Basic Category Theory for Models of Syntax[⋆] (Preliminary Version)

R. L. Crole (R.Crole@mcs.le.ac.uk)

Department of Mathematics and Computer Science, University of Leicester, Leicester, LE1 7RH, U.K.

Abstract. These notes form the basis of four lectures given at the *Summer School on Generic Programming*, Oxford, UK, which took place during August 2002. The aims of the notes are to provide an introduction to very elementary category theory, and to show how such category theory can be used to provide both abstract and concrete mathematical models of syntax. Much of the material is now standard, but some of the ideas which are used in the modeling of syntax involving variable binding are quite new.
It is assumed that readers are familiar with elementary set theory and discrete mathematics, and have met formal accounts of the kinds of syntax as may be defined using the (recursive) datatype declarations that are common in modern functional programming languages. In particular, we assume readers know the basics of λ-calculus.
A pedagogical feature of these notes is that we only introduce the category theory required to present models of syntax, which are illustrated by example rather than through a general theory of syntax.

This Technical Report encorporates a few minor corrections.

⋆ This work was supported by EPSRC grant number GR/M98555.

1 Introduction

1.1 Prerequisites

These notes form the basis of four lectures given at the *Summer School on Generic Programming*, Oxford, UK, which took place during August 2002. The audience consisted mainly of mathematically mature postgraduate computer scientists. As such, we assume that readers already have a reasonable understanding of

- very basic (naive) set theory;
- simple discrete mathematics, such as relations, functions, preordered sets, and equivalence relations;
- simple (naive) logic and the notion of a formal system;
- a programming language, preferably a functional one, and in particular of recursive datatypes; language syntax presented as finite trees;
- inductively defined sets; proof by mathematical and rule induction;
- the basics of λ-calculus, especially free and bound variables.

Some of these topics will be reviewed as we proceed. The appendix defines abstract syntax trees, inductively defined sets, and rule induction. We *do not* assume any knowledge of category theory, introducing all that we need.

1.2 The Aims

Through these notes we aim to teach some of the very basics of category theory, and to apply this material to the study of programming language syntax with binding. We give formal definitions of the category theory we need, and some concrete examples. We also give a few technical results which can be given very abstractly, but for the purposes of these notes are given in a simplified and concrete form. We study syntax through particular examples, by giving categorical models. We do not discuss the general theory of binding syntax. This is a current research topic, but readers who study these notes should be well placed to read (some of) the literature.

1.3 Learning Outcomes

By studying these notes, and completing at least some of the exercises, you should

- know how simple examples of programming language syntax with binding can be specified via (simple) inductively defined formal systems;
- be able to define categories, functors, natural transformations, products and coproducts, presheaves and algebras;
- understand a variety of simple examples from basic category theory;
- know, by example, how to compute initial algebras for polynomial functors over sets;
- understand some of the current issues concerning how we model and implement syntax involving variable binding;
- understand some simple abstract models of syntax based on presheaves;
- know, by example, how to manufacture a categorical formulation of simple formal systems (for syntax);
- be able to prove that the abstract model and categorical formal system are essentially the same;
- know enough basic material to read some of the current research material concerning the implementation and modeling of syntax.

2 Syntax Defined from Datatypes

In this section we assume that readers are already acquainted with formal (programming) languages. In particular, we assume knowledge of syntax trees; occurrences of, free, bound and binding variables in a syntax tree; inductive definitions and proof by (rule) induction; formal systems for specifying judgements about syntax; and elementary facts about recursive datatype declarations. Some of these topics are discussed very briefly in the Appendix, page 47. Mostly, however, we prompt the reader by including definitions, but we do not include extensive background explanations and intuition.

What is the motivation for the work described in these notes? Computer Scientists sometimes want to use existing programming languages, and other tools such as automated theorem provers, to reason about (other) programming languages. We sometimes say that the existing system is our **metalanguage**, (for example Haskell) and the system to be reasoned about is our **object language** (for example, the λ-calculus). Of course, the object language will (hopefully) have a syntax, and we will need to encode, or specify, this object level language within the metalanguage. It turns out that this is not so easy, even for fairly simple object languages, if the object language has binders. This is true of the λ-calculus, and is in fact true of almost any (object level) programming language.

We give four examples, each showing how to specify the syntax, or **terms**, of a tiny fragment of a (object level) programming language. In each case we give a datatype whose elements are called *expressions*. Such expressions denote *abstract syntax trees*—see Appendix. The expressions form a superset of the set of terms we are interested in. We then give *further definitions* which specify a *subset* of the datatype which constitutes exactly the terms of interest. For example, in the case of λ-calculus, the expressions are raw syntax trees, and the terms are such trees quotiented up to α-equivalence. What we would like to be able to do is

Write down a datatype, as one can do in a functional programming language, such that the expressions of the datatype are precisely the terms of the object language we wish to encode.

Then, the "further definitions" alluded to above would not be needed. We would have a very clean encoding of our object level language terms, given by an explicit datatype and no other infrastructure (such as α-equivalence).

2.1 An Example with Distinguished Variables and without Binding

Take **constructor symbols** V, S and A with arities one, one and two respectively. Take also a **set of variables** $\mathbb{V}$ whose *actual* elements will be denoted by v^i where $i \in \mathbb{N}$. The **set of expressions** *Exp* is inductively defined by the grammar[1]

$$Exp ::= \mathsf{V}\ \mathbb{V} \mid \mathsf{S}\ Exp \mid \mathsf{A}\ Exp\ Exp$$

The metavariable e will range over expressions. It should be intuitively clear what we mean by v^i **occurs** in e, which we abbreviate by $v^i \in e$. We omit a formal definition. The set of **(free) variables** of any e is denoted by $fv(e)$.

Later in these notes, we will want to consider expressions e for which

$$fv(e) \subset \{ v^0, \ldots, v^{n-1} \}$$

[1] This formal BNF grammar corresponds to a datatype declaration in Haskell.

$$\frac{0 \le i < n}{\Gamma^n \vdash^{\bar{\mathsf{db}}} v^i} \qquad \frac{\Gamma^n \vdash^{\bar{\mathsf{db}}} e}{\Gamma^n \vdash^{\bar{\mathsf{db}}} \mathsf{S}\ e} \qquad \frac{\Gamma^n \vdash^{\bar{\mathsf{db}}} e \quad \Gamma^n \vdash^{\bar{\mathsf{db}}} e'}{\Gamma^n \vdash^{\bar{\mathsf{db}}} \mathsf{A}\ e\ e'}$$

Fig. 1. Expressions in Environment—Distinguished Variables, No Binding

The idea is that when "constructing" an expression by using the grammar above, we build it out of the initial segment (according to the indices) of variables. It is convenient to give an inductive definition of such expressions. First we inductively a set of judgements $\Gamma^n \vdash^{\bar{\mathsf{db}}} e$ where $n \ge 1$, $\Gamma^n \overset{\text{def}}{=} v^0, \dots, v^{n-1}$ is a list, and of course e is an expression. We refer to Γ^n as an **environment** of variables. The rules appear in Figure 1. $\Gamma^n \vdash^{\bar{\mathsf{db}}} e$ is simply a notation for a binary relationship, written infix style, with relation symbol $\vdash^{\bar{\mathsf{db}}}$. Strictly speaking, we are inductively defining the set (of pairs) $\vdash^{\bar{\mathsf{db}}}$. One can then prove by rule induction that if $\Gamma^n \vdash^{\bar{\mathsf{db}}} e$ then $fv(e) \subset \Gamma^n$. Check the simple details as an *Exercise. Hint: With the notation of the Appendix, we prove by Rule Induction*

$$(\forall\ (\Gamma^n, e) \in \vdash^{\bar{\mathsf{db}}})\ \boxed{(fv(e) \subset \Gamma^n)}$$

See Figure 6, page 50. One has to check property closure for each rule in Figure 1.

2.2 An Example with Distinguished Variables and Binding

We assume that readers are familiar with the syntax of the λ-calculus. One might want to implement (encode) such syntax in a language such as Haskell. If L and E are new constructors then we might consider the grammar (datatype)

$$Exp ::= \mathsf{V}\ \mathbb{V} \mid \mathsf{L}\ \mathbb{V}\ Exp \mid \mathsf{E}\ Exp\ Exp$$

We assume that readers are familiar with **free**, **binding** and **bound** variables, in particular the idea that for a particular occurrence of a variable in a syntax tree defined by the grammar above, the occurrence is either free or bound. In particular recall that $fv(\mathsf{L}\ v^i\ e) = fv(e) \setminus \{v^i\}$; any occurrence of v^i in $\mathsf{L}\ \underline{v^i}\ e$ is bound, and in particular we call the occurrence $\underline{v^i}$ binding. We will want to consider expressions e for which $fv(e) \subset \{v^0, \dots, v^{n-1}\}$, and we also give an inductive definition of such expressions. More precisely we inductively define a set of judgements $\Gamma^n \vdash^{\mathsf{db}} e$ where $n \ge 1$. The rules appear in Figure 2.

$$\frac{0 \leq i < n}{\Gamma^n \vdash^{\mathsf{db}} v^i} \qquad \frac{\Gamma^{n+1} \vdash^{\mathsf{db}} e}{\Gamma^n \vdash^{\mathsf{db}} \mathsf{L}\ v^n\ e} \qquad \frac{\Gamma^n \vdash^{\mathsf{db}} e \quad \Gamma^n \vdash^{\mathsf{db}} e'}{\Gamma^n \vdash^{\mathsf{db}} \mathsf{E}\ e\ e'}$$

Fig. 2. Expressions in Environment—Distinguished Variables and Binding

$$\frac{x \in \Delta}{\Delta \vdash^{\mathsf{ab}} x} \qquad \frac{\Delta, x \vdash^{\mathsf{ab}} e}{\Delta \vdash^{\mathsf{ab}} \mathsf{L}\ x\ e} \qquad \frac{\Delta \vdash^{\mathsf{ab}} e \quad \Delta \vdash^{\mathsf{ab}} e'}{\Delta \vdash^{\mathsf{ab}} \mathsf{E}\ e\ e'}$$

Fig. 3. Expressions in Environment—Arbitrary Variables and Binding

One can then prove by rule induction that if $\Gamma^n \vdash^{\mathsf{db}} e$ then $fv(e) \subset \Gamma^n$. Check the simple details as an *Exercise.* Notice that the rule for introducing abstractions $\mathsf{L}\ v^n\ e$ forces a **distinguished** choice of binding variable. This means that we lose the usual fact that the name of the binding variable does not matter. However, the pay-off of what we call *distinguished binding* is that the expressions inductively defined in Figure 2 correspond exactly to the terms of the λ-calculus, without the need to define α-equivalence. In essence, we are forced to pick a representative of each α-equivalence class.

2.3 An Example with Arbitrary Variables and Binding

Expressions are still defined by

$$Exp ::= \mathsf{V}\ \mathbb{V} \mid \mathsf{L}\ \mathbb{V}\ Exp \mid \mathsf{E}\ Exp\ Exp$$

Now let Δ range over *all non-empty finite lists* of variables *which have distinct elements.* Thus a typical non-empty Δ is $v^1, v^8, v^{100}, v^2 \in [\mathbb{V}]$. We will use typical metavariables x, y, z which range over $\mathbb{V}$. If Δ contains $n \geq 1$ variables, we may write $\Delta = x_0, \ldots, x_{n-1}$. Once again we inductively a set of judgements, this time of the form $\Delta \vdash^{\mathsf{ab}} e$. The rules appear in Figure 3. One can then prove by rule induction that if $\Delta \vdash^{\mathsf{ab}} e$ then $fv(e) \subset \Delta$. Check the simple details as an *Exercise.*

It is convenient to introduce a notion of simultaneous substitution of variables at this point. This will allow us to define the usual notion of α-equivalence of expressions—yielding the *terms* of the λ-calculus. Such substitution will also be used in our mathematical models, later on. Suppose that $0 \leq p \leq \mathsf{len}(\Delta) - 1$. Then $\mathsf{el}_p(\Delta)$ is the pth element of Δ, with position 0 the "first" element. We write ϵ for the empty list. We will define by recursion over expressions e, new expressions $e\{\epsilon/\epsilon\}$ and $e\{\Delta'/\Delta\}$, where $\mathsf{len}(\Delta) = \mathsf{len}(\Delta')$. Informally, $e\{\Delta'/\Delta\}$ is the expression e in which any free occurrence of $\mathsf{el}_p(\Delta)$ in e is replaced by $\mathsf{el}_p(\Delta')$, with bound variables being changed to avoid capture of $\mathsf{el}_p(\Delta')$. For example,

$$(\mathsf{L}\ v^8\ (\mathsf{A}\ v^{10}\ v^2))\{v^3, v^8/v^8, v^2\} = \mathsf{L}\ v^{11}\ (\mathsf{A}\ v^{10}\ v^8)$$

where the binding variable v^8 is changed to v^{11}. We set $e\{\epsilon/\epsilon\} \stackrel{\text{def}}{=} e$ for any e. We also define

$$(\mathsf{V}\ x)\{\Delta'/\Delta\} \stackrel{\text{def}}{=} \begin{cases} x \\ \quad \text{if} \quad (\forall p)(\mathsf{el}_p(\Delta) \neq x) \\ \mathsf{el}_p(\Delta') \\ \quad \text{if} \quad (\exists p)(\mathsf{el}_p(\Delta) = x) \end{cases}$$

$$(\mathsf{L}\ x\ e)\{\Delta'/\Delta\} \stackrel{\text{def}}{=} \begin{cases} \mathsf{L}\ x\ e\{\overline{\Delta'}/\overline{\Delta}\} \\ \quad \text{if} \quad (\forall p)(\mathsf{el}_p(\Delta') \neq x \ \vee \ \mathsf{el}_p(\Delta) \notin fv(e)) \\ \mathsf{L}\ x'\ e\{\overline{\Delta'}, x'/\overline{\Delta}, x\} \\ \quad \text{if} \quad (\exists p)(\mathsf{el}_p(\Delta') = x \wedge \mathsf{el}_p(\Delta) \in fv(e)) \end{cases}$$

$$(\mathsf{E}\ e\ e')\{\Delta'/\Delta\} \stackrel{\text{def}}{=} \mathsf{E}\ e\{\Delta'/\Delta\}\ e'\{\Delta'/\Delta\}$$

where

- $\overline{\Delta}$ is Δ with x deleted (from position p, if it occurs) and, *if x does occur*, $\overline{\Delta'}$ is Δ' with the element in position p deleted, and is otherwise Δ'; and
- x' is the variable v^w where w is 1 plus the maximum of the indices appearing in $\overline{\Delta'}$ and $fv(e)$.

We can inductively define the relation $\sim_\alpha$ of α**-equivalence** on the set of all expressions with the single axiom[2] (schema) $\mathsf{L}\ x\ e \sim_\alpha \mathsf{L}\ x'\ e\{x'/x\}$ where x' $(\neq x)$ is any variable not in $fv(e)$, and rules ensuring that $\sim_\alpha$ is an equivalence relation and a congruence for the constructors. Congruence for L, and

[2] Base rule.

transitivity, are given by the rules (schemas)

$$\frac{e \sim_\alpha e' \qquad e' \sim_\alpha e''}{e \sim_\alpha e''} \qquad\qquad \frac{e \sim_\alpha e'}{\mathsf{L}\ x\ e \sim_\alpha \mathsf{L}\ x\ e'}$$

Write down all of the rules as an *Exercise.*

Note that the terms of the λ-calculus are given by the

$$[e]_\alpha \stackrel{\text{def}}{=} \{e' \mid e' \sim_\alpha e\}$$

where e is any expression. One reason for also defining the judgements $\Delta \vdash^{\mathsf{ab}} e$ is that they will be used in Section 4 to formulate a mathematical model.

2.4 An Example without Variables but with Binding

As mentioned, we assume familiarity with de Bruijn notation, and the notion of *index level.* **Note: You can follow the majority of these notes, without knowing about de Bruijn terms. Just omit the sections on this topic.** Here is our grammar of raw de Bruijn expressions

$$Exp ::= \mathsf{V}\ \mathbb{N} \mid \lambda\ Exp \mid \$\ Exp\ Exp$$

Recall the basic idea is that the property of syntax which is captured by variable binding, can also be embodied by *node count* in an expression. An example may make this more transparent. Consider

$$e \stackrel{\text{def}}{=} \mathsf{L}\ v^1\ (\mathsf{L}\ v^5\ \mathsf{L}\ v^4\ (\mathsf{E}\ v^5 v^1\ ))$$

The first occurrence of v^1 (binding) indicates that the second occurrence is bound. However, if we draw out the finite tree, we can see that there is a single path between the two variables, and the fact that the second v^1 is bound by the first is “specified” by counting the number, 2, of nodes L *strictly between* them. There is also one L between the two occurrences of v^5. In de Bruijn notation, e is rendered $\lambda\ (\lambda\ (\lambda\ (\mathsf{E}\ 1\ 2)))$. For example, $\lambda\ 0$ corresponds to $\mathsf{L}\ v^0\ v^0$ and $\mathsf{L}\ v^0\ (\mathsf{L}\ v^1 v^0\ )$ to $\lambda\ (\lambda\ 1)$. Now let n range over $\mathbb{N}$. We will regard n as the set $\{0, \ldots, n-1\}$ of natural numbers, with the elements treated as De Bruijn indices. We inductively define a set of judgements, this time of the form $n \vdash^{\mathsf{ib}} e$. The rules appear in Figure 4. One can then prove by rule induction that if $n \vdash^{\mathsf{ib}} e$ then e is a de Bruijn expression of level n. Check the simple details as an *Exercise.*

$$\frac{0 \le j \le n-1}{n \vdash^{\mathsf{ib}} \mathsf{V}\ j} \qquad \frac{n+1 \vdash^{\mathsf{ib}} e}{n \vdash^{\mathsf{ib}} \lambda\ e} \qquad \frac{n \vdash^{\mathsf{ib}} e \quad n \vdash^{\mathsf{ib}} e'}{n \vdash^{\mathsf{ib}} \$\ e\ e'}$$

Fig. 4. Expressions in Environment—Binding via Node Depth

We finish this section with an *Exercise*. Write down rules of the form $\Delta \vdash e$ with e corresponding to the datatype of Section 2.1, and Δ as defined in Section 2.3. Show that in this example, where variables are arbitrary and there is no binding, the set of expressions e for which $\Delta \vdash e$ is precisely the set of elements of the datatype.

3 Category Theory

3.1 Categories

A category consists of two collections. Elements of one collection are called *objects*. Elements of the other collection are called *morphisms*. Speaking informally, each morphism might be thought of as a "relation" or "connection" between two objects. Here are some informal examples.

• The collection of all sets (each set is an object), together with the collection of all set-theoretic functions (each function is morphism).
• The collection of all posets, together with all monotone functions.
• The set of real numbers $\mathbb{R}$ (in this case each object is just a real number r in $\mathbb{R}$), together with the order relation $\le$ on the set $\mathbb{R}$ (a morphism is a pair (r, r') in the set $\le$).

It is important to note that the objects of a category do not have to be sets (in the fourth example they are real numbers) and that the morphisms do not have to be functions (in the fourth example they are elements of the order relation $\le$). Of course, there are some precise rules which define exactly what a category is, and we come to these shortly: the reader may care to glance ahead at the definition given on page 10. We give a more complete example before coming to the formal definition.

We can create an example of a category using the definitions of Section 2.1. We illustrate carefully the general definition of a category using this example.

The collection of *objects* is $\mathbb{N}$ and the collection of *morphisms* is

$$\bigcup_{m \geq 1} [\{\, e \mid \Gamma^m \vdash^{\mathsf{d\bar{b}}} e \,\}]$$

Given any morphism es, there is a corresponding *source* and *target*. We define

$$src(es) \overset{\text{def}}{=} \mathsf{max}\,\{\, i \mid v^i \in e \wedge e \in es \,\} + 1$$

and

$$tar(es) \overset{\text{def}}{=} \text{ case of } \begin{cases} \epsilon \to 0 \\ [\, e_0, \ldots, e_{n-1} \,] \to n \end{cases}$$

We write $es{:}\, m \to n$ or $m \overset{es}{\to} n$ to indicate the source and target of es. For example, we have

$$2 \xrightarrow{[\, \mathsf{A}\ (\mathsf{A}\ v^0\ v^0)\ v^1, \mathsf{A}\ v^1\ v^0, \mathsf{A}\ v^0\ (\mathsf{S}\ v^0) \,]} 3$$

In the following situation

$$\begin{tikzcd} l \arrow[r, "es'"] & m \arrow[r, "es"] & n \end{tikzcd}$$

we say that es and es' are *composable*. This means that there is a morphism $es \circ es'{:}\, l \to n$ which can be defined using es and es', and is said to be their composition. For example, if

$$1 \xrightarrow{[\, \mathsf{S}\ v^0, \mathsf{A}\ v^0\ v^0 \,]} 2 \xrightarrow{[\, \mathsf{A}\ (\mathsf{A}\ v^0\ v^1)\ v^1, \mathsf{A}\ v^1\ v^0, \mathsf{A}\ v^0\ (\mathsf{S}\ v^1) \,]} 3$$

then the composition is

$$1 \xrightarrow{[\, \mathsf{A}\ (\mathsf{A}\ (\mathsf{S}\ v^0)\ (\mathsf{A}\ v^0\ v^0))\ (\mathsf{A}\ v^0\ v^0), \mathsf{A}\ (\mathsf{A}\ v^0\ v^0)\ (\mathsf{S}\ v^0), \mathsf{A}\ (\mathsf{S}\ v^0)\ (\mathsf{S}\ (\mathsf{A}\ v^0\ v^0)) \,]} 3$$

Informally, the general definition of composition is that the element in position p in $es \circ es'$ is the element in es in position p in which the m elements of es' are substituted simultaneously for the free variables $v^0, \ldots v^{m-1}$. We leave the actual definition as an *Exercise*—do not forget what happens if either es or es' is empty. Finally, for any object m there is an *identity* morphism,

$$m \xrightarrow{[\, v^0, \ldots, v^{m-1} \,]} m$$

We now give the formal definition of a category. A **category** $\mathcal{C}$ is specified by the following data:

• A collection $ob\ \mathcal{C}$ of entities called **objects**. An object will often be denoted by a capital letter such as $A, B, C \dots$
• A collection $mor\ \mathcal{C}$ of entities called **morphisms**. A morphism will often be denoted by a small letter such as $f, g, h \dots$
• Two operations assigning to each morphism f its **source** $src(f)$ which is an object of $\mathcal{C}$ and its **target** $tar(f)$ also an object of $\mathcal{C}$. We write $f\colon src(f) \longrightarrow tar(f)$ to indicate this, or perhaps $f\colon A \to B$ where $A = src(f)$ and $B = tar(f)$. Sometimes we just say "let $f\colon A \to B$ be a morphism of $\mathcal{C}$" to mean f is a morphism of $\mathcal{C}$ with source A and target B.
• Morphisms f and g are **composable** if $tar(f) = src(g)$. There is an operation assigning to each pair of composable morphisms f and g their **composition** which is a morphism denoted by $g \circ f$ and such that $src(g \circ f) = src(f)$ and $tar(g \circ f) = tar(g)$. So for example, if $f\colon A \to B$ and $g\colon B \to C$, then there is a morphism $g \circ f\colon A \to C$. There is also an operation assigning to each object A of $\mathcal{C}$ an **identity** morphism $id_A\colon A \to A$. These operations are required to be **unitary**

$$id_{tar(f)} \circ f = f$$
$$f \circ id_{src(f)} = f$$

and **associative**, that is given morphisms $f\colon A \to B$, $g\colon B \to C$ and $h\colon C \to D$ then

$$(h \circ g) \circ f \ = \ h \circ (g \circ f).$$

As an *Exercise* check that the operations from the previous example are unitary and associative.

Here are some more examples of categories.

1. The category of sets and total functions, $\mathcal{S}et$. The objects of the category are **sets** and the morphisms are **functions** which are triples (A, f, B) where A and B are sets and $f \subseteq A \times B$ is a subset of the cartesian product of A and B for which

   $$(\forall a \in A)(\exists! b \in B)((a, b) \in f)$$

   We sometimes call f the **graph** of the function (A, f, B). The source and target operations are defined by $src(A, f, B) \overset{\text{def}}{=} A$ and $tar(A, f, B) \overset{\text{def}}{=} B$. Suppose that we have another morphism (B, g, C). Then $tar(A, f, B) = src(B, g, C)$, and the composition is given by

   $$(B, g, C) \circ (A, f, B) = (A, g \circ f, C)$$

where $g \circ f$ is the usual composition of the graphs f and g. Finally, if A is any set, the identity morphism assigned to A is given by (A, id_A, A) where $id_A \subseteq A \times A$ is the identity graph. We leave the reader to check as an *Exercise* that composition is an associative operation and that composition by identities is unitary. *Note: we now follow informal practice, and talk about "functions" f as morphisms, even though f is strictly speaking the graph component of the function* (A, f, B).

2. The category of sets and partial functions, $\mathcal{P}art$. The objects are sets and the morphisms are partial functions. The definition of composition is the expected one, namely given $f\colon A \to B$, $g\colon B \to C$, then for each element a of A, $g \circ f(a)$ is defined with value $g(f(a))$ if both $f(a)$ and $g(f(a))$ are defined, and is otherwise not defined.
3. Any preordered set $(X, \leq)$ may be viewed as a category. Recall that a preorder $\leq$ on a set X is a reflexive, transitive relation on X. The set of objects is X. The set of morphisms is $\leq$. $(X, \leq)$ forms a category with identity morphisms (x, x) for each object x (because $\leq$ is reflexive) and composition $(y, z) \circ (x, y) \stackrel{\text{def}}{=} (x, z)$ (because $\leq$ is transitive). Note for x and y elements of X, there is at most one morphism from x to y according to whether $x \leq y$ or not.
4. The category $\mathcal{P}reset$ has objects all preordered sets, and morphisms the **monotone** functions. More precisely, a morphism with source $(X, \leq_X)$ and target $(Y, \leq_Y)$ is specified by giving a function $f\colon X \to Y$ such that if $x \leq_X x'$ then $f(x) \leq_Y f(x')$.
5. A **discrete** category is one for which the only morphisms are identities. So a very simple example of a discrete category is given by regarding any set as a category in which the objects are the elements of the set, there is an identity morphism for each element, and there are no other morphisms.
6. The objects of the category $\mathbb{F}$ are the elements $n \in \mathbb{N}$, where we regard n as the set $\{0, \dots, n-1\}$ for $n \geq 1$, and 0 is the empty set $\varnothing$. A morphism $\rho\colon n \to n'$ is any set-theoretic function.
7. Let $\mathcal{C}$ and $\mathcal{D}$ be categories. The **product category** $\mathcal{C} \times \mathcal{D}$ has as objects all pairs (C, D) where C is an object of $\mathcal{C}$ and D is an object of $\mathcal{D}$, and the morphisms are the obvious pairs $(f, g)\colon (C, D) \to (C', D')$.

It is an *Exercise* to work through the details of these examples.

In category theory, a notion which is pervasive is that of *isomorphism.* If two objects are isomorphic, then they are very similar, but not necessarily identical. In the case of $\mathcal{S}et$, two sets X and Y are isomorphic just in case there is a bijection between them—informally, they have the same number of elements. The usual definition of bijection is that there is a function $f\colon X \to Y$ which is injective and surjective. Equivalently, there are functions $f\colon X \to Y$ and $g\colon Y \to X$ which are mutually inverse. We can use the idea that a pair of mutually inverse functions in the category $\mathcal{S}et$ gives rise to bijective sets to define the notion of isomorphism in an arbitrary category.

A morphism $f\colon A \to B$ in a category $\mathcal{C}$ is said to be an **isomorphism** if there is some $g\colon B \to A$ for which $f \circ g = id_B$ and $g \circ f = id_A$. We say that g is an **inverse** for f and that f is an inverse for g. Given objects A and B in $\mathcal{C}$, we say that A is **isomorphic** to B and write $A \cong B$ if such a mutually inverse pair of morphisms exists, and we say that the pair of morphisms **witnesses** the fact that $A \cong B$. Note that there may be many such pairs. In the category determined by a partially ordered set, the only isomorphisms are the identities, and in a preorder X with $x, y \in X$ we have $x \cong y$ iff $x \leq y$ and $y \leq x$. Note that in this case there can be only one pair of mutually inverse morphisms witnessing the fact that $x \cong y$. Here are a couple of *Exercise*s.

(1) Let $\mathcal{C}$ be a category and let $f\colon A \to B$ and $g, h\colon B \to A$ be morphisms. If $f \circ h = id_B$ and $g \circ f = id_A$ show that $g = h$. Deduce that any morphism f has a **unique** inverse if such exists.
(2) Let $\mathcal{C}$ be a category and $f\colon A \to B$ and $g\colon B \to C$ be morphisms. If f and g are isomorphisms, show that $g \circ f$ is too. What is its inverse?

3.2 Functors

A function $f\colon X \to Y$ is a relation between two sets. We can think of the function f as specifying an element of Y for each element of X; from this point of view, f is rather like a program which outputs a value $f(x) \in Y$ for each $x \in X$. We might say that the element $f(x)$ is **assigned** to x. A functor is rather like a function between two categories. Roughly, a functor from a category $\mathcal{C}$ to a category $\mathcal{D}$ is an assignment which sends each object of $\mathcal{C}$ to an object of $\mathcal{D}$, and each morphism of $\mathcal{C}$ to a morphism of $\mathcal{D}$. This assignment has to satisfy some rules. For example, the identity on an object A of $\mathcal{C}$ is sent to the identity in $\mathcal{D}$ on the object FA, where the functor sends the object A in $\mathcal{C}$ to FA in $\mathcal{D}$. Further, if two morphisms in $\mathcal{C}$ compose, then their images under the functor must compose in $\mathcal{D}$. Very informally, we might think of the functor as "preserving the structure" of $\mathcal{C}$. Let us move to the formal definition of a functor.

A **functor** F between categories $\mathcal{C}$ and $\mathcal{D}$, written $F\colon \mathcal{C} \to \mathcal{D}$, is specified by

- an operation assigning objects FA in $\mathcal{D}$ to objects A in $\mathcal{C}$, and
- an operation assigning morphisms $Ff\colon FA \to FB$ in $\mathcal{D}$, to morphisms $f\colon A \to B$ in $\mathcal{C}$,

for which $F(id_A) = id_{FA}$, and whenever the composition of morphisms $g \circ f$ is defined in $\mathcal{C}$ we have $F(g \circ f) = Fg \circ Ff$. Note that $Fg \circ Ff$ is defined in $\mathcal{D}$ whenever $g \circ f$ is defined in $\mathcal{C}$, that is, Ff and Fg are composable in $\mathcal{D}$ whenever f and g are composable in $\mathcal{C}$.

Sometimes we give the specification of a functor F by writing the operation on an object A as $A \mapsto FA$ and the operation on a morphism f, where $f\colon A \to B$, as $f\colon A \to B \mapsto Ff\colon FA \to FB$. Provided that everything is clear, we sometimes say "the functor $f\colon \mathcal{C} \to \mathcal{D}$ is defined by an assignment

$$f\colon A \longrightarrow B \quad \mapsto \quad Ff\colon FA \longrightarrow FB$$

where $f\colon A \to B$ is any morphism of $\mathcal{C}$." We refer informally to $\mathcal{C}$ as the source of the functor F, and to $\mathcal{D}$ as the target of F. Here are some examples.

1. Let $\mathcal{C}$ be a category. The **identity** functor $id_{\mathcal{C}}$ is defined by $id_{\mathcal{C}}(A) \stackrel{\text{def}}{=} A$ where A is any object of $\mathcal{C}$ and $id_{\mathcal{C}}(f) \stackrel{\text{def}}{=} f$ where f is any morphism of $\mathcal{C}$.
2. We may define a functor $F\colon \mathcal{S}et \to \mathcal{S}et$ by taking the operation on objects to be $FA \stackrel{\text{def}}{=} [A]$ and the operation on morphisms $Ff \stackrel{\text{def}}{=} map(f)$, where the function $map(f)\colon [A] \to [B]$ is defined by

$$map(f)(as) \stackrel{\text{def}}{=} \text{case of} \begin{cases} \epsilon \to \epsilon \\ ([a_0, \ldots, a_{l-1}]) = [f(a_0), \ldots, f(a_{l-1})] \end{cases}$$

Being our first example, we give explicit details of the verification that F is indeed a functor. To see that $F(id_A) = id_{FA}$ note that on non-empty lists

$$\begin{aligned} F(id_A)([a_0,\ldots,a_{l-1}]) &\overset{\text{def}}{=} map(id_A)([a_0,\ldots,a_{l-1}]) \\ &= [a_0,\ldots,a_{l-1}] \\ &= id_{[A]}([a_0,\ldots,a_{l-1}]) \\ &\overset{\text{def}}{=} id_{FA}([a_0,\ldots,a_{l-1}]), \end{aligned}$$

and to see that $F(g \circ f) = Fg \circ Ff$ note that

$$\begin{aligned} F(g \circ f)([a_0,\ldots,a_{l-1}]) &\overset{\text{def}}{=} map(g \circ f)([a_0,\ldots,a_{l-1}]) \\ &= [g(f(a_0)),\ldots,g(f(a_{l-1}))] \\ &= map(g)([f(a_0),\ldots,f(a_{l-1})]) \\ &= map(g)(map(f)([a_0,\ldots,a_{l-1}])) \\ &= Fg \circ Ff([a_0,\ldots,a_{l-1}]). \end{aligned}$$

3. Given categories $\mathcal{C}$ and $\mathcal{D}$ and an object D of $\mathcal{D}$, the **constant** functor $\tilde{D}: \mathcal{C} \to \mathcal{D}$ sends any object A of $\mathcal{C}$ to D and any morphism $f: A \to B$ of $\mathcal{C}$ to $id_D: D \to D$.
4. Given a set A, recall that the powerset $\mathcal{P}(A)$ is the set of subsets of A. We can define a functor $\mathcal{P}(f): \mathcal{S}et \to \mathcal{S}et$ which is given by

$$f: A \to B \quad \mapsto \quad \mathcal{P}(f): \mathcal{P}(A) \to \mathcal{P}(B),$$

where $f: A \to B$ is a function and $\mathcal{P}(f)$ is defined by

$$\mathcal{P}(f)(A') \overset{\text{def}}{=} \{f(a') \mid a' \in A'\}$$

where $A' \in \mathcal{P}(A)$. We call $\mathcal{P}(f): \mathcal{S}et \to \mathcal{S}et$ the **covariant powerset** functor.

5. Given functors $F: \mathcal{C} \to \mathcal{C}'$ and $G: \mathcal{D} \to \mathcal{D}'$, the **product functor**

$$F \times G: \mathcal{C} \times \mathcal{D} \to \mathcal{C}' \times \mathcal{D}'$$

assigns the morphism $(Ff, Gg): (FC, FD) \to (FC', FD')$ to any morphism $(f, g): (C, D) \to (C', D')$ in $\mathcal{C} \times \mathcal{D}$.
6. Any functor between two preorders A and B regarded as categories is precisely a monotone function from A to B.

It is an *Exercise* to check that the definitions define functors between categories.

3.3 Natural Transformations

Let $\mathcal{C}$ and $\mathcal{D}$ be categories and $F, G\colon \mathcal{C} \to \mathcal{D}$ be functors. Then a **natural transformation** α from F to G, written $\alpha\colon F \to G$, is specified by an operation which assigns to each object A in $\mathcal{C}$ a morphism $\alpha_A\colon FA \to GA$ in $\mathcal{D}$, such that for any morphism $f\colon A \to B$ in $\mathcal{C}$, we have $Gf \circ \alpha_A = \alpha_B \circ Ff$. We denote this equality by the following diagram

$$\begin{tikzcd}
FA \arrow[r, "\alpha_A"] \arrow[d, "Ff"'] & GA \arrow[d, "Gf"] \\
FB \arrow[r, "\alpha_B"'] & GB
\end{tikzcd}$$

which is said to **commute**. The morphism α_A is called the **component** of the natural transformation α at A. We also write $\alpha\colon F \to G\colon \mathcal{C} \to \mathcal{D}$ to indicate that α is a natural transformation between the functors $F, G\colon \mathcal{C} \to \mathcal{D}$. If we are given such a natural transformation, we refer to the above commutative square by saying "consider naturality of α at $f\colon A \to B$."

1. Recall the functor $F\colon \mathcal{S}et \to \mathcal{S}et$ defined on page 14. We can define a natural transformation $rev\colon F \to F$ which has components $rev_A\colon [A] \to [A]$ defined by

   $$rev_A(as) \stackrel{\text{def}}{=} \text{case of } \begin{cases} \epsilon \to \epsilon \\ [a_0, \ldots, a_{l-1}] \to [a_{l-1}, \ldots, a_0] \end{cases}$$

   where $[A]$ is the set of finite lists over A (see Appendix). It is trivial to see that this does define a natural transformation:

   $$Ff \circ rev_A([a_0, \ldots, a_{l-1}]) = [f(a_{l-1}), \ldots, f(a_0)] = rev_B \circ Ff([a_0, \ldots, a_{l-1}]).$$

2. Let X and A be sets. Write $X \to A$ for the set of functions from X to A, and let $(X \to A) \times X$ be the usual cartesian product of sets. Define a functor $F_X\colon \mathcal{S}et \to \mathcal{S}et$ by setting $F_X(A) \stackrel{\text{def}}{=} (X \to A) \times X$ where A is any set and letting

   $$F_X(f)\colon (X \to A) \times X \longrightarrow (X \to B) \times X$$

   be the function defined by $(g, x) \mapsto (f \circ g, x)$ where $f\colon A \to B$ is any function and $(g, x) \in (X \to A) \times X$. Then we can define a natural transformation $ev\colon F_X \to id_{\mathcal{S}et}$ by setting $ev_A(g,x) \overset{\text{def}}{=} g(x)$. To see that we have defined a natural transformation ev with components $ev_A\colon (X \to A)\times X \to A$ let $f\colon A \to B$ be a set function, $(g,x) \in (X \to A) \times X$, and note that

$$\begin{aligned}
(id_{\mathcal{S}et}(f) \circ ev_A)(g,x) &= f(ev_A(g,x)) \\
&= f(g(x)) \\
&= ev_B(f \circ g, x) \\
&= ev_B(F_X(f)(g,x)) \\
&= (ev_B \circ F_X(f))(g,x).
\end{aligned}$$

It will be convenient to introduce some notation for dealing with methods of "composing" functors and natural transformations. Let $\mathcal{C}$ and $\mathcal{D}$ be categories and let F, G, H be functors from $\mathcal{C}$ to $\mathcal{D}$. Also let $\alpha\colon F \to G$ and $\beta\colon G \to H$ be natural transformations. We can define a natural transformation $\beta \circ \alpha\colon F \to H$ by setting the components to be $(\beta \circ \alpha)_A \overset{\text{def}}{=} \beta_A \circ \alpha_A$. This yields a category $\mathcal{D}^{\mathcal{C}}$ with objects functors from $\mathcal{C}$ to $\mathcal{D}$, morphisms natural transformations between such functors, and composition as given above. $\mathcal{D}^{\mathcal{C}}$ is called the **functor category** of $\mathcal{C}$ and $\mathcal{D}$. As an *Exercise*, given categories $\mathcal{C}$ and $\mathcal{D}$, verify that $\mathcal{D}^{\mathcal{C}}$ is indeed a category. For example, one thing to check is that $\beta \circ \alpha$ as defined above is indeed natural.

An isomorphism in a functor category is referred to as a **natural isomorphism**. If there is a natural isomorphism between the functors F and G, then we say that F and G are **naturally isomorphic**.

Lemma 1. *Let $\alpha\colon F \to G\colon \mathcal{C} \to \mathcal{D}$ be a natural transformation. Then α is a natural isomorphism just in case each component α_C is an isomorphism in $\mathcal{D}$. More precisely, if we are given a natural isomorphism α in $\mathcal{D}^{\mathcal{C}}$ with inverse β, then each β_C is an inverse for α_C in $\mathcal{D}$; and if given a natural transformation α in $\mathcal{D}^{\mathcal{C}}$ for which each component α_C has an inverse (say β_C) in $\mathcal{D}$, then the β_C are the components of a natural transformation β which is the inverse of α in $\mathcal{D}^{\mathcal{C}}$.*

Proof. Direct calculations from the definitions, left as an *Exercise.*

In order to consider categorical models of syntax, we shall require some notation and facts concerning a particular functor category. The following definitions will be used in Section 4.

Recall the category $\mathbb{F}$ defined on page 12. The functor category which will be of primary concern is $\mathcal{S}et^{\mathbb{F}}$. We will write $\mathcal{F}$ for it. A typical object $F: \mathbb{F} \to \mathcal{S}et$ is an example[3] of a **presheaf** over $\mathbb{F}$. A simple example is given by the powerset functor $\mathcal{P}$, defined on page 15, restricted to sets of the form n. Another (trivial) example is the **empty presheaf** $\varnothing$ which maps any $\rho: n \to m$ to the empty function with empty source and target.

Now let F and F' be two such objects (presheaves). Suppose that for any n in $\mathbb{F}$, $F'n \subset Fn$, and that the diagram

$$\begin{tikzcd}
F'n \arrow[r, phantom, "\subset"] \arrow[d, "F'\rho"'] & Fn \arrow[d, "F\rho"] \\
F'n' \arrow[r, phantom, "\subset"] & Fn'
\end{tikzcd}$$

commutes for any $\rho: n \to n'$. This gives rise to a natural transformation, which we denote by $i: F' \hookrightarrow F$.

We also need a functor $\delta: \mathcal{F} \to \mathcal{F}$. Suppose that F is an object in $\mathcal{F}$. Then $\delta\, F$ is defined by assigning to any morphism $\rho: n \to n'$ in $\mathbb{F}$ the function

$$(\delta\, F)\rho \stackrel{\text{def}}{=} F(\rho + id_1): F(n+1) \longrightarrow F(n'+1)$$

If $\alpha: F \to F'$ in $\mathcal{F}$, then the components of $\delta\, \alpha$ are given by $(\delta\, \alpha)_n \stackrel{\text{def}}{=} \alpha_{n+1}$. It is an *Exercise* to verify all the details.

Lemma 2. *Suppose that* $(S_r \mid r \geq 0)$ *is a family of presheaves in* $\mathcal{F}$*, with* $i_r: S_r \hookrightarrow S_{r+1}$ *for each* r*. Then there is a* ***union*** *presheaf* T *in* $\mathcal{F}$*, such that* $i_r': S_r \hookrightarrow T$*. We sometimes write* $\cup_r S_r$ *for* T*.*

Proof. Let $\rho: n \to n'$ be any morphism in $\mathbb{F}$. Then we define $Tn \stackrel{\text{def}}{=} \bigcup_r S_r n$, and the function $T\rho: Tn \to Tn'$ is defined by $(T\rho)(\xi) \stackrel{\text{def}}{=} (S_r\rho)(\xi)$ where $\xi \in Tn$, and r is any index for which $\xi \in S_r(n)$. It is an *Exercise* to verify that we have defined a functor. Why is it well-defined? *Hint: prove that*

$$(\forall r, r' \geq 0)(r' \geq r \implies S_r \hookrightarrow S_{r'})$$

Lemma 3. *Let* $(\phi_r: S_r \to A \mid r \geq 0)$ *be a family of natural transformations in* $\mathcal{F}$ *with the* S_r *as in Lemma 2, and such that* $\phi_{r+1} \circ i_r = \phi_r$*. Then there is a unique natural transformation* $\phi: T \to A$*, such that* $\phi \circ i_r' = \phi_r$*.*

[3] In general, we call $\mathcal{S}et^{\mathcal{C}}$ the category of presheaves over $\mathcal{C}$.

Proof. It is an *Exercise* to prove the lemma. The proof requires a simple calculation using the definitions. *Hint: Note that there are functions* $\phi_n : Tn \to An$ *where we set* $\phi_n(\xi) \overset{\text{def}}{=} (\phi_r)_n(\xi)$ *for* $\xi \in S_r n$*. The conditions of the lemma (trivially) imply the existence and uniqueness of the* ϕ_n*, which are natural in* n*.*

3.4 Products

The notion of "product of two objects in a category" can be viewed as an abstraction of the idea of a cartesian product of two sets. The definition of a cartesian product of sets is an "internal" one; we specify the elements of the product in terms of the elements of the sets from which the product is composed. However the cartesian product has a particular property, namely

(*Property* Φ) *Given* any two sets A and B, *then* there is a set P and functions $\pi : P \to A$, $\pi' : P \to B$ *such that* the following condition holds: given any functions $f : C \to A$, $g : C \to B$ with C any set, then there is a unique function $h : C \to P$ making the diagram

$$\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[d, "h"] \arrow[dr, "g"] & \\
A & P \arrow[l, "\pi"] \arrow[r, "\pi'"'] & B
\end{tikzcd}$$

commute. End of definition of (*Property* Φ).

Let us investigate an instance of (*Property* Φ) in the case of two given sets A and B. Suppose that $A \overset{\text{def}}{=} \{a, b\}$ and $B \overset{\text{def}}{=} \{c, d, e\}$. Let us take P to be $A \times B \overset{\text{def}}{=} \{(x, y) \mid x \in A, y \in B\}$ and the functions π and π' to be coordinate projection to A and B respectively, and see if (P, π, π') makes the instance of (*Property* Φ) for the given A and B hold. Let C be any other set and $f : C \to A$ and $g : C \to B$ be any two functions. Define the function $h : C \to P$ by $z \mapsto (f(z), g(z))$. We leave the reader to verify that indeed $f = \pi \circ h$ and $g = \pi' \circ h$, and that h is the only function for which these equations hold with the given f and g. Now define $P' \overset{\text{def}}{=} \{1, 2, 3, 4, 5, 6\}$ along with functions $p : P' \to A$ and $q : P' \to B$ where

$$\begin{array}{lllcll}
p(1), & p(2), & p(3) = a & \qquad & q(1), & q(4) = c \\
p(4), & p(5), & p(6) = b & & q(2), & q(5) = d \\
 & & & & q(3), & q(6) = e
\end{array}$$

In fact (P', p, q) also makes the instance of (*Property* Φ) for the given A and B hold true. To see this, one can check by enumerating six cases that there is a unique function $h: C \to P'$ for which $f = p \circ h$ and $g = q \circ h$ (for example, if $x \in C$ and $f(x) = a$ and $g(x) = d$ then we must have $h(x) = 2$, and this is one case).

Now notice that there is a bijection between P (the cartesian product

$$\{(a,c),(a,d),(a,e),(b,c),(b,d),(b,e)\}$$

of A and B) and P'. In fact any choices for the set P can be shown to be bijective. It is very often useful to determine sets up to bijection rather than worry about their elements or "internal make up," so we might consider taking (*Property Φ*) as a definition of cartesian product of two sets and think of the P and P' in the example above as two implementations of the notion of cartesian product of the sets A and B. Of course (*Property Φ*) only makes sense when talking about the collection of sets and functions; we can give a definition of cartesian product for an arbitrary category which is exactly (*Property Φ*) for the "category" of sets and functions.

A **binary product** of objects A and B in a category $\mathcal{C}$ is specified by

- an object $A \times B$ of $\mathcal{C}$, together with
- two **projection** morphisms $\pi_A: A \times B \to A$ and $\pi_B: A \times B \to B$,

for which given any object C and morphisms $f: C \to A$, $g: C \to B$, there is a unique morphism $\langle f, g\rangle: C \to A \times B$ for which $\pi_A \circ \langle f, g\rangle = f$ and $\pi_B \circ \langle f, g\rangle = g$.

We refer simply to a binary product $A \times B$ instead of $(A \times B, \pi_A, \pi_B)$, without explicit mention of the projection morphisms. The data for a binary product is more readily understood as a commutative diagram, where we have written $\exists!$ to mean "there exists a unique":

$$\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[d, dashed, "\exists!\ \langle f,g\rangle"] \arrow[dr, "g"] & \\
A & A \times B \arrow[l, "\pi_A"] \arrow[r, "\pi_B"'] & B
\end{tikzcd}$$

Given a binary product $A \times B$ and morphisms $f: C \to A$ and $g: C \to B$, the unique morphism $\langle f, g\rangle: C \to A \times B$ (making the above diagram commute) is called the **mediating** morphism for f and g. We sometimes refer to a property which involves the "existence of a unique morphism" leading to a structure which is determined up to isomorphism as a **universal** property. We also call $\langle f, g\rangle$ the **pair** of f and g. We say that the category $\mathcal{C}$ **has binary products** if there is a product in $\mathcal{C}$ of any two objects A and B, and that $\mathcal{C}$ has **specified** binary products if there is a given canonical choice of binary product for each pair of objects. For example, in $\mathcal{S}et$ we can specify binary products by setting $A \times B \stackrel{\text{def}}{=} \{ (a,b) \mid a \in A, b \in B \}$ with projections given by the usual set-theoretic projection functions. Talking of **specified** binary products is a reasonable thing to do: given A and B, any binary product of A and B will be isomorphic to the specified $A \times B$.

Lemma 4. *A binary product of A and B in a category $\mathcal{C}$ is unique up to isomorphism if it exists.*

Proof. Suppose that (P, p_A, p_B) and (P', p'_A, p'_B) are two candidates for the binary product. Then we have $\langle p_A, p_B\rangle : P \to P'$ by applying the defining property of (P', p'_A, p'_B) to the morphisms $p_A : P \to A$, $p_B : P \to B$, and further $\langle p'_A, p'_B\rangle : P' \to P$ exists from a similar argument. So we have diagrams of the form

$$\begin{tikzcd}
 & P \arrow[dl, "p_A"'] \arrow[d, dashed, "\langle p_A, p_B\rangle"] \arrow[dr, "p_B"] & \\
A & P' \arrow[l, "p'_A"] \arrow[r, "p'_B"'] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
 & P' \arrow[dl, "p'_A"'] \arrow[d, dashed, "\langle p'_A, p'_B\rangle"] \arrow[dr, "p'_B"] & \\
A & P \arrow[l, "p_A"] \arrow[r, "p_B"'] & B
\end{tikzcd}$$

But then $f \stackrel{\text{def}}{=} \langle p'_A, p'_B\rangle \circ \langle p_A, p_B\rangle : P \to P$ and one can check that $p_A \circ f = p_A$ and that $p_B \circ f = p_B$, that is f is a mediating morphism for the binary product (P, p_A, p_B); we can picture this as the following commutative diagram:

$$\begin{tikzcd}
 & P \arrow[dl, "p_A"'] \arrow[d, dashed, "f"] \arrow[dr, "p_B"] & \\
A & P \arrow[l, "p_A"] \arrow[r, "p_B"'] & B
\end{tikzcd}$$

But it is trivial that id_P is also such a mediating morphism, and so uniqueness implies $f = id_P$. Similarly one proves that $\langle p_A, p_B\rangle \circ \langle p'_A, p'_B\rangle = id_{P'}$, to deduce $P \cong P'$ witnessed by the morphisms $\langle p_A, p_B\rangle$ and $\langle p'_A, p'_B\rangle$.

Here are some examples

1. The category $\mathcal{Preset}$ has binary products. Given objects $A \stackrel{\text{def}}{=} (X, \leq_X)$ and $B \stackrel{\text{def}}{=} (Y, \leq_Y)$, the product object $A \times B$ is given by $(X \times Y, \leq_{X \times Y})$ where $X \times Y$ is cartesian product, and $(x, y) \leq_{X\times Y} (x', y')$ just in case $x \leq_X x'$ and $y \leq_Y y'$. It is an *Exercise* to check the details.
2. The category $\mathcal{Part}$ has binary products. Given objects A and B, the binary product object is defined by

$$(A \times B) \cup (A \times \{ *_A \}) \cup (B \times \{ *_B \})$$

where $\times$ is cartesian product, and $*_A$ and $*_B$ are distinct elements not in A nor in B. The project π_A is undefined on $B \times \{ *_B \}$ and π_B is undefined on $A \times \{ *_A \}$. Of course $\pi_A(a, *_A) = a$ for all $a \in A$, and $\pi_B(b, *_B) = b$ for all $b \in B$.
3. The category $\mathbb{F}$ has binary products. The product of n and m is written $n \times m$ and is given by $n * m$, that is, the set $\{ 0, \ldots, (n * m) - 1 \}$. We leave it as an *Exercise* to formulate possible definitions of projection functions. *Hint: Think about the general illustration of products at the start of this section.*
4. Consider a preorder $(X, \leq)$ which has all binary meets $x \wedge y$ for $x, y \in X$ as a category. It is an *Exercise* to verify that binary meets are binary products.

It will be useful to have a little additional notation which will be put to use later on.

We can define the **ternary product** $A \times B \times C$ for which there are three projections, and any mediating morphism can be written $\langle f, g, h \rangle$ for suitable f, g and h. It is an *Exercise* to make all the definitions and details precise.

Let $\mathcal{C}$ be a category with finite products and take morphisms $f\colon A \to B$ and $f'\colon A' \to B'$. We write

$$f \times f'\colon A \times A' \to B \times B'$$

for the morphism $\langle f \circ \pi, f' \circ \pi' \rangle$ where $\pi\colon A \times A' \to A$ and $\pi'\colon A \times A' \to A'$. The uniqueness condition (universal property) of mediating morphisms means that in general one has

$$id_A \times id_{A'} = id_{A \times A'} \qquad \text{and} \qquad (g \times g') \circ (f \times f') = g \circ f \times g' \circ f',$$

where $g: B \to C$ and $g': B' \to C'$. Thus in fact we have a functor

$$\times: \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$$

where $\mathcal{C} \times \mathcal{C}$ is a product of categories. Note that we sometimes write the image of (A, A) or (f, f) under $\times$ as A^2 or f^2.

We finish this section with another example, and some exercises.

The category $\mathcal{F} \stackrel{\text{def}}{=} \mathcal{S}et^{\mathbb{F}}$ has binary products. If F and F' are presheaves in $\mathcal{F}$, the product object $F \times F': \mathbb{F} \to \mathcal{S}et$ is defined on objects n in $\mathbb{F}$ by

$$(F \times F')n \stackrel{\text{def}}{=} (Fn) \times (F'n).$$

Now let $\rho: n \to n'$ be a morphism in $\mathbb{F}$. Hence $(F \times F')\rho$ should be a morphism with source and target

$$(Fn) \times (F'n) \longrightarrow (Fn') \times (F'n')$$

In fact we define $(F \times F')\rho \stackrel{\text{def}}{=} (F\rho) \times (F'\rho)$ where we consider $\times: \mathcal{S}et \times \mathcal{S}et \longrightarrow \mathcal{S}et$. The projection $\pi_F: F \times F' \to F$ is defined by giving components $(\pi_F)_n \stackrel{\text{def}}{=} \pi_{Fn}$ where n is an object of $\mathbb{F}$, with $\pi_{F'}$ defined similarly.

1. Verify that the projections are natural transformations (morphisms in $\mathcal{F}$), and that these definitions do yield a binary product.
2. Verify the equalities (uniqueness conditions) given above.
3. Let $\mathcal{C}$ be a category with finite products and let

$$\begin{array}{lll} l: X \to A & f: A \to B & g: A \to C \\ h: B \to D & k: C \to E & \end{array}$$

   be morphisms of $\mathcal{C}$. Show that

$$(h \times k) \circ \langle f, g \rangle = \langle h \circ f, k \circ g \rangle \qquad \langle f, g \rangle \circ l = \langle f \circ l, g \circ l \rangle$$

3.5 Coproducts

A coproduct is a dual notion of product. A **binary coproduct** of objects A and B in a category $\mathcal{C}$ is specified by

- an object $A + B$ of $\mathcal{C}$, together with
- two **insertion** morphisms $\iota_A: A \to A + B$ and $\iota_B: B \to A + B$,

such that for each pair of morphisms $f: A \to C$, $g: B \to C$ there exists a unique morphism $[f,g]: A+B \to C$ for which $[f,g] \circ \iota_A = f$ and $[f,g] \circ \iota_B = g$. We can picture this definition through the following commutative diagram:

$$\begin{tikzcd}
A \arrow[r, "\iota_A"] \arrow[dr, "f"'] & A+B \arrow[d, dashed, "{[f,g]}"] & B \arrow[l, "\iota_B"'] \arrow[dl, "g"] \\
& C &
\end{tikzcd}$$

1. In the category $\mathcal{S}et$, the binary coproduct of sets A and B is given by their disjoint union together with the obvious insertion functions. We can define the disjoint union $A+B$ of A and B as the union $(A \times \{1\}) \cup (B \times \{2\})$ with the insertion functions

$$\iota_A : A \to A+B \leftarrow B : \iota_B$$

where ι_A is defined by $a \mapsto (a,1)$ for all $a \in A$, and ι_B is defined analogously. Given functions $f: A \to C$ and $g: B \to C$, then $[f,g]: A+B \to C$ is defined by

$$[f,g](\xi) \overset{\text{def}}{=} \text{case } \xi \text{ of} \quad \begin{array}{l} \iota_A(\xi_A) = (\xi_A, 1) \mapsto f(\xi_A) \\ \iota_B(\xi_B) = (\xi_B, 2) \mapsto f(\xi_B) \end{array}$$

We sometimes say that $[f,g]$ is defined by **case analysis**.

2. The category $\mathcal{P}reset$ has binary coproducts. Given objects $A \overset{\text{def}}{=} (X, \leq_X)$ and $B \overset{\text{def}}{=} (Y, \leq_Y)$, the product object $A+B$ is given by $(X+Y, \leq_{X+Y})$ where $X+Y$ is disjoint union, and $\xi \leq_{X+Y} \xi'$ just in case $\xi = (x,1)$ and $\xi' = (x',1)$ for some x, x' such that $x \leq_X x'$, or $\xi = (y,2)$ and $\xi' = (y',2)$ for some y, y' such that $y \leq_Y y'$. It is an *Exercise* to check the details.
3. In $\mathbb{F}$ the coproduct object of n and m is $n+m$ where we interpret $+$ as addition on $\mathbb{N}$. It is an *Exercise* to work out choices of coproduct insertions. What might we take as the *canonical* projections?
4. Suppose that F and F' are presheaves in $\mathcal{F}$. Let ξ be any object or morphism of $\mathbb{F}$. We define $F+F'$ by setting $(F+F')\xi \overset{\text{def}}{=} (F\xi) + (F'\xi)$, where the latter $+$ means coproduct in $\mathcal{S}et$, extended to morphisms in the section immediately after these exercises (recall the definition of products in

$\mathcal{F}$)! The insertion morphism (natural transformation) $\iota_F: F + F' \to F$ has components $(\iota_F)_n \overset{\text{def}}{=} \iota_{Fn}: (Fn) + (F'n) \to Fn$. ι'_F is defined analogously. We leave it as an *Exercise* to verify that $F + F'$ is indeed a functor, that the insertions are natural, and that the definitions above do give rise to binary coproducts.

5. When does a preordered set have binary coproducts?

We end this section with notation that will be used later on.

One can define the **ternary coproduct** of three objects, which will have three insertions, and mediating morphisms $[f, g, h]$. Fill in the details as an *Exercise*. Of course we can generalize to an n-ary coproduct, and call the mediating morphism a **cotupling**.

Let $\mathcal{C}$ be a category with finite coproducts and take morphisms $f: A \to B$ and $f': A' \to B'$. We write

$$f + f': A + A' \to B + B'$$

for the morphism $[\iota_B \circ f, \iota_{B'} \circ f']$ where $\iota_B: B \to B + B'$ and $\iota_{B'}: B' \to B + B'$. The uniqueness condition of mediating morphisms means that one has

$$id_A + id_{A'} = id_{A+A'} \qquad \text{and} \qquad (g + g') \circ (f + f') = g \circ f + g' \circ f',$$

where $g: B \to C$ and $g': B' \to C'$. So we also have a functor

$$+: \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$$

In a category with binary coproducts, then for any morphisms f, g, h, k, l with certain source and target, the equalities

$$l \circ [f, g] = [l \circ f, l \circ g] \qquad\qquad [f, g] \circ (h + k) = (f \circ h) + (g \circ k)$$

always hold due to the universal property of (binary) coproducts. What are the sources and targets? Prove the equalities.

3.6 Algebras

Let F be an **endofunctor** on $\mathcal{C}$, that is a functor $F: \mathcal{C} \to \mathcal{C}$. An **algebra** for the functor F is specified by a pair (A, σ_A) where A is an object of $\mathcal{C}$ and $\sigma_A: FA \to$

A is a morphism. An **initial** F-algebra (I, σ_I) is an algebra for which given any other (A, σ_A), there is a unique morphism $\overline{f}: I \to A$ such that

$$\begin{tikzcd} FI \arrow[r, "\sigma_I"] \arrow[d, "F\overline{f}"'] & I \arrow[d, "\overline{f}"] \\ FA \arrow[r, "\sigma_A"'] & A \end{tikzcd}$$

commutes. As an *Exercise*, show that if $\sigma_I: FI \to I$ is initial, then so too is $\sigma_{I'}: FI' \to I'$ where I' is any object isomorphic to I, and $\sigma_{I'}$ is a morphism for you to define.

One reason for defining initial algebras is that certain datatypes can be modeled as instances. Here is the rough idea. Each constructor of a datatype can be thought of as a coproduct insertion.[4] Each constructor is applied to a finite number (eg 2) of expressions, constituting a tuple of expressions. This can be thought of as a product (eg binary). The datatype

$$Exp ::= \mathsf{V}\ \mathbb{V} \mid \mathsf{S}\ Exp \mid \mathsf{A}\ Exp\ Exp$$

corresponds to (is modeled by) an object $V + E + (E \times E)$. The recursiveness of the datatype declaration is modeled by requiring

$$E \cong V + E + (E \times E) \qquad \dagger$$

In these notes we shall see how to solve an equation such as † in the category $\mathcal{S}et$. In fact if we define a functor $\Sigma: \mathcal{S}et \to \mathcal{S}et$ by $\Sigma\xi \overset{\text{def}}{=} \tilde{\mathbb{V}} + \xi + (\xi \times \xi)$, then it will turn out that the solution we would construct using the methods below is an initial algebra (σ_E, E). We now give a few examples, and illustrate the solution method.

3.7 The Functor $1 + (-): \mathcal{S}et \longrightarrow \mathcal{S}et$

We write $1: \mathcal{S}et \to \mathcal{S}et$ for the functor which maps any function $f: A \to B$ to $id_{\{*\}}: \{*\} \to \{*\}$ where $\{*\}$ is a one element set. Note that we will often also write 1 for such a set. The functor maps $f: A \to B$ to $id_1 + f: 1 + A \to 1 + B$.

[4] Such insertions must be injective. There are categories in which insertions are *not* injective! In all of the examples in these notes, however, insertions will be injective.

The initial algebra is $\mathbb{N}$ up to isomorphism. We show how to construct an initial algebra—the method will be applied in later sections, in adapted form, to produce models of datatypes which represent syntax (see Section 2).

We set $S_0 \stackrel{\text{def}}{=} \varnothing$ and $S_{r+1} \stackrel{\text{def}}{=} 1 + S_r$. Note that there is a coproduct insertion $\iota_{S_r}\colon S_r \to S_{r+1}$. Note also that there is an inclusion[5] function (morphism) $i_r\colon S_r \hookrightarrow S_{r+1}$ where $i_0 \stackrel{\text{def}}{=} \varnothing\colon S_0 \to S_1$, and $i_{r+1} \stackrel{\text{def}}{=} id_1 + i_r$. The difference is an elementary, but subtle, point! For example, we have $i_1, \iota_{S_1}\colon S_1 \to S_2 = 1 + S_1$ for which $\iota_{S_1}(*, 1) = ((*, 1), 2)$, where $(*, 1) \in S_1 = 1 + \varnothing$. But $i_1 = id_1 + i_0$ and so

$$i_1(*, 1) = [\iota_1 \circ id_1, \iota_{S_1} \circ i_0](*, 1) = \iota_1 \circ id_1(*) = (*, 1)$$

We also write $i'_r\colon S_r \hookrightarrow T$ where $T \stackrel{\text{def}}{=} \cup_r S_r$, and claim that T is the object part of an initial algebra for $1 + (-)$. Note that as $\sigma_T\colon 1 + T \to T$, σ_T must be the copair of two morphisms. We set $\sigma_T \stackrel{\text{def}}{=} [k, k']$ where $k\colon 1 \to T$ and $k'\colon T \to T$, with k and k' defined as follows. Note there is a function

$$\begin{tikzcd} 1 \arrow[r, "\iota_1"] & 1 + \varnothing = S_1 \arrow[r, "i'_1"] & T \end{tikzcd}$$

and we set $k \stackrel{\text{def}}{=} i'_1 \circ \iota_1$. Note there are also functions

$$\begin{tikzcd} S_r \arrow[r, "\iota_{S_r}"] & 1 + S_r = S_{r+1} \arrow[r, "i'_{r+1}"] & T \end{tikzcd}$$

whose composition we call k'_r. It is an *Exercise* to check that $k'_{r+1} \circ i_r = k'_r$ by induction on r. Hence we can legitimately define the function $k'\colon T \to T$ by setting $k'(\xi) \stackrel{\text{def}}{=} k'_r(\xi)$ for any r such that $\xi \in S_r$.

We have to verify that $\sigma_T\colon 1 + T \to T$ is an initial algebra, namely, there is exactly one commutative diagram

$$\begin{tikzcd} 1 + T \arrow[r, "\sigma_T"] \arrow[d, "id_1 + \overline{f}"'] & T \arrow[d, "\overline{f}"] \\ 1 + A \arrow[r, "f"'] & A \end{tikzcd}$$

for any such given f. We define a family of functions $\overline{f}_r\colon S_r \to A$ by setting $\overline{f}_0 \stackrel{\text{def}}{=} \varnothing\colon S_0 \to A$, and recursively $\overline{f}_{r+1} \stackrel{\text{def}}{=} [f \circ \iota_1, f \circ \iota_A \circ \overline{f}_r]$. It is an *Exercise*

[5] That is, $S_r \subset S_{r+1}$ for all $r \geq 0$.

to check that $\overline{f}_{r+1} \circ i_r = \overline{f}_r$. Hence we can legitimately define $\overline{f}\colon T \to A$ by $\overline{f}(\xi) \overset{\mathrm{def}}{=} \overline{f}_r(\xi)$ for any r where $\xi \in S_r$.

To check that the diagram commutes, we have to prove that

$$\overline{f} \circ [k, k'] = f \circ (id_1 + \overline{f})$$

By the universal property of coproducts, this is equivalent to showing

$$[\overline{f} \circ k, \overline{f} \circ k'] = [f \circ \iota_1, f \circ \iota_A \circ \overline{f}]$$

which we can do by checking that the respective components are equal. We give details for $\overline{f} \circ k' = f \circ \iota_A \circ \overline{f}$. Take any element $\xi \in T$. Then we have

$$\begin{aligned}
\overline{f}(k'(\xi)) &= \overline{f}(\iota_{S_r}(\xi)) \\
&= \overline{f}_{r+1}(\iota_{S_r}(\xi)) \\
&= [f \circ \iota_1, f \circ \iota_A \circ \overline{f}_r](\iota_{S_r}(\xi)) \\
&= f(\iota_A(\overline{f}_r(\xi))) \\
&= f(\iota_A(\overline{f}(\xi)))
\end{aligned}$$

The first equality is by definition of k' and k'_r; the second by definition of $\overline{f}$; the third by definition of $\overline{f}_{r+1}$.

A final *Exercise* is to check that $T \cong N$.

3.8 The Functor $A + (-)\colon \mathcal{S}et \longrightarrow \mathcal{S}et$

Let A be a set, $+$ denote coproduct. Then the functor $A + (-)$ has an initial algebra $(A \times \mathbb{N}, \sigma_{A\times\mathbb{N}})$ where $\sigma_{A\times\mathbb{N}}\colon A + (A \times \mathbb{N}) \to A \times \mathbb{N}$ is defined by

$$\begin{aligned}
\sigma_{A\times\mathbb{N}}(\xi) \overset{\mathrm{def}}{=} \;& \mathsf{case}\ \xi\ \mathsf{of} \\
& \iota_A(a) \mapsto (a, 0) \\
& \iota_{A\times\mathbb{N}}(a, n) \mapsto (a, n+1)
\end{aligned}$$

where ι_A and $\iota_{A\times\mathbb{N}}$ are the left and right coproduct insertions, and $n \geq 0$.

Then given any function $f\colon A + S \to S$, we can define $\overline{f}$ where

$$\begin{tikzcd}
A + (A \times \mathbb{N}) \arrow[r, "\sigma_{A\times\mathbb{N}}"] \arrow[d, "id_A + \overline{f}"'] & A \times \mathbb{N} \arrow[d, "\overline{f}"] \\
A + S \arrow[r, "f"'] & S
\end{tikzcd}$$

by setting

$$\overline{f}(a,0) \stackrel{\text{def}}{=} f(\iota_A(a))$$
$$\overline{f}(a,n+1) \stackrel{\text{def}}{=} f(\iota_{A\times\mathbb{N}}(\overline{f}(a,n)))$$

3.9 The Functor $1 + (A \times -)\colon \mathcal{S}et \to \mathcal{S}et$

For $k \geq 1$ we define the set A^k to be the collection of functions $\{ 1,\dots,k \} \to A$. We identify A with A^1; an element $a \in A$ is essentially a function $1 \to A$. If $a \in A$ and $l \in A^k$, then we define $al \in A^{k+1}$ by $al(1) \stackrel{\text{def}}{=} a$ and $al(r) \stackrel{\text{def}}{=} l(r-1)$ for $r \geq 2$.

The functor $1 + (A \times -)$ has an initial algebra (L, σ_L), where we set $L \stackrel{\text{def}}{=} \{ \mathsf{nil} \} \cup (\bigcup_{1\leq k<\omega} A^k)$, and $\sigma_L\colon 1 + (A \times L) \to L$ is defined by

$$\sigma_L(\iota_1(*)) \stackrel{\text{def}}{=} \mathsf{nil}$$
$$\sigma_L(\iota_{A\times L}(a, nil)) \stackrel{\text{def}}{=} a$$
$$\sigma_L(\iota_{A\times L}(a, l)) \stackrel{\text{def}}{=} al$$

It is an *Exercise* to verify that this *does* yield an initial algebra.

4 Models of Syntax

Recall that in Section 2 we defined the syntax terms of an object level programming language by making use of recursive datatypes. Note the phrase "making use of". Recall (pages 4 and 26) that we would like to have an ideal situation (IS) in which

- we could write down a recursive datatype, whose elements are precisely the terms which we are interested in; and
- we have a mathematical model of our syntax which is given as a solution to a recursive equation derived directly from the datatype.

Unfortunately, this is not so easily done when we are considering syntax involving binding. It can be done (as is well known, and illustrated in these notes) for simple syntax terms which involve *algebraic* constructors, such as the example in Section 2.1. We can come close to (IS) for binding syntax, but we can't achieve it exactly using the traditional techniques described here.

In Section 2.1 we wrote down a datatype for expressions e, and then restricted attention to expressions for which $\Gamma^n \vdash^{\mathsf{d}\bar{\mathsf{b}}} e$, that is, the variables occurring in e must come from $v^0, \ldots, v^{n-1}$. This does not conform to standard practice. We would expect to deal with expressions *specified using* metavariables x which would denote *any* of the actual variables v^i. We could of course change the judgement $\Gamma^n \vdash^{\mathsf{d}\bar{\mathsf{b}}} e$ to $\Delta \vdash^{\mathsf{d}\bar{\mathsf{b}}} e$ (as in Section 2.3 and recall the exercise at the end of Section 2) where Δ is *any* finite environment of metavariables $x_0, \ldots, x_{n-1}$ and any x_j denotes an actual variable. In this case, the expressions e such that $\Delta \vdash^{\mathsf{d}\bar{\mathsf{b}}} e$ *correspond exactly* with the expressions given by the datatype, and moreover are the terms of the object syntax. Further, we have a mathematical model described in the first half of Section 4.1, and we manage to achieve (IS).

The reason for describing the judgements $\Gamma^n \vdash^{\mathsf{d}\bar{\mathsf{b}}} e$ in this simple setting, is that they illustrate a methodology which we must apply if we are to get near to (IS) when dealing with binding syntax. In order to "avoid" the extra infrastructure of α-equivalence, and define the terms of the λ-calculus inductively (although not *exactly* as the elements of a datatype), we can restrict to environments Γ^n so that we can choose unique, explicit binding variables in abstractions. Further, the use of environments Γ^n with variables chosen systematically from a specified order will be used crucially when formulating the mathematical models in presheaf categories. Here is the rough idea of how we use presheaves over $\mathbb{F}$ to model expressions. A presheaf associates to every n the set of expressions whose (free) variables appear in Γ^n. And given any morphism $\rho{:}\, n \to n'$ in $\mathbb{F}$, the presheaf associates to ρ a function which, roughly, maps any expression e to

$$e\{v^{\rho(0)}, \ldots, v^{\rho(n-1)}/v^0, \ldots, v^{n-1}\}$$

Thus ρ renames the (free) variables in e.

In this section we give some mathematical models of each system of syntax from Section 2. In each case, we

Step 1 define an abstract endofunctor over $\mathcal{F} \overset{\text{def}}{=} \mathcal{S}et^{\mathbb{F}}$, which bears similarities to the datatype in question;

Step 2 construct an initial algebra T for the endofunctor;

Step 3 show that the datatype and system of syntax gives rise to a functor $Exp{:}\, \mathbb{F} \to \mathcal{S}et$, that is a presheaf in $\mathcal{F}$;

Step 4 complete the picture by showing that $T \cong \mathit{Exp}$ so that T forms an abstract mathematical model of the syntax.

Note: In the remainder of this section, we make considerable use of the notation and lemmas in Section 3.3, the fact that the category $\mathcal{F} = \mathcal{S}et^{\mathbb{F}}$ has products and coproducts, and the fact that both $\times$ and $+$ can be regarded as functors.

4.1 A Model of Syntax with Distinguished Variables and without Binding

Step 1 We define a functor $\Sigma_{\mathbb{V}}$ which "corresponds" to the signature of Section 2.1. First, we define the functor $\mathbb{V}: \mathbb{F} \to \mathcal{S}et$. Let $\rho: m \to n$ in $\mathbb{F}$. Then we set $\mathbb{V}m \overset{\text{def}}{=} \{ v^0, \dots, v^{m-1} \}$ and $\mathbb{V}\rho(v^i) \overset{\text{def}}{=} v^{\rho i}$. It is trivial that $\mathbb{V}$ is a functor. Recall that $\mathcal{S}et^{\mathbb{F}}$ has finite products and coproducts, and moreover that the operations $+$ and $\times$ can be regarded as functors. Thus we can define a functor $\Sigma_{\mathbb{V}}: \mathcal{S}et^{\mathbb{F}} \to \mathcal{S}et^{\mathbb{F}}$ by setting $\Sigma_{\mathbb{V}}\xi \overset{\text{def}}{=} \mathbb{V} + \xi + \xi^2$ where ξ is either an object or a morphism.

Step 2 We now show that the functor $\Sigma_{\mathbb{V}}$ has an initial algebra, which we denote by $\sigma_T: \Sigma_{\mathbb{V}} T \to T$. We define T as the union of a family of presheaves $(S_r \mid r \geq 0)$ which satisfy the conditions of Lemma 2. We set $S_0 \overset{\text{def}}{=} \varnothing$ which is the empty presheaf, and then set

$$S_{r+1} \overset{\text{def}}{=} \Sigma_{\mathbb{V}} S_r = \mathbb{V} + S_r + S_r^2$$

We now check that the conditions of Lemma 2 hold, that is, $i_r: S_r \hookrightarrow S_{r+1}$ for all $r \geq 0$. We use induction over r. It is immediate that $i_0: S_0 \hookrightarrow S_1$ from the definition of S_0. Now suppose that for any r, $i_r: S_r \hookrightarrow S_{r+1}$. We are required to show that $i_{r+1}: S_{r+1} \hookrightarrow S_{r+2}$, that is, for any n in $\mathbb{F}$,

$$\begin{tikzcd}
\mathbb{V}n + S_r n + (S_r n)^2 \arrow[r, phantom, "\subset"] \arrow[d, "\mathbb{V}\rho + S_r\rho + (S_r\rho)^2"'] & \mathbb{V}n + S_{r+1} n + (S_{r+1} n)^2 \arrow[d, "\mathbb{V}\rho + S_{r+1}\rho + (S_{r+1}\rho)^2"] \\
\mathbb{V}n' + S_r n' + (S_r n')^2 \arrow[r, phantom, "\subset"] & \mathbb{V}n' + S_{r+1} n' + (S_{r+1} n')^2
\end{tikzcd}$$

It follows from the induction hypothesis, $S_r n \subset S_{r+1} n$, and the definition of $+$ and $\times$ in $\mathcal{S}et$, that we have subsets as indicated in the diagram. As an *Exercise* make sure that you understand this—you need to examine the definitions of $\times$ and $+$. In fact the top inclusion is the component at n of the natural transformation $\Sigma_{\mathbb{V}} i_r = id_{\mathbb{V}} + i_r + i_r^2$. Thus we have $i_{r+1} = \Sigma_{\mathbb{V}} i_r$. It is an *Exercise* to check that the diagram commutes. Thus we can define $T \overset{\text{def}}{=} \bigcup_r S_r$ in $\mathcal{F}$.

Next we consider the structure map σ_T. This natural transformation in $\mathcal{F}$ has source and target $\mathbb{V} + T + T^2 \to T$ and so it is given by $\sigma_T \overset{\text{def}}{=} [\kappa, \kappa', \kappa'']$, the cotupling of (insertion) natural transformations

$$\begin{aligned} \kappa &: \mathbb{V} \to T \\ \kappa' &: T \to T \\ \kappa'' &: T^2 \to T \end{aligned}$$

For the first morphism, note that $\mathbb{V} \cong S_1$, so that $\kappa\colon \mathbb{V} \cong S_1 \hookrightarrow T$. It is a simple *Exercise* to check that you understand the definition of κ; do not forget that $S_1 = \mathbb{V} + \varnothing + \varnothing^2$, and so $S_1 n = \mathbb{V}n \times \{1\}$. We define κ' by specifying the family of morphisms

$$\kappa'_r\colon S_r \xrightarrow{\iota_{S_r}} \mathbb{V} + S_r + S_r^2 = S_{r+1} \hookrightarrow T$$

and appealing to Lemma 3. Note that κ'_r is natural as it is the composition of natural transformations. We must also check that $\kappa'_{r+1} \circ i_r = \kappa'_r$. To do this, we have to check that the following diagram commutes

$$\begin{tikzcd} S_{r+1} \arrow[r, "\iota_{S_{r+1}}"] & S_{r+2} \arrow[r, "\hookrightarrow"] & T \\ S_r \arrow[u, "i_r"] \arrow[r, "\iota_{S_r}"'] & S_{r+1} \arrow[u, "i_{r+1}"] \arrow[r, "\hookrightarrow"'] & T \arrow[u, equal] \end{tikzcd}$$

The right hand square commutes trivially. The left commutes by applying the fact (deduced above) that $i_{r+1} = id_{\mathbb{V}} + i_r + i_r^2$. We can also define κ'' by applying Lemma 3, but the definition requires a little care. Write $S'_r \overset{\text{def}}{=} S_r^2$. Consider the family of morphisms

$$\kappa''_r\colon S'_r = S_r^2 \xrightarrow{\iota_{S_r^2}} \mathbb{V} + S_r + S_r^2 = S_{r+1} \hookrightarrow T$$

We can check that the κ''_r satisfy the conditions of Lemma 3, and so they define a morphism $\kappa''\colon U \to T$ where $U \overset{\text{def}}{=} \cup_r S'_r$. But note that (why!?)

$$Un = \cup_r S'_r n = \cup_r (S_r n)^2 = (\cup_r S_r n)^2 = (Tn)^2 = T^2 n$$

and one can also check that $U\rho = T^2\rho$. Hence $U = T^2$, and we have our definition of κ''. Thus σ_T is defined in $\mathcal{F}$, as this category has ternary coproducts.

We verify that $\sigma_T\colon \Sigma_{\mathbb{V}} T \to T$ is an initial algebra. Consider

$$\begin{tikzcd}
\mathbb{V} + T + T^2 \arrow[r, "\sigma_T"] \arrow[d, "\mathbb{V} + \overline{\alpha} + \overline{\alpha}^2"'] \arrow[dr, phantom, "(*)"] & T \arrow[d, "\overline{\alpha}"] \\
\mathbb{V} + A + A^2 \arrow[r, "\alpha"] & A
\end{tikzcd}$$

To define $\overline{\alpha}\colon T \to A$ we specify a family of natural transformations $\overline{\alpha}_r\colon S_r \to A$ and appeal to Lemma 3. Note that $\overline{\alpha}_0\colon \varnothing \to A$ and thus we define $\overline{\alpha}_0$ to be the natural transformation with components the empty functions $\varnothing\colon \varnothing \to An$ for each n in $\mathbb{F}$. Note that

$$\overline{\alpha}_{r+1}\colon S_{r+1} = \mathbb{V} + S_r + S_r^2 \to A$$

and hence we can recursively define $\overline{\alpha}_{r+1} \overset{\text{def}}{=} [\alpha \circ \iota_{\mathbb{V}}, \alpha \circ \iota_A \circ \overline{\alpha}_r, \alpha \circ \iota_{A^2} \circ \overline{\alpha}_r^2]$. It follows very simply, by induction, that the $\overline{\alpha}_r$ are natural; if $\overline{\alpha}_r$ is natural, then so too is $\overline{\alpha}_{r+1}$, it being the cotupling of compositions of natural transformations. It is an *Exercise* to verify by induction that the conditions of Lemma 3 hold, that is, $\overline{\alpha}_{r+1} \circ i_r = \overline{\alpha}_r$ for all $r \geq 0$.

Using the universal property of finite coproducts, proving that the diagram $(*)$ commutes is equivalent to proving

$$[\overline{\alpha} \circ \kappa, \overline{\alpha} \circ \kappa', \overline{\alpha} \circ \kappa'] = [\alpha \circ \iota_{\mathbb{V}}, \alpha \circ \iota_A \circ \overline{\alpha}, \alpha \ \circ \iota_{A^2} \circ \overline{\alpha}^2]$$

which in turn is equivalent to proving that the respective components of the cotuples are equal. We prove that $\overline{\alpha} \circ \kappa' = \alpha \circ \iota_A \circ \overline{\alpha}\colon T \to A$. This amounts to proving that $\overline{\alpha}_n \circ \kappa'_n = \alpha_n \circ \iota_{An} \circ \overline{\alpha}_n\colon Tn \to An$ in $\mathcal{S}et$ for any n in $\mathbb{F}$. Suppose that ξ is an arbitrary element of Tn, where $\xi \in S_r n$. Then we have

$$\begin{aligned}
\overline{\alpha}_n(\kappa'_n(\xi)) &= \overline{\alpha}_n(\iota_{S_r n}(\xi)) \\
&= (\overline{\alpha}_{r+1})_n(\iota_{S_r n}(\xi)) \\
&= [\alpha_n \circ \iota_{\mathbb{V}n}, \alpha_n \circ \iota_{An} \circ (\overline{\alpha}_r)_n, \alpha_n \circ \iota_{(An)^2} \circ (\overline{\alpha}_r)_n^2](\iota_{S_r n}(\xi)) \\
&= \alpha_n(\iota_{An}((\overline{\alpha}_r)_n(\xi))) \\
&= \alpha_n(\iota_{An}(\overline{\alpha}_n(\xi)))
\end{aligned}$$

Each step follows by applying an appropriate function definition.

Step 3 Suppose that $\rho\colon n \to n'$ is any function. We define

$$\mathit{Exp}_{\mathsf{d\bar{b}}}\, n \overset{\text{def}}{=} \{\, e \mid \Gamma^n \vdash^{\mathsf{d\bar{b}}} e \,\}$$

For any expression e there is another expression denoted by $(\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)e$ which is, informally, the expression e in which any occurrence of the variable v^i is replaced by $v^{\rho(i)}$. We can define formally the expression $(\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)e$ by recursion over e, by setting

- $(\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)(\mathsf{V}\ v^i) \overset{\text{def}}{=} \mathsf{V}\ \rho i$
- $(\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)(\mathsf{S}\ e) \overset{\text{def}}{=} \mathsf{S}\ (\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)e$
- $(\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)(\mathsf{A}\ e\ e') \overset{\text{def}}{=} \mathsf{A}\ (\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)e\ (\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)e'$

Further, one can show that if $e \in \mathit{Exp}_{\mathsf{d\bar{b}}}\, n$, then $(\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho)e \in \mathit{Exp}_{\mathsf{d\bar{b}}}\, n'$. Thus we have a function $\mathit{Exp}_{\mathsf{d\bar{b}}}\, \rho\colon \mathit{Exp}_{\mathsf{d\bar{b}}}\, n \to \mathit{Exp}_{\mathsf{d\bar{b}}}\, n'$ for any $\rho\colon n \to n'$. It is an *Exercise* to verify that these definitions yield a functor $\mathit{Exp}_{\mathsf{d\bar{b}}}\colon \mathbb{F} \to \mathcal{S}et$. Further, note that there are natural transformations (*Exercise*) $\mathsf{S}\colon \mathit{Exp}_{\mathsf{d\bar{b}}} \to \mathit{Exp}_{\mathsf{d\bar{b}}}$ and $\mathsf{A}\colon {\mathit{Exp}_{\mathsf{d\bar{b}}}}^2 \to \mathit{Exp}_{\mathsf{d\bar{b}}}$ whose obvious definitions are omitted.

Step 4 We have constructed an initial algebra $\sigma_T\colon \Sigma_{\mathbb{V}} T \to T$ in the category $\mathcal{F}$. We now show that the presheaf algebra T is isomorphic to the presheaf $\mathit{Exp}_{\mathsf{d\bar{b}}}$. To do this we need natural transformations $\phi\colon T \to \mathit{Exp}_{\mathsf{d\bar{b}}}$ and $\psi\colon \mathit{Exp}_{\mathsf{d\bar{b}}} \to T$, such that for any n in $\mathbb{F}$, the functions ϕ_n and ψ_n give rise to a bijection between Tn and $\mathit{Exp}_{\mathsf{d\bar{b}}}\, n$.

To specify $\phi\colon T \to \mathit{Exp}_{\mathsf{d\bar{b}}}$ we define a family of natural transformations $\phi_r\colon S_r \to \mathit{Exp}_{\mathsf{d\bar{b}}}$, and appeal to Lemma 3.

- $\phi_0\colon S_0 = \varnothing \to \mathit{Exp}_{\mathsf{d\bar{b}}}$ has the empty function as components $(\phi_0)_n\colon \varnothing \to \mathit{Exp}_{\mathsf{d\bar{b}}}\, n$
- Recursively we define

$$\phi_{r+1} \overset{\text{def}}{=} [\mathsf{V}, \mathsf{S} \circ \phi_r, \mathsf{A} \circ \phi_r^2]\colon S_{r+1} = \mathbb{V} + S_r + S_r^2 \to \mathit{Exp}_{\mathsf{d\bar{b}}}$$

Note that ϕ_0 is obviously natural, and that ϕ_{r+1} is natural if ϕ_r is, because $\mathcal{F}$ has ternary coproducts. It is an *Exercise* to verify the conditions of the lemma.

To specify $\psi\colon \mathit{Exp}_{\mathsf{d\bar{b}}} \to T$, for any n in $\mathbb{F}$ we define functions $\psi_n\colon \mathit{Exp}_{\mathsf{d\bar{b}}}\, n \to Tn$ as follows. First note that $S_r n \subset Tn$ for any r by definition of T. Then we define

- $\psi_n(\mathsf{V}\ v^i) \stackrel{\text{def}}{=} (v^i, 1) \in S_1 n$
- $\psi_n(\mathsf{S}\ e) \stackrel{\text{def}}{=} \iota_{S_r n}(\psi_n(e))$ where $r \geq 1$ is the height of the deduction of $\mathsf{S}\ e$
- $\psi_n(\mathsf{A}\ e\ e') \stackrel{\text{def}}{=} \iota_{(S_r n)^2}((\psi_n(e), \psi_n(e')))$ where $r \geq 1$ is the height of the deduction of $\mathsf{A}\ e\ e'$.

We next check that for any n in $\mathbb{F}$,

$$\begin{tikzcd} Tn \arrow[r, shift left, "\phi_n"] \arrow[r, phantom, "\cong"] & Exp_{\mathsf{d\bar{B}}}\, n \arrow[l, shift left, "\psi_n"] \end{tikzcd}$$

We need the lemma

Lemma 5. *For any $r \geq 0$, and $\xi \in S_{r+1}n$, the expression $(\phi_{r+1})_n(\xi)$ has a deduction of height r.*

Proof. By induction on r.

Suppose that $\xi \in S_r n \subset Tn$ for some r. Then by definition, $\psi_n(\phi_n(\xi)) = \psi_n((\phi_r)_n(\xi))$. We show by induction that for all $r \geq 0$, if ξ is any element of $S_r n$ and n any object of $\mathbb{F}$, then $\psi_n((\phi_r)_n(\xi)) = \xi$. For $r = 0$ the assertion is vacuously true, as $S_0 n$ is always empty. We assume the result holds for any $r \geq 0$. Let $\xi \in S_{r+1}n = \mathbb{V}n + S_r n + S_r n^2$. Then we have

$$\psi_n((\phi_{r+1})_n(\xi)) = \psi_n([\mathsf{V}_n, \mathsf{S}_n \circ (\phi_r)_n, \mathsf{A}_n \circ (\phi_r)^2_n](\xi))$$

We can complete the proof by case analysis. The situation $r = 0$ requires a little care, but we leave it as an *Exercise.* We just consider the case when $\xi = \iota_{S_r n}(\xi')$ for some $\xi' \in S_r n$ (which implies that $r \geq 1$). We have

$$\begin{aligned} \psi_n((\phi_{r+1})_n(\xi)) &= \psi_n((\mathsf{S}_n \circ (\phi_r)_n)(\xi')) && (1) \\ &= \psi_n(\mathsf{S}\ (\phi_r)_n(\xi')) && (2) \\ &= \iota_{S_r n}(\psi_n((\phi_r)_n(\xi'))) && (3) \\ &= \iota_{S_r n}(\xi') && (4) \\ &= \xi && (5) \end{aligned}$$

where equation 3 follows from Lemma 5, and equation 4 by induction. It is an *Exercise* to show that ϕ_n is a left inverse for ψ_n. By appeal to Lemma 1 we are done, and step 4 is complete.

However, by way of illustration, we can check directly by hand that ψ is natural, that is for any $\rho\colon n \to n'$ the diagram

$$\begin{tikzcd}
Exp_{\mathsf{d\bar{b}}}\, n \arrow[r, "\psi_n"] \arrow[d, "Exp_{\mathsf{d\bar{b}}}\, \rho"'] & Tn \arrow[d, "T\rho"] \\
Exp_{\mathsf{d\bar{b}}}\, n' \arrow[r, "\psi'_n"'] & Tn'
\end{tikzcd}$$

commutes. We must prove that for all $\Gamma^n \vdash^{\mathsf{d\bar{b}}} e$, we have

$$T\rho(\psi_n(e)) = \psi_{n'}((Exp_{\mathsf{d\bar{b}}}\, \rho)(e))$$

We consider only the inductive case of $\mathsf{S}\ e$:

$$\begin{aligned}
T\rho(\psi_n(\mathsf{S}\ e)) &= T\rho(\iota_{S_r n}(\psi_n(e))) \\
&= S_{r+1}\rho(\iota_{S_r n}(\psi_n(e))) \\
&= (\mathbb{V}\rho + S_r\rho + (S_r\rho)^2)((\iota_{S_r n}(\psi_n(e)))) \\
&= \iota_{S_r n}((S_r\rho)(\psi_n(e))) \\
&= \iota_{S_r n}((T\rho)(\psi_n(e))) \\
&= \iota_{S_r n'}((\psi'_n(Exp_{\mathsf{d\bar{b}}}\, \rho(e)))) \\
&= \psi_{n'}(\mathsf{S}\ (Exp_{\mathsf{d\bar{b}}}\, \rho)(e)) \\
&= \psi_{n'}(Exp_{\mathsf{d\bar{b}}}\, \rho(\mathsf{S}\ e))
\end{aligned}$$

It is an *Exercise* to check this calculation; the sixth equality follows by induction.

4.2 A Model of Syntax with Distinguished Variables and with Binding

Step 1 We define a functor which "corresponds" to the signature of Section 2.2. This is $\Sigma_{\mathbb{V}}\colon \mathcal{F} \to \mathcal{F}$ where $\Sigma_{\mathbb{V}}\xi \overset{\text{def}}{=} \mathbb{V} + \delta\ \xi + \xi^2$ and ξ is either an object or a morphism. The functor $\delta\colon \mathcal{F} \to \mathcal{F}$ was defined on page 18.

Step 2 We can show that the functor $\Sigma_{\mathbb{V}}$ has an initial algebra, which we denote by $\sigma_T\colon \Sigma_{\mathbb{V}} T \to T$, by adapting the methods given in Section 4.1. In fact there is very little change in the details, so we just sketch the general approach and leave the technicalities as an *Exercise*.

We define a family of presheaves $(S_r \mid r \geq 0)$, such that we may apply Lemma 2. We set $S_0 \stackrel{\text{def}}{=} \varnothing$ which is the empty presheaf, and set $S_{r+1} \stackrel{\text{def}}{=} \Sigma_{\mathbb{V}} S_r$. We must check that the conditions of Lemma 2 hold, that is, $i_r \colon S_r \hookrightarrow S_{r+1}$ for all $r \geq 0$. This can be done by induction over r, and is left as an *exercise*. We can then define $T \stackrel{\text{def}}{=} \bigcup_r S_r$ in $\mathcal{F}$.

Next we consider the structure map σ_T. This natural transformation in $\mathcal{F}$ has source and target $\mathbb{V} + \delta\ T + T^2 \to T$ and so it is given by $\sigma_T \stackrel{\text{def}}{=} [\kappa, \kappa', \kappa'']$, the cotupling of (insertion) natural transformations

$$\begin{aligned} \kappa &: \mathbb{V} \to T \\ \kappa' &: \delta\ T \to T \\ \kappa'' &: T^2 \to T \end{aligned}$$

The definitions of κ and κ'' are the same as in Section 4.1. Note that $(\delta\ T)n \stackrel{\text{def}}{=} T(n+1) = \bigcup_r S_r(n+1) = \bigcup_r (\delta\ S_r)n = (\bigcup_r \delta\ S_r)n$. In fact we can easily check that $\delta\ T = \cup_r \delta\ S_r$, as similar equalities hold if we replace n by any ρ. Hence we can apply an instance of Lemma 3 to give a definition of κ' by specifying the family of morphisms κ'_r with the following definition

$$\kappa'_r \colon \delta\ S_r \xrightarrow{\iota_{S_r}} \mathbb{V} + \delta\ S_r + S_r^2 = S_{r+1} \hookrightarrow T$$

Note that we must check to see that $\delta\ S_r \hookrightarrow \delta\ S_{r+1}$ for all r. Use induction to verify this as an *Exercise*; note that for any $F' \hookrightarrow F$ in $\mathcal{F}$, we have $\delta\ F' \hookrightarrow \delta\ F$.

We must verify that $\sigma_T \colon \Sigma_{\mathbb{V}} T \to T$ is an initial algebra. Consider

$$\begin{tikzcd} \mathbb{V} + \delta\ T + T^2 \arrow[r, "\sigma_T"] \arrow[d, "\mathbb{V} + \delta\ \overline{\alpha} + \overline{\alpha}^2"'] \arrow[dr, phantom, "(*)"] & T \arrow[d, "\overline{\alpha}"] \\ \mathbb{V} + \delta\ A + A^2 \arrow[r, "\alpha"'] & A \end{tikzcd}$$

To define $\overline{\alpha} \colon T \to A$ we specify a family of natural transformations $\overline{\alpha}_r \colon S_r \to A$ and appeal to Lemma 3. We define $\overline{\alpha}_0$ to be the natural transformation with components the empty functions $\varnothing \colon \varnothing \to An$ for each n in $\mathbb{F}$, and

$$\overline{\alpha}_{r+1} \stackrel{\text{def}}{=} [\alpha \circ \iota_{\mathbb{V}}, \alpha \circ \iota_A \circ \delta\ \overline{\alpha}_r, \alpha \circ \iota_{A^2} \circ \overline{\alpha}_r^2] \colon S_{r+1} = \mathbb{V} + S_r + S_r^2 \to A$$

The verification that $(*)$ commutes is technically identical to the analogous situation in Section 4.1, and the details are left as an *Exercise*.

Step 3 Suppose that $\rho: n \to n'$. We write $Exp_{db}\ n$ for the set $\{\ e \mid \Gamma^n \vdash^{db} e\ \}$ of expressions deduced using the rules in Figure 2. Let $\rho\{n'/n\}: n+1 \to n'+1$ be the function

$$\rho\{n'/n\}(j) \stackrel{\text{def}}{=} \begin{cases} j & \text{if} \quad 0 \le j \le n-1 \\ n' & \text{if} \quad j = n \end{cases}$$

Consider the following (syntactic) definitions

- $(Exp_{db}\ \rho)(\mathsf{V}\ v^i) \stackrel{\text{def}}{=} \mathsf{V}\ v^{\rho i}$
- $(Exp_{db}\ \rho)(\mathsf{L}\ v^n\ e) \stackrel{\text{def}}{=} \mathsf{L}\ v^{n'}\ (Exp_{db}\ \rho\{n'/n\})(e)$ and
- $(Exp_{db}\ \rho)(\mathsf{E}\ e\ e') \stackrel{\text{def}}{=} \mathsf{E}\ ((Exp_{db}\ \rho)e)\ ((Exp_{db}\ \rho)e')$

One can prove by rule induction that if $\Gamma^n \vdash^{db} e$ and $\rho: n \to n'$, then $\Gamma^{n'} \vdash^{db} (Exp_{db}\ \rho)e$. In fact we have a functor Exp_{db} in $\mathcal{F}$ and the details are an *Exercise.* Note also that there are natural transformations $\mathsf{L}: \delta\ Exp_{db} \to Exp$ and $\mathsf{E}: Exp^2 \to Exp$, *provided certain assumptions are made about the category* $\mathbb{F}$!! The latter's definition is the obvious one. For the former, the components are functions $\mathsf{L}_n: Exp_{db}\ (n+1) \to Exp_{db}\ n$ defined by $e \mapsto \mathsf{L}\ v^n\ e$. Naturality is the requirement that for any $\rho: n \to n'$ in $\mathbb{F}$, the diagram below commutes

$$\begin{tikzcd}
(\delta\ Exp_{db})n = Exp_{db}\ (n+1) \arrow[r, "\mathsf{L}_n"] \arrow[d, "(\delta\ Exp_{db})\rho = Exp_{db}\ (\rho + id_1)"'] & Exp_{db}\ n \arrow[d, "Exp_{db}\ \rho"] \\
(\delta\ Exp_{db})n' = Exp_{db}\ (n'+1) \arrow[r, "\mathsf{L}_{n'}"'] & Exp_{db}\ n'
\end{tikzcd}$$

Note that at the element e, this requires that

$$\mathsf{L}\ v^{n'}\ (Exp_{db}\ \rho\{n'/n\})e = \mathsf{L}\ v^{n'}\ ((Exp_{db}\ (\rho + id_1))e)$$

and by considering when e is a variable, we conclude that this equality holds if and only if

$$\rho\{n'/n\} = \rho + id_1$$

which is true only if in $\mathbb{F}$ the coproduct insertion $\iota_1: 1 \to m+1$ maps $*$ to m, and $\iota_m: m \to m+1$ maps i to ρi for any $i \in m$.

Step 4 We now show that the presheaf algebra T is isomorphic to the presheaf Exp_{db}. We have to show that there are natural transformations $\phi: T \to Exp_{db}$

and $\psi\colon Exp_{\mathsf{db}} \to T$, such that for any n in $\mathbb{F}$, the functions ϕ_n and ψ_n give rise to a bijection between Tn and $Exp_{\mathsf{db}}\, n$.

To specify $\phi\colon T \to Exp_{\mathsf{db}}$ we define a family of natural transformations $\phi_r\colon S_r \to Exp_{\mathsf{db}}$, and appeal to Lemma 3.

- $\phi_0\colon S_0 = \varnothing \to Exp_{\mathsf{db}}$ has as components the empty function, and
- recursively we define

$$\phi_{r+1} \stackrel{\text{def}}{=} [\mathsf{V}, \mathsf{L} \circ \delta\, \phi_r, \mathsf{E} \circ \phi_r^2]\colon S_{r+1} = \mathbb{V} + \delta\, S_r + S_r^2 \to Exp_{\mathsf{db}}$$

To specify $\psi\colon Exp_{\mathsf{db}} \to T$, for any n in $\mathbb{F}$ we define functions $\psi_n\colon Exp_{\mathsf{db}}\, n \to Tn$ as follows. First note that $S_r n \subset Tn$ for any r by definition of T. Then we define

- $\psi_n(\mathsf{V}\ v^i) \stackrel{\text{def}}{=} (v^i, 1) \in S_1 n$.
- $\psi_n(\mathsf{L}\ v^n\ e) \stackrel{\text{def}}{=} \iota_{S_r(n+1)}(\psi_{n+1}(e))$ where $r \geq 0$ is the height of the deduction of $\mathsf{L}\ v^n\ e$.
- $\psi_n(\mathsf{E}\ e\ e') \stackrel{\text{def}}{=} \iota_{(S_r n)^2}((\psi_n(e), \psi_n(e')))$ where $r \geq 0$ is the height of the deduction of $\mathsf{E}\ e\ e'$.

We check also that for any n in $\mathbb{F}$,

$$\begin{tikzcd} Tn \arrow[r, shift left, "\phi_n"] \arrow[r, phantom, "\cong"] & Exp_{\mathsf{db}}\, n \arrow[l, shift left, "\psi_n"] \end{tikzcd}$$

Suppose that $\xi \in S_r n \subset Tn$. Then by definition, $\psi_n(\phi_n(\xi)) = \psi_n((\phi_r)_n(\xi))$. We show by induction that for all $r \geq 0$, if ξ is any element in level r and n any object of $\mathbb{F}$, then $\psi_n((\phi_r)_n(\xi)) = \xi$. For $r = 0$ the assertion is vacuously true, as $S_0 n$ is always empty. We assume the result holds for any $r \geq 0$. Let $\xi \in S_{r+1}n = \mathbb{V}n + S_r(n+1) + S_r n^2$. Then we have

$$\psi_n((\phi_{r+1})_n(\xi)) = \psi_n([\mathsf{V}_n, \mathsf{L}_n \circ (\phi_r)_{n+1}, \mathsf{E}_n \circ (\phi_r)^2_n](\xi))$$

We can complete the proof by analyzing the cases which arise depending on which component ξ lives in. We just consider the case when $\xi = \iota_{S_r(n+1)}(\xi')$ for some $\xi' \in S_r(n+1)$. We have

$$\psi_n((\phi_{r+1})_n(\xi)) = \psi_n((\mathsf{L}_n \circ (\phi_r)_{n+1})(\xi')) \quad (6)$$

$$= \psi_n(\mathsf{L}\ v^n\ (\phi_r)_{n+1}(\xi')) \quad (7)$$

$$= \iota_{S_r(n+1)n}(\psi_{n+1}((\phi_r)_{n+1}(\xi'))) \tag{8}$$
$$= \iota_{S_r(n+1)n}(\xi') \tag{9}$$
$$= \xi \tag{10}$$
$$\tag{11}$$

where equation 8 follows from Lemma 5, and equation 9 by induction. It is an *Exercise* to show that ϕ_n is a left inverse for ψ_n, and we are done appealing to Lemma 1.

By way of illustration, we give a sample of the direct calculation that each ψ_n is natural. We prove that for all $\Gamma^n \vdash^{\mathsf{db}} e$, we have

$$T\rho(\psi_n(e)) = \psi_{n'}((\mathit{Exp}_{\mathsf{db}}\ \rho)(e))$$

We consider the case of $\mathsf{L}\ v^n\ e$; checking the details is an *Exercise*.

$$\begin{aligned}
T\rho(\psi_n(\mathsf{L}\ v^n\ e)) &= T\rho(\iota_{S_r(n+1)}(\psi_{n+1}(e))) \\
&= S_{r+1}\rho(\iota_{S_r(n+1)}(\psi_{n+1}(e))) \\
&= ((\mathbb{V}\rho + (\delta\ S_r)(\rho) + (S_r\rho)^2)\rho)(\iota_{S_r(n+1)}(\psi_{n+1}(e))) \\
&= ((\mathbb{V}\rho + S_r(\rho + id_1) + (S_r\rho)^2)\rho)(\iota_{S_r(n+1)}(\psi_{n+1}(e))) \\
&= \iota_{S_r(n'+1)}((S_r(\rho + id_1))(\psi_{n+1}(e))) \\
&= \iota_{S_r(n'+1)}((T(\rho + id_1))(\psi_{n+1}(e))) \\
&= \iota_{S_r(n'+1)}((\psi_{n'+1}(\mathit{Exp}_{\mathsf{db}}\ (\rho + id_1)(e))) \\
&= \iota_{S_r(n'+1)}(\psi_{n'+1}((\mathit{Exp}_{\mathsf{db}}\ \rho\{n'/n\})e)))) \\
&= \psi_{n'}(\mathsf{L}\ v^{n'}\ (\mathit{Exp}_{\mathsf{db}}\ \rho\{n'/n\})(e)) \\
&= \psi_{n'}((\mathit{Exp}_{\mathsf{db}}\ \rho)(\mathsf{L}\ v^n\ e))
\end{aligned}$$

4.3 A Model of Syntax with Arbitrary Variables and Binding

We define a functor which "corresponds" to the signature of Section 2.3. The functor $\Sigma_{\mathbb{V}}: \mathcal{F} \to \mathcal{F}$ is defined by setting $\Sigma_{\mathbb{V}}\xi \overset{\text{def}}{=} \mathbb{V} + \delta\ \xi + \xi^2$. As you see, it is identical to the functor given at the start of Section 4.2, and thus has an initial algebra $\sigma_T: \Sigma_{\mathbb{V}} T \to T$.

We show in this section that we can define a functor $\mathit{Exp}_{\mathsf{ab}}$ in $\mathcal{F}$ which captures the essence of the inductive system of expressions given in Section 2.3 and is such that $\mathit{Exp}_{\mathsf{ab}} \cong T$. We could prove this by proceeding (directly) as we did in Section 4, undertaking the steps 2 to 4 of page 30. However, it is in fact easier, and more instructive, to first define $\mathit{Exp}_{\mathsf{ab}}$, step 3, and then prove that $\mathit{Exp}_{\mathsf{ab}} \cong \mathit{Exp}_{\mathsf{db}}$. Given previous results, this gives us steps 2 and 4.

We will need two lemmas which yield *admissible* rules (see Appendix). The rules cannot be derived.

Lemma 6. *Suppose that* $\Delta \vdash^{\mathsf{ab}} e$*, and that* Δ' *is also an environment. Then* $\Delta' \vdash^{\mathsf{ab}} e\{\Delta'/\Delta\}$.

Proof. We prove by rule induction

$$(\forall \Delta \vdash^{\mathsf{ab}} e) \quad (\forall \Delta')\ (\Delta' \vdash^{\mathsf{ab}} e\{\Delta'/\Delta\})$$

We prove property closure for the rule introducing abstractions $\mathsf{L}\ x\ e$. Suppose that $\Delta \vdash^{\mathsf{ab}} \mathsf{L}\ x\ e$. Then $\Delta, x \vdash^{\mathsf{ab}} e$. Pick any Δ'. We try to prove that

$$\Delta' \vdash^{\mathsf{ab}} (\mathsf{L}\ x\ e)\{\Delta'/\Delta\}$$

We consider only the case when $x \in \Delta'$ at position p, and $y \stackrel{\text{def}}{=} \mathsf{el}_p(\Delta) \in \mathit{fv}(e)$. We must then prove

$$\Delta' \vdash^{\mathsf{ab}} \mathsf{L}\ x'\ e\{\Delta', x'/\Delta, x\} \qquad *$$

which is well-defined as $x \notin \Delta$ and $x' \notin \Delta'$. By induction, we have $\Delta', x' \vdash^{\mathsf{ab}} e\{\Delta', x'/\Delta, x\}$. Hence $*$ follows.

Lemma 7. *If* $\Delta \vdash^{\mathsf{ab}} e$*, and* Δ *is a sublist of an environment* Δ_1*, then* $\Delta_1 \vdash^{\mathsf{ab}} e$.

Proof. Rule Induction. *Exercise.*

Back to the task at hand. First we must define $\mathit{Exp}_{\mathsf{ab}}$. For n in $\mathbb{F}$ we set

$$\mathit{Exp}_{\mathsf{ab}}\, n \stackrel{\text{def}}{=} \{\ [e]_\alpha \ \mid\ \Gamma^n \vdash^{\mathsf{ab}} e\ \}$$

Now let $\rho\colon n \to n'$. We define

$$(\mathit{Exp}_{\mathsf{ab}}\, \rho)([e]_\alpha) \stackrel{\text{def}}{=} [e\{v^{\rho 0}, \ldots, v^{\rho(n-1)}/v^0, \ldots, v^{n-1}\}]_\alpha$$

To check if this is a good definition, we need to show that if $\Gamma^n \vdash^{\mathsf{ab}} e$ then

$$\Gamma^{n'} \vdash^{\mathsf{ab}} e\{v^{\rho 0}, \ldots, v^{\rho(n-1)}/v^0, \ldots, v^{n-1}\}$$

This follows from Lemma 6 and Lemma 7. We must also check that if there is any e' with $e \sim_\alpha e'$, then

$$e\{v^{\rho 0}, \ldots, v^{\rho(n-1)}/v^0, \ldots, v^{n-1}\} \sim_\alpha e'\{v^{\rho 0}, \ldots, v^{\rho(n-1)}/v^0, \ldots, v^{n-1}\}$$

This is proved by rule induction for $\sim_\alpha$, a tedious *Exercise*.

We now show that $\phi\colon Exp_{\mathsf{ab}} \cong Exp_{\mathsf{db}} : \psi$. The components of ψ are functions $\psi_n\colon Exp_{\mathsf{db}}\ n \to Exp_{\mathsf{ab}}\ n$ given by $\psi_n(e) \stackrel{\text{def}}{=} [e]_\alpha$. We consider the naturality of ψ at a morphism $\rho\colon n \to n'$, computed at an element ξ of $Exp_{\mathsf{db}}\ n$. We show naturality for the case $\xi = \mathsf{L}\ v^n\ e$.

$$\begin{aligned}(Exp_{\mathsf{ab}}\ \rho) \circ \psi_n(\xi) &= (Exp_{\mathsf{ab}}\ \rho)[\mathsf{L}\ v^n\ e]_\alpha \\ &= [(\mathsf{L}\ v^n\ e)\{v^{\rho 0}, \ldots, v^{\rho(n-1)}/v^0, \ldots, v^{n-1}\}]_\alpha \\ &\stackrel{\text{def}}{=} \Box\end{aligned}$$

Let us consider the case when renaming takes place. Suppose that there is a j for which $\rho(j) = n$ and $v^j \in fv(e)$. Then[6]

$$\begin{aligned}&(\mathsf{L}\ v^n\ e)\{v^{\rho(0)}, \ldots, v^{\rho(n-1)}/v^0, \ldots, v^{n-1}\} = \\ &\qquad \mathsf{L}\ v^w\ e\{v^{\rho(0)}, \ldots v^{\rho(n-1)}, v^w/v^0, \ldots, v^{n-1}, v^n\}\end{aligned}$$

where w is 1 plus the maximum of the indices occurring freely in e and the indices $\rho(0), \ldots, \rho(n-1)$. Thus $\rho(i) < w$ for all $0 \leq i \leq n-1$. But the free variables in e must lie in $v^0, \ldots, v^n$ (why?) and moreover $n = \rho(j)$ occurs in $\rho(0), \ldots, \rho(n-1)$. Finally note that $\rho(i) < n'$, and so we must have $w \leq n'$. If $w < n'$, then $v^{n'}$ is not free in $e\{v^{\rho(0)}, \ldots v^{\rho(n-1)}, v^w/v^0, \ldots, v^{n-1}, v^n\}$. Otherwise (of course) $w = n'$. Either way (why!?),

$$\begin{aligned}&\mathsf{L}\ v^w\ e\{v^{\rho(0)}, \ldots v^{\rho(n-1)}, v^w/v^0, \ldots, v^{n-1}, v^n\} \\ &\qquad \sim_\alpha \mathsf{L}\ v^{n'}\ e\{v^{\rho(0)}, \ldots v^{\rho(n-1)}, v^{n'}/v^0, \ldots, v^{n-1}, v^n\}\end{aligned}$$

and so

$$\begin{aligned}\Box &= [\mathsf{L}\ v^{n'}\ e\{v^{\rho 0}, \ldots, v^{\rho(n-1)}, v^{n'}/v^0, \ldots, v^{n-1}, v^n\}]_\alpha \\ &= [\mathsf{L}\ v^{n'}\ (Exp_{\mathsf{db}}\ \rho\{n'/n\})e]_\alpha \\ &= \psi_{n'} \circ (Exp_{\mathsf{db}}\ \rho)(\xi)\end{aligned}$$

Next we define the functions $\phi_n\colon Exp_{\mathsf{ab}}\ n \to Exp_{\mathsf{db}}\ n$ by setting $\phi_n([e]_\alpha) \stackrel{\text{def}}{=} R^n(e)$ where

- $R^m(\mathsf{V}\ x) \stackrel{\text{def}}{=} \mathsf{V}\ x$

[6] There is no deletion.

- $R^m(\mathsf{L}\ x\ e) \overset{\text{def}}{=} \mathsf{L}\ v^m\ R^{m+1}(e\{v^m/x\})$
- $R^m(\mathsf{E}\ e\ e') \overset{\text{def}}{=} \mathsf{E}\ R^m(e)\ R^m(e')$

To be well-defined, we require $R^m(e) = R^m(e')$ for all $e \sim_\alpha e'$. We can prove this by induction over $\sim_\alpha$. The only tricky case concerns the axiom for renaming, $\mathsf{L}\ x\ e \sim_\alpha \mathsf{L}\ x'\ e\{x'/x\}$ where $x' \notin fv(e)$. We have

$$\begin{aligned} R^m(\mathsf{L}\ x'\ e\{x'/x\}) &= \mathsf{L}\ v^m\ R^{m+1}(e\{x'/x\}\{v^m/x'\}) \\ &= \mathsf{L}\ v^m\ R^{m+1}(e\{v^m/x\}) \\ &= R^m(\mathsf{L}\ x\ e) \end{aligned}$$

with the second equality holding as $x' \notin fv(e)$. Let $e \in Exp_{\mathsf{db}}\ n$. We will have $\phi_n(\psi_n(e)) = e$ provided that $R^n(e) = e$. We can prove this by rule induction, showing

$$(\forall \Gamma^n \vdash^{\mathsf{db}} e)(R^n(e) = e)$$

The details are easy and left as an *Exercise*. Let $e \in Exp_{\mathsf{ab}}\ n$. It remains to prove that $e = \psi_n(\phi_n(e))$. This will hold provided that $R^n(e) \sim_\alpha e$. We prove

$$(\forall \Delta \vdash^{\mathsf{ab}} e) \quad (\forall \Gamma^n)\ (\Delta = \Gamma^n \implies R^n(e) \sim_\alpha e)$$

We show property closure for the rule introducing abstractions. Suppose that $\Gamma^n \vdash^{\mathsf{ab}} \mathsf{L}\ x\ e$. We must show that $R^n(\mathsf{L}\ x\ e) \sim_\alpha \mathsf{L}\ x\ e$. Now $\Gamma^n, x \vdash^{\mathsf{ab}} e$ and so by a *careful* use of Lemma 6 we get $\Gamma^{n+1} \vdash^{\mathsf{ab}} e\{v^n/x\}$. Hence by induction $R^{n+1}(e\{v^n/x\}) \sim_\alpha e\{v^n/x\}$, and so

$$R^n(\mathsf{L}\ x\ e) \overset{\text{def}}{=} \mathsf{L}\ v^n\ R^{n+1}(e\{v^n/x\}) \sim_\alpha \mathsf{L}\ v^n\ e\{v^n/x\}$$

If $x = v^n$ we are done. If not, noting that $x \notin \Gamma^n$ by assumption, $v^n \notin fv(e)$. Hence $\mathsf{L}\ v^n\ e\{v^n/x\} \sim_\alpha \mathsf{L}\ x\ e$.

4.4 A Model of Syntax without Variables but with Binding

Again, we show how some syntax, this time the de Bruijn expressions of Section 2.4, can be rendered as a functor. We give only bare details, and leave most of the working to the reader. We do assume that readers are already familiar with the de Bruijn notation.

For any n in $\mathbb{F}$ we define $Exp_{\mathsf{ib}}\ n \overset{\text{def}}{=} \{\ e \mid n \vdash^{\mathsf{ib}} e\ \}$. We define for $\rho: n \to n'$ the function $Exp_{\mathsf{ib}}\ \rho$ by recursion. Consider the following (syntactic) definitions

- $(Exp_{\mathsf{ib}}\ \rho)(\mathsf{V}\ i) \overset{\text{def}}{=} \mathsf{V}\ \rho i$
- $(Exp_{\mathsf{ib}}\ \rho)(\lambda\ e) \overset{\text{def}}{=} \lambda\ (Exp_{\mathsf{ib}}\ \rho^*)(e)$ and
- $(Exp_{\mathsf{ib}}\ \rho)(\$\ e\ e') \overset{\text{def}}{=} \$\ ((Exp_{\mathsf{ib}}\ \rho)e)\ ((Exp_{\mathsf{ib}}\ \rho)e')$

where $\rho^*: n+1 \to n'+1$ and $\rho^*(0) \overset{\text{def}}{=} 0$ and $\rho^*(i+1) \overset{\text{def}}{=} \rho(i)+1$ for $0 \leq i \leq n-1$. One can prove by induction that if $n \vdash^{\mathsf{ib}} e$ then $n' \vdash^{\mathsf{ib}} (Exp_{\mathsf{ib}}\ \rho)e$, and thus $Exp_{\mathsf{ib}}\ \rho$ is well-defined.

One can prove that $Exp_{\mathsf{ib}} \cong T$ by adapting the methods of Section 4.2, establishing a natural isomorphism $\phi: T \cong Exp_{\mathsf{ib}} : \psi$. Such an isomorphism exists only for a specific choice of coproducts in $\mathbb{F}$.

To specify $\phi: T \to Exp_{\mathsf{ib}}$ we define a family of natural transformations $\phi_r: S_r \to Exp_{\mathsf{ib}}$, and appeal to Lemma 3, as follows.

- $\phi_0: S_0 = \varnothing \to Exp_{\mathsf{ib}}$ has as components the empty function, and
- recursively we define

$$\phi_{r+1} \overset{\text{def}}{=} [\mathsf{V}, \lambda \circ \delta\ \phi_r, \$ \circ \phi_r^2]: S_{r+1} = \mathbb{V} + \delta\ S_r + S_r^2 \to Exp_{\mathsf{ib}}$$

where there are natural transformations $\lambda: Exp_{\mathsf{ib}} \to Exp_{\mathsf{ib}}$ and $\$: (Exp_{\mathsf{ib}})^2 \to Exp_{\mathsf{ib}}$. Of course $\lambda_n(e) \overset{\text{def}}{=} \lambda\ e$ for any e in $Exp_{\mathsf{ib}}\ n$. This is natural only if the coproduct insertion $\iota_1: 1 \to m+1$ maps $*$ to 0, and $\iota_m: m \to m+1$ maps $i+1$ to $\rho(i)+1$ for any $0 \leq i \leq m-1$.

We leave the definition of ψ, and all other calculations, as a long(ish) *Exercise.*

4.5 Where to now?

There are a number of books which cover basic category theory. For a short and gentle introduction, see [29]. For a longer first text see [19]. Both of these books are intended for computer scientists. The original and recommended general reference for category theory is [25], which was written for mathematicians. A very concise and fast paced introduction can be found in [21] which also covers the theory of allegories (which, roughly, are to relations, what categories are to functions). Again for the more advanced reader, try [31] which is an essential read for anyone interested in categorical logic, and which has a lot of useful background information. The Handbook of Logic in Computer Science has a wealth of material which is related to categorical logic; there is a chapter [30] on category theory. Equalities such as those that arise from universal properties can often be established using so-called calculational methods. For a general introduction, and more references, see [2]. Finally we mention [32] which, apart from being a very interesting introduction to category theory due to the many and varied computing examples, has a short chapter devoted to distributive categories.

The material in these notes has its origins in the paper [6]. You will find that these notes provide much of the detail which is omitted from the first few sections of this paper. In addition, you will find an interesting abstract account of substitution, and a detailed discussion of initial algebra semantics.

The material in these notes, and in [6], is perhaps rather closely allied to the methodology of de Bruijn expressions. Indeed, in these notes, when we considered λ-calculus, we either introduced α-equivalence, or we had a system of expressions which forced a binding variable to be v^n whenever the free variables of the subexpression of an abstraction were in Γ^n. We would like to be able to define a datatype whose elements are the expressions of the λ-calculus, already identified up to α-equivalence. This is achieved by Pitts and Gabbay [7], who undertake a fundamental study of α-equivalence, and formulate a new set theory within which this is possible. For further work see [16] and the references therein.

There is a wealth of literature on so called Higher Order Abstract Syntax. This is another methodology for encoding syntax with binders. For an introduction, see [15, 14], although the ideas go back to Church. The paper [10] provides links between Higher Order Abstract Syntax, and the presheaf models described in these notes. For material on implementation, see [4, 5]. A more recent approach, which combines de Bruijn notation and ordinary λ-calculus in a *hybrid* syntax, is described in [1].

If you are interested in direct implementations of α-equivalence, see [8, 9]. See [3] for the origins of de Bruijn notation.

The equation $E \cong V + E + (E \times E)$ is a very simple example of a *domain* equation. Such equations arise frequently in the study of the semantics of programming languages. They do not always have solutions in $\mathcal{S}et$. However, many can be solved in other categories. See for example [17]. Readers should note that the lemmas given in Section 3.3 can be presented in a more general, categorical manner, which is described in *loc. cit.* In fact our so called union presheaf $\cup_r S_r$ is better described as a colimit (itself a generalization of coproduct) of the diagram

$$\ldots S_{r+1} \longrightarrow S_r \longrightarrow \ldots \longrightarrow S_1 \longrightarrow S_0$$

... but this is another story.

Acknowledgements

Thanks to Simon Ambler for useful discussions. I should emphasize again that these notes owe much to the ideas presented in Fiore, Plotkin and Turi's paper [6], and, more indirectly, to the recent work of Gabbay, Hofmann, and Pitts.

Appendix

4.6 Lists

We require the notion of a *finite list*. For the purposes of these notes, a (finite) **list** over a set S is an element of the set

$$[\,S\,] \stackrel{\text{def}}{=} \bigcup_{n<\omega} S^n$$

where S^n is the set of n-tuples of S and $S^0 \stackrel{\text{def}}{=} \{\,\epsilon\,\}$ where ϵ denotes the empty list. We denote a typical non-empty element of $[\,S\,]$ by $[\,s_1,\dots,s_n\,]$ or sometimes just $s_1,\dots,s_n$, and we write $s \in L$ to indicate that s occurs in the list (tuple) L. We write $\mathsf{len}(l)$ for the length of any list l.

4.7 Abstract Syntax Trees

We adopt the following notation for finite trees: If T_1, T_2, T_3 and so on to T_n is a (finite) sequence of finite trees, then we write $\mathsf{C}\ T_1\ T_2\ T_3\ \dots\ T_n$ for the finite tree which has the form

$$\begin{tikzcd}
 & & \mathsf{C} \arrow[dll] \arrow[dl] \arrow[d] \arrow[drr] & & \\
T_1 & T_2 & T_3 & \dots & T_n
\end{tikzcd}$$

Each T_i is itself of the form $\mathsf{C}'\ T'_1\ T'_2\ T'_3\ \dots\ T'_m$. We call C a **constructor** and say that C takes n arguments. Any constructor which takes 0 arguments is a **leaf** node. We call C the **root** node of the tree. The roots of the trees T_i are called the **children** of C. The constructors are **labels** for the *nodes* of the tree. Each of the T_i above is a **subtree** of the whole tree—in particular, any leaf node is a subtree.

If we say that A is a constructor which takes two arguments, and S and V constructors which takes one argument, then the tree in Figure 5 is denoted by $\mathsf{A}\ (\mathsf{V}\ v^2)\ (\mathsf{A}\ (\mathsf{V}\ v^2)\ \mathsf{S}\ (\mathsf{V}\ v^8))$. Note that in this (finite) tree, we regard each node as a constructor. To do this, we can think of any v^i as constructors which take no arguments!!. These form the *leaves* of the tree. We call the root of the tree the **outermost** constructor, and refer to trees of this kind as **abstract syntax** trees. We often refer to an abstract syntax tree by its outermost constructor—the tree above is an "A" expression.

$$\begin{tikzcd}
 & \mathsf{A} \arrow[dl] \arrow[dr] & & \\
\mathsf{V} \arrow[d] & & \mathsf{A} \arrow[dl] \arrow[dr] & \\
v^2 & \mathsf{V} \arrow[d] & & \mathsf{S} \arrow[d] \\
 & v^3 & & \mathsf{V} \arrow[d] \\
 & & & v^8
\end{tikzcd}$$

Fig. 5. An Abstract Syntax Tree

4.8 Inductively Defined Sets

In this section we introduce a method for defining sets. Any such set will be known as an *inductively defined set*. Let us first introduce some notation. We let U be any set. A **rule** R is a pair (H, c) where $H \subseteq U$ is any finite set, and $c \in U$ is any element. Note that H might be $\varnothing$, in which case we say that R is a **base** rule. Sometimes we refer to base rules as **axioms**. If H is non-empty we say R is an **inductive** rule. In the case that H is non-empty we might write $H = \{ h_1, \dots, h_k \}$ where $1 \leq k$. We can write down a base rule $R = (\varnothing, c)$ using the following notation

Base

$$\frac{}{c}\,(R)$$

and an inductive rule $R = (H, c) = (\{ h_1, \dots, h_k \}, c)$ as

Inductive

$$\frac{h_1 \quad h_2 \quad \dots \quad h_k}{c}\,(R)$$

Given a set U and a set $\mathcal{R}$ of rules based on U, a **deduction** is a finite tree with nodes labelled by elements of U such that

• each leaf node label c arises as a base rule $(\varnothing, c) \in \mathcal{R}$
• for any non-leaf node label c, if H is the set of children of c then $(H, c) \in \mathcal{R}$ is an inductive rule.

We then say that the set **inductively defined** by $\mathcal{R}$ consists of those elements $u \in U$ which have a deduction with root node labelled by u.

Example 1. 1. Let U be the set $\{ u_1, u_2, u_3, u_4, u_5, u_6 \}$ and let $\mathcal{R}$ be the set of rules

$$\{ R_1 = (\varnothing, u_1), R_2 = (\varnothing, u_3), R_3 = (\{ u_1, u_3 \}, u_4), R_4 = (\{ u_1, u_3, u_4 \}, u_5) \}$$

Then a deduction for u_5 is given by the tree

$$\begin{tikzcd}
 & u_5 \arrow[dl] \arrow[d] \arrow[dr] & & \\
u_1 & u_3 & u_4 \arrow[dl] \arrow[dr] & \\
 & u_1 & & u_3
\end{tikzcd}$$

which is more normally written up-side down and in the following style

$$\dfrac{\dfrac{}{u_1}\,R_1 \qquad \dfrac{}{u_3}\,R_2 \qquad \dfrac{\dfrac{}{u_1}\,R_1 \qquad \dfrac{}{u_3}\,R_2}{u_4}\,R_3}{u_5}\,R_4$$

2. A set $\mathcal{R}$ of rules for defining the set $E \subseteq \mathbb{N}$ of even numbers is $\mathcal{R} = \{ R_1, R_2 \}$ where

$$\dfrac{}{0}\,(R_1) \qquad\qquad \dfrac{e}{e+2}\,(R_2)$$

Note that rule R_2 is, strictly speaking, a rule **schema**, that is e is acting as a variable. There is a “rule” for each instantiation of e. A deduction of 6 is given by

$$\dfrac{\dfrac{\dfrac{\dfrac{}{0}\,(R_1)}{0+2}\,(R_2)}{2+2}\,(R_2)}{4+2}\,(R_2)$$

3. Let V be a set of **propositional variables**. The set of (first order) propositions *Prop* is inductively defined by the rules below. There are two distinguished (atomic) propositions true and false. Each proposition denotes a finite tree. In fact true and false are constructors with zero arguments, as is each p. The remaining logical connectives are constructors with two arguments, and are written in a sugared (infix) notation.

$$\frac{-}{v}\,[v \in V] \qquad \frac{}{\mathsf{false}} \qquad \frac{}{\mathsf{true}} \qquad \frac{\phi \quad \psi}{\phi \wedge \psi} \qquad \frac{\phi \quad \psi}{\phi \vee \psi} \qquad \frac{\phi \quad \psi}{\phi \to \psi}$$

4.9 Rule Induction

In this section we see how inductive techniques of proof which the reader has met before fit into the framework of inductively defined sets. We write $\phi(x)$ to denote a proposition about x. For example, if $\phi(x) \overset{\text{def}}{=} x \geq 2$, then $\phi(3)$ is true and $\phi(0)$ is false. If $\phi(a)$ is *true* then we often say that $\phi(a)$ **holds**.

We present in Table 6 a useful principle called **Rule Induction**. It will be used throughout the remainder of these notes.

Let I be inductively defined by a set of rules $\mathcal{R}$. Suppose we wish to show that a proposition $\phi(i)$ holds for all elements $i \in I$, that is, we wish to prove

$$\forall i \in I. \quad \boxed{\phi(i)}.$$

Then all we need to do is

- for every base rule $\frac{-}{b} \in \mathcal{R}$ prove that $\phi(b)$ holds; and
- for every inductive rule $\frac{h_1 \ldots h_k}{c} \in \mathcal{R}$ prove that whenever $h_i \in I$,

$$(\phi(h_1) \wedge \phi(h_2) \wedge \ldots \wedge \phi(h_k)) \quad \Longrightarrow \quad \phi(c)$$

We call the propositions $\phi(h_j)$ **inductive hypotheses**. We refer to carrying out the bulleted (•) tasks as "verifying **property closure**".

Fig. 6. Rule Induction

The Principle of Mathematical Induction arises as a special case of Rule Induction. We can regard the set $\mathbb{N}$ as inductively defined by the rules

$$\frac{-}{0}\,(zero) \qquad \frac{n}{n+1}\,(add1)$$

Suppose we wish to show that $\phi(n)$ holds for all $n \in \mathbb{N}$, that is $\forall n \in \mathbb{N}.\phi(n)$. According to Rule Induction, we need to verify

- property closure for *zero*, that is $\phi(0)$; and
- property closure for *add1*, that is for every natural number n, $\phi(n)$ implies $\phi(n+1)$, that is $\forall n \in \mathbb{N}.\ (\phi(n) \implies \phi(n+1))$

and this amounts to precisely what one needs to verify for Mathematical Induction.

1. Here is another example of abstract syntax trees defined inductively. Let a set of constructors be $\mathbb{Z} \cup \{+, -\}$. The integers will label leaf nodes, and $+$, $-$ will take two arguments written with an infix notation. The set of abstract syntax trees $\mathcal{T}$ inductively defined by these constructors is given by

$$\frac{}{n} \qquad \frac{T_1 \quad T_2}{T_1 + T_2} \qquad \frac{T_1 \quad T_2}{T_1 - T_2}$$

Note that the base rules correspond to leaf nodes. In the example tree

$$\begin{tikzcd}
 & & + \arrow[dl] \arrow[dr] & \\
 & - \arrow[dl] \arrow[dr] & & 2 \\
55 & & 7 &
\end{tikzcd}$$

$55 - 7$ is a subtree of $(55 - 7) + 2$, as are the leaves 55, 7 and 2.

The principle of **structural induction** is defined to be an instance of rule induction when the inductive definition is of abstract syntax trees. Make sure you understand that if $\mathcal{T}$ is an inductively defined set of syntax trees, to prove $\forall T \in \mathcal{T}.\phi(T)$ we have to prove:

- $\phi(L)$ for each leaf node L; and
- assuming $\phi(T_1)$ and ... and $\phi(T_n)$ prove $\phi(C(T_1, \ldots, T_n))$ for *each* constructor C and *all* trees $T_i \in \mathcal{T}$.

These two points are precisely property closure for base and inductive rules.

Consider the proposition $\phi(T)$ given by $L(T) = N(T) + 1$ where $L(T)$ is the number of leaves in T, and $N(T)$ is the number of $+, -$-nodes of T. We can prove by structural induction

$$\forall T \in \mathcal{T}. \quad \boxed{L(T) = N(T) + 1}$$

where the functions $L, N\colon \mathcal{T} \to \mathbb{N}$ are defined recursively by

- $L(n) = 1$ and $L(+(T_1, T_2)) = L(T_1) + L(T_2)$ and $L(-(T_1, T_2)) = L(T_1) + L(T_2)$
- $N(n) = 0$ and $N(+(T_1, T_2)) = N(T_1) + N(T_2) + 1$ and $N(-(T_1, T_2)) = N(T_1) + N(T_2) + 1$

This is left as an *exercise.*

Sometimes it is convenient to add a rule R to a set $\mathcal{R}$, which does not alter the resulting set I. We say that a rule

$$\frac{h_1 \quad \ldots \quad h_k}{c}\, R$$

is a **derived** rule of $\mathcal{R}$ if there is a deduction tree whose leaves are either the conclusions of base rules or are instances of the h_i, and the conclusion is c.

The rule R is called **admissible** if one can prove

$$(h_1 \in I) \wedge \ldots \wedge (h_k \in I) \implies (c \in I)$$

Proposition 1. *Let I be inductively defined by $\mathcal{R}$, and suppose that R is a derived rule. Then the set I' inductively defined by $\mathcal{R} \cup \{ R \}$ is also I. Any derived rule is admissible.*

Proof. It is clear that $I \subset I'$. It is an exercise in rule induction to prove that $I' \subset I$. Verify property closure for each of the rules in $\mathcal{R} \cup \{ R \}$, the property $\phi(i) \overset{\text{def}}{=} i \in I$. It is clear that derived rules are admissible.

4.10 Recursively Defined Functions

Let I be inductively defined by a set of rules $\mathcal{R}$, and A any set. A function $f\colon I \to A$ can be defined by

• specifying an element $f(b) \in A$ for every base rule $\frac{}{b} \in \mathcal{R}$; and
• specifying $f(c) \in A$ in terms of $f(h_1) \in A$ and $f(h_2) \in A$ .... and $f(h_k) \in A$ for every inductive rule $\frac{h_1 \ldots, h_k}{c} \in \mathcal{R}$,

provided that each instance of a rule in $\mathcal{R}$ introduces a different element of I—why do we need this condition? When a function is defined in this way, it is said to be **recursively** defined.

Example 2. 1. The factorial function $F\colon \mathbb{N} \to \mathbb{N}$ is usually defined recursively. We set

- $F(0) \overset{\text{def}}{=} 1$ and
- $\forall n \in \mathbb{N}. F(n+1) \overset{\text{def}}{=} (n+1) * F(n)$.

Thus $F(3) = (2+1) * F(2) = 3 * 2 * F(1) = 3 * 2 * 1 * F(0) = 3 * 2 * 1 * 1 = 6$. Are there are brackets missing from the previous calculation? If so, insert them.

2. Consider the propositions defined on page 50. Suppose that ψ_i and x_i are propositions and propositional variables for $1 \leq i \leq n$. Then there is a recursively defined function *Prop* $\to$ *Prop* whose action is written $\phi \mapsto \phi\{\psi_1, \ldots, \psi_n / x_1, \ldots, x_n\}$ which computes the *simultaneous substitution* of the ϕ_i for the x_i where the x_i *are distinct*. We set

- $x\{\psi_1, \ldots, \psi_n / x_1, \ldots, x_n\} \overset{\text{def}}{=} \psi_j$ if x is x_j;
- $x\{\psi_1, \ldots, \psi_n / x_1, \ldots, x_n\} \overset{\text{def}}{=} x$ if x is none of the x_i;
- $$\begin{aligned}&(\phi \wedge \phi')\{\psi_1, \ldots, \psi_n / x_1, \ldots, x_n\} \overset{\text{def}}{=} \\ &\qquad (\phi\{\psi_1, \ldots, \psi_n / x_1, \ldots, x_n\}) \wedge (\phi'\{\psi_1, \ldots, \psi_n / x_1, \ldots, x_n\})\end{aligned}$$
- The other clauses are similar.

References

1. S. J. Ambler and R. L. Crole and A. Momigliano. Combining Higher Order Abstract Syntax with Tactical Theorem Proving and (Co)Induction. Accepted for the 15th International Conference on Theorem Proving in Higher Order Logics, 20-23 August, Virginia, U.S.A. To appear in LNCS.
2. R. Backhouse and R. L. Crole and J. Gibbons. Algebraic and Coalgebraic Methods in the Mathematics of Program Construction. Revised lectures from an International Summer School and Workshop, Oxford, UK, April 2000. Springer Verlag Lecture Notes in Computer Science 2297, 2002.
3. N. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.
4. J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 124–138, Edinburgh, Scotland, Apr. 1995. Springer-Verlag LNCS 902.
5. J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163, Nancy, France, Apr. 1997. Springer-Verlag LNCS.
6. M. Fiore and G. D. Plotkin and D. Turi. Abstract Syntax and Variable Binding. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 193–202, Trento, Italy, 1999. IEEE Computer Society Press.
7. M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 214–224, Trento, Italy, 1999. IEEE Computer Society Press.
8. A. Gordon. A mechanisation of name-carrying syntax up to alpha-conversion. In J.J. Joyce and C.-J.H. Seger, editors, *International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume 780 of *Lecture Notes in Computer Science*, pages 414–427, Vancouver, Canada, Aug. 1993. University of British Columbia, Springer-Verlag, published 1994.
9. A. D. Gordon and T. Melham. Five axioms of alpha-conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'96)*, volume 1125 of *Lecture Notes in Computer Science*, pages 173–190, Turku, Finland, August 1996. Springer-Verlag.
10. M. Hofmann. Semantical analysis for higher-order abstract syntax. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 204–213, Trento, Italy, July 1999. IEEE Computer Society Press.
11. F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on systems in higher-order abstract syntax. In *Proc. ICALP'01*, number 2076 in LNCS, pages 963–978. Springer-Verlag, 2001.
12. R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transaction in Computational Logic*, 2001. To appear.
13. J. McKinna and R. Pollack. Some Type Theory and Lambda Calculus Formalised. To appear in Journal of Automated Reasoning, Special Issue on Formalised Mathematical Theories (F. Pfenning, Ed.),
14. F. Pfenning. Computation and deduction. Lecture notes, 277 pp. Revised 1994, 1996, to be published by Cambridge University Press, 1992.

15. F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1988.
16. A. M. Pitts. Nominal logic: A first order theory of names and binding. In N. Kobayashi and B. C. Pierce, editors, *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29-31, 2001, Proceedings*, volume 2215 of *Lecture Notes in Computer Science*, pages 219–242. Springer-Verlag, Berlin, 2001.
17. M.B. Smyth and G.D. Plotkin. The Category Theoretic Solution of Recursive Domain Equations. In *SIAM Journal of Computing*, 1982, volume 11(4), pages 761–783.
M.B. Smyth and G.D. Plotkin. The Category Theoretic Solution of Recursive Domain Equations. In *SIAM Journal of Computing*, 1982, volume 11(4), pages 761–783.
18. M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer-Verlag, 1985.
19. M. Barr and C. Wells. *Category Theory for Computing Science.* International Series in Computer Science. Prentice Hall, 1990.
20. R. L. Crole. *Categories for Types.* Cambridge Mathematical Textbooks. Cambridge University Press, 1993. xvii+335 pages, ISBN 0521450926HB, 0521457017PB.
21. P.J. Freyd and A. Scedrov. *Categories, Allegories.* Elsevier Science Publishers, 1990. Appears as Volume 39 of the North-Holland Mathematical Library.
22. Bart Jacobs. *Categorical Logic and Type Theory.* Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Elsevier, 1999.
23. P.T. Johnstone. *Topos Theory.* Academic Press, 1977.
24. J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.
25. S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics.* Springer-Verlag, 1971.
26. C. McLarty. *Elementary Categories, Elementary Toposes*, volume 21 of *Oxford Logic Guides.* Oxford University Press, 1991.
27. Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory.* Universitext. Springer-Verlag, New York, 1992.
28. Saunders Mac Lane and Ieke Moerdijk. Topos theory. In M. Hazewinkel, editor, *Handbook of Algebra, Vol. 1*, pages 501–528. North-Holland, Amsterdam, 1996.
29. B.C. Pierce. *Basic Category Theory for Computer Scientists.* Foundations of Computing Series. The MIT Press, 1991.
30. A. Poigné. Basic category theory. In *Handbook of Logic in Computer Science, Volume 1.* Oxford University Press, 1992.
31. Paul Taylor. *Practical Foundations of Mathematics.* Number 59 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 1999.
32. R. F. C. Walters. *Categories and Computer Science.* Number 28 in Cambridge Computer Science Texts. Cambridge University Press, 1991.

Subject Index

Notation Index