If we drop isContr from the definition, we get the notion of weak initiality, which corresponds to the recursion principle for Γ. Although we call this predicate Initial, in this chapter we do not yet show that algebras form a category. We provide the extended semantics in Chapter 4, and the currently computed algebras and morphisms remain the same there.

Morphisms vs. logical relations. The $-^M$ interpretation can be viewed as a special case of logical relations over the $-^A$ model: every morphism is a *functional* logical relation, where the chosen relation between the underlying sets happens to be a function. Consider now a more general relational interpretation for types:

$$\begin{aligned}&-^R : (A : \mathsf{Ty})\{X_0\,X_1 : \mathsf{Set}\}(X^R : X_0 \to X_1 \to \mathsf{Set}) \to A^A\,X_0 \to A^A\,X_1 \to \mathsf{Set}\\ &\iota^R \qquad\quad X^R\,\alpha_0\ \alpha_1 \equiv X^R\,\alpha_0\,\alpha_1\\ &(\iota \to A)^R\,X^R\,\alpha_0\ \alpha_1 \equiv (x_0 : X_0)(x_1 : X_1) \to X^R\,x_0\,x_1 \to A^R\,X^R\,(\alpha_0\,x_0)\,(\alpha_1\,x_1)\end{aligned}$$

Here, functions are related if they map related inputs to related outputs. If we know that $X^M\,\alpha_0\,\alpha_1 \equiv (f\,\alpha_0 = \alpha_1)$ for some f function, we get

$$(x_0 : X_0)(x_1 : X_1) \to f\,x_0 = x_1 \to A^R\,X^R\,(\alpha_0\,x_0)\,(\alpha_1\,x_1)$$

Now, we can simply substitute along the input equality proof in the above type, to get the previous definition for $(\iota \to A)^M$:

$$(x_0 : X_0) \to A^R\,X^R\,(\alpha_0\,x_0)\,(\alpha_1\,(f\,x_0))$$

This substitution along the equation is called "singleton contraction" in the jargon of homotopy type theory [**?**]. The ability to perform contraction here is at the

heart of the *strict positivity restriction* for inductive signatures. Strict positivity in our setting corresponds to only having first-order function types in signatures. If we allowed function domains to be arbitrary types, in the definition of $(A \to B)^M$ we would only have a black-box $A^M\, X^M : A^A\, X_0 \to A^A\, X_1 \to \mathsf{Set}$ relation, which is not known to be given as an equality.

In Chapter 4 we expand on this. As a preliminary summary: although higher-order functions have relational interpretation, such relations do not generally compose. What we eventually aim to have is a *category* of algebras and algebra morphisms, where morphisms properly compose. We need a *directed* model of the theory of signatures, where every signature becomes a category of algebras. The way to achieve this, is to prohibit higher-order functions, thereby avoiding the polarity issues that prevent a directed interpretation for general function types.

2.2.3 Displayed Algebras

At this point we do not yet have specification for induction principles (or dependent elimination principles). We use the term *displayed algebra* to refer to "dependent" algebras, where every displayed algebra component lies over corresponding components in the base algebra. For the purpose of specifying induction, displayed algebras can be viewed as bundles of induction motives and methods.

Displayed algebras over some $\gamma : \mathsf{Alg}\,\Gamma$ are equivalent to slices over γ in the category of Γ-algebras; we show this in Chapter 4. A slice $f : \mathsf{Sub}\,\Gamma\,\gamma'\,\gamma$ maps elements of γ's underlying set to elements in the base algebra. Why do we need displayed algebras, then? The reasons are twofold. First, we need to compute induction principles exactly, not merely up to isomorphisms, if we are to eventually implement inductive types in a mechanized setting. Second, displayed algebras naturally avoid some strictness issues: they support strictly functorial reindexing, unlike slice objects, whose pullbacks are functorial only up to isomorphism.

For more illustration of using some displayed algebras in a type-theoretic setting, see [**?**]. We adapt the term "displayed algebra" from ibid. as a generalization of displayed categories and functors.

The displayed algebra interpretation is a *logical predicate* interpretation, de-

fined as follows.

$$
\begin{aligned}
&-^D : (A : \mathsf{Ty})\{X\} \to (X \to \mathsf{Set}) \to A^A\, X \to \mathsf{Set}\\
&\iota^D \qquad\quad X^D\, \alpha \equiv X^D\, \alpha\\
&(\iota \to A)^D\, X^D\, \alpha \equiv (x : X)(x^D : X^D\, x) \to A^D\, X^D\, (\alpha\, x)
\end{aligned}
$$

$$
\begin{aligned}
&-^D : (\Gamma : \mathsf{Con})\{X\} \to (X \to \mathsf{Set}) \to \Gamma^A\, X \to \mathsf{Set}\\
&\Gamma^D\, X^D\, \gamma \equiv \{A : \mathsf{Ty}\}(x : \mathsf{Var}\, \Gamma\, A) \to A^D\, X^D\, (\gamma\, x)
\end{aligned}
$$

$$
\begin{aligned}
&-^D : (t : \mathsf{Tm}\, \Gamma\, A) \to \Gamma^D\, X^D\, \gamma \to A^D\, X^D\, (t^A\, \gamma)\\
&(\mathsf{var}\, x)^D \quad \gamma^D \equiv \gamma^D\, x\\
&(\mathsf{app}\, t\, u)^D\, \gamma^D \equiv t^D\, \gamma^D\, (u^A\, \gamma)\, (u^D\, \gamma^D)
\end{aligned}
$$

$$
\begin{aligned}
&-^D : (\sigma : \mathsf{Sub}\, \Gamma\, \Delta) \to \Gamma^D\, X^D\, \gamma \to \Delta^D\, X^D\, (\sigma^A\, \gamma)\\
&\sigma^D\, \gamma^D\, x \equiv (\sigma\, x)^D\, \gamma^D
\end{aligned}
$$

Analogously to before, everything depends on a predicate interpretation $X^D : X \to \mathsf{Set}$ for ι. For types, a predicate holds for a function if the function preserves predicates. The interpretation of terms is again a fundamental lemma, and we have pointwise definitions for contexts and substitutions, as usual.

Example 4. We have the following computation:

2.2.4 Sections

2.3 Term Algebras

2.3.1 Weak Initiality

2.3.2 Dependent Elimination

2.4 Related and Alternative Approaches

Chapter 3

Semantics in Two-Level Type Theory

Introduction: generalizing semantics, distinguishing strict and non-strict equations. Summary

- Formal syntax for TT as cwfs, type formers, universes
- Presheaf models for TT
- 2LTT

3.1 Categories with Families

Describe and motivate cwfs for formal syntax. De bruijn indices, examples of representing stuff. Examples for type formers and universes. Copy from previous papers.

3.2 Presheaf Models of Type Theories

3.3 Two-Level Type Theory

3.3.1 Models

3.3.2 Properties

3.4 Simple Inductive Signatures

3.4.1 Internal Semantics

3.4.2 Strict and Weak Morphisms

3.4.3 Internal Term Algebras

- AMDS
- finite product semantics, computed examples
- Inner term algebras.
- Weak initiality
- Dependent elimination

Chapter 4

Finitary Quotient Inductive-Inductive Types

4.1 Theory of Signatures

4.1.1 Models

4.1.2 Examples

4.2 Semantics

4.2.1 Finite Limit Cwfs

4.2.2 Equivalence of Initiality and Induction

4.2.3 Model of the Theory of Signatures

4.3 Term Algebras

4.3.1 Generic Term Algebras

4.3.2 Induction for Term Algebras

4.3.3 Church Encodings

4.3.4 Awodey-Frey-Speight Encodings

4.4 Left Adjoints of Signature Morphisms

Chapter 5

Infinitary Quotient Inductive-Inductive Types

5.1 Theory of Signatures

5.2 Term Algebras

Chapter 6

Levitation, Bootstrapping and Universe Levels

6.1 Levitation for Closed QIITs

6.2 Levitation for Infinitary QIITs

Chapter 7

Higher Inductive-Inductive Types

7.1 Theory of Signatures

7.2 Semantics

Chapter 8

Reductions

8.1 Finitary Inductive Types

8.2 Finitary Inductive-Inductive Types

8.3 Closed Quotient Inductive-Inductive Types

Chapter 9

Conclusion