# Denotation of contextual modal type theory (CMTT): syntax and metaprogramming

Murdoch J. Gabbay

*www.gabbay.org.uk*

*School of Mathematical and Computer Sciences, Heriot–Watt University, Riccarton Edinburgh, EH14 4AS, United Kingdom*

Aleksandar Nanevski

*software.imdea.org/~aleks/*

*IMDEA Software, Facultad de Informática (UPM), Campus Montegancedo, 28660 Boadilla del Monte, Madrid, Spain*

**Abstract**

The modal logic S4 can be used via a Curry-Howard style correspondence to obtain a $\lambda$-calculus. Modal (boxed) types are intuitively interpreted as 'closed syntax of the calculus'. This $\lambda$-calculus is called modal type theory—this is the basic case of a more general *contextual* modal type theory, or CMTT.

CMTT has never been given a denotational semantics in which modal types *are* given denotation as closed syntax. We show how this can indeed be done, with a twist. We also use the denotation to prove some properties of the system.

*Key words:* S4 modal logic; Curry-Howard correspondence; contextual modal type theory; meta-programming; higher-order logic; syntax

**Contents**

**1 Introduction** **2**
- 1.1 Keeping it simple . . . 3
- 1.2 Key ideas . . . 3
- 1.3 On intuitions . . . 4
  - 1.3.1 'Syntax' means syntax . . . 4
  - 1.3.2 'Functions' means functions . . . 5

**2 Syntax and typing of the system with box types** **5**
- 2.1 The basic syntax . . . 5
- 2.2 Typing . . . 7
- 2.3 Examples of terms typable in the modal system . . . 8
  - 2.3.1 Short examples . . . 8
  - 2.3.2 There is no natural term of type $A \to \Box A$ . . . 8
  - 2.3.3 A term for *Axiom K* . . . 9
  - 2.3.4 The example of exponentiation . . . 9
- 2.4 Substitution . . . 9

**3 Denotational semantics for types and terms of the modal type system** **11**
3.1 Denotation of types . . . 12
3.2 Denotation of terms . . . 13
3.3 Discussion of the denotation . . . 14
3.3.1 About the term-formers . . . 14
3.3.2 Example: denotation of $let\ X{=}\Box(1+2)\ in\ \Box\Box X_{@}$ . . . 15
3.3.3 Why the natural version does not work . . . 15
3.3.4 Example: denotation of $exp\,2$ . . . 16
3.3.5 Example: denotation of terms for axioms $(\mathbf{T})$ and $(\mathbf{4})$ . . . 16
3.4 Results about the denotation . . . 16

**4 Reduction** **18**
4.1 Results concerning substitution on atoms . . . 18
4.2 Results concerning substitution on unknowns . . . 19
4.3 Reduction . . . 20

**5 Syntax and typing of the system with contextual types** **21**
5.1 Syntax of the contextual system . . . 21
5.2 Typing for the contextual system . . . 23
5.3 Substitution . . . 23

**6 Contextual models** **25**
6.1 Denotational semantics . . . 25
6.2 Typings and denotations in the contextual system . . . 28
6.2.1 Moving between $[A]B$ and $[\,](A{\to}B)$ . . . 28
6.2.2 The example of exponentiation, revisited . . . 29
6.2.3 Syntax to denotation . . . 29
6.2.4 Modal-style axioms . . . 29
6.2.5 More general contexts . . . 30

**7 Shapeliness** **30**

**8 $\Box$ as a (relative) comonad** **32**
8.1 $\Box$ as a comonad . . . 33
8.2 $\Box$ as a relative comonad . . . 35

**9 Conclusions** **35**

## 1. Introduction

The box modality $\Box$ from modal logic has proven its usefulness in logic. It admits various logical and semantic interpretations in the spirit of 'we know that' or 'we can prove that' or 'in the future it will be the case that'. A nice historical overview of modal logic, which also considers the specific impact of computer science, is in [BdRV01, Subsection 1.7].

CMTT (contextual modal type theory) is a typed $\lambda$-calculus based via the Curry-Howard correspondence on the modal logic S4. The box modality becomes a type-former, and box types are intuitively interpreted as 'closed syntax of'.

So CMTT has types for programs that generate CMTT syntax.

Because of this, CMTT has been applied to meta-programming, but it has independent interest as a language, designed according to rigorous mathematical principles and in harmony with modal logic, which interprets $\Box$ in a programming rather than a logical context. Box types are types of the syntax of terms.

Until now this has not been backed up by a denotational semantics in which box types really *are* populated by the syntax of terms. In this paper, we do that: our intuitions are realised in the denotational semantics in a direct and natural, and also unexpected, manner.

The denotation is interesting from the point of view of the interface between logic and programming. Furthermore, we exploit the denotation to prove properties of the language, showing how denotations are not only illuminating but can also serve for new proof-methods.

### *1.1. Keeping it simple*

This paper considers two related systems:

- The purely modal system, based on box types like $\Box A$.
- The contextual modal system, based on 'boxes containing types' like $[A_1, A_2]B$—the reader might like to think of the contextual system as a multimodal logic [GKWZ03, Subsection 1.4] (whose modalities are themselves indexed over propositions).

Broadly speaking, the purely modal system is nicer to study but a little too simple. The contextual modal system generalises the purely modal system and gives it slightly more expressive power, but it can be a little complicated; not obscure, just long to write out.

Therefore, we open this paper with the modal system, make the main point of our denotation in the simplest and clearest possible manner—the reader who wants to jump right in and work backwards could do worse than start with the example denotations in Subsection 3.3.2 onwards—and then we consider the contextual system as the maths becomes more advanced. Section 2 presents syntax and typing of the modal system and Section 5 does the same for the contextual modal system; Section 3 gives modal denotations and Section 6 gives contextual modal denotations.

The developments are parallel, but not identical. Where proofs are not very different between the modal and contextual systems, we omit routine repetition. We consider reduction of the modal system in Section 4 but not reduction of the contextual system.[1] Also, we develop the important notion of *shapeliness* only for the contextual system in Section 7; it is obvious how the modal case would be a special case.

### *1.2. Key ideas*

Our main technical results are Theorems 3.14 and 6.10, and Corollary 7.8.

However, the key technical ideas that make these results work, and indeed contribute to making them interesting, occur beforehand. It might be useful to briefly enumerate some of them here, as an aid to navigating the mathematics. Exposition is in the body of the paper:

- *Inflation* in the case of $[\![\Box A]\!]$ in Figure 3, and the 'tail of' semantics of $X_{@}$ in Figure 4. This is discussed in Remark 3.5.
- Proposition 2.23 and the fact that it is needed for soundness of the denotation in Theorem 3.14.

[1] We choose not to do this, not because it is hard, but because it is easy (apologies to John F. Kennedy).

- The remarkable Proposition 3.13, in which valuations get turned into substitutions and closed syntax in the denotation interacts directly with the typing system. This is a kind of dual to the interaction seen in Proposition 2.23.
- The denotation of $[\![[A_i]A]\!]$ in Figure 8, which in the context of the rest of the paper is very natural.
- The notion of *shapeliness* in Definition 7.1 and the 'soundness result' Proposition 7.7.

### *1.3. On intuitions*

#### *1.3.1. 'Syntax' means syntax*

One early difficulty the authors of this paper faced was in communication, because we sometimes used terms synonymously without realising that the words were so slippery.

The intuition we give to $\Box A$ is self-reflectively *closed syntax of the language itself*. This is a distinct intuition from 'computations', 'code', 'values', or 'intensions', because these are not necessarily intended self-reflectively.

It is very important not to confuse this intuition with apparently similar intuitions expressed as 'code of $A$', 'values of $A$', 'computations of $A$', or 'intension of $A$'. These are not quite the same thing. It may be useful to briefly survey them here:

- 'Code of $A$' is an ambiguous term; this is often understood as precompiled code or bytecode, rather than syntax of the original language. See [WLP98] for a system based on that intuition.
- 'Values of $A$' is a dangerous intuition and there probably should be a law against it: depending on whom one is speaking with, this could be synonymous in their mind with 'normal forms of $A$' (a syntactic notion) or 'denotations of $A$' (a non-syntactic notion). Matters become even worse if one's interlocutor assumes that denotations may be silently added to syntax as constants (fine for mathematicians; not so fine for programmers). More than one conversation has been corrupted by the associated misunderstandings.
- For a discussion of 'computation of $A$' see the Related Work in the Conclusions, where we discuss how this intuition can lead to a notion of Moggi-style *monad*.
- 'Intension of $A$' is similar to 'syntax of $A$', but significantly more general: there is no requirement that the intension be syntactic, or if it is syntactic, that it be the same calculus. One could argue that 'intension of' should also satisfy that the denotation of $\Box\Box A$ be identical in some strong sense—e.g. be the same set as—to that of $\Box A$, since taking an intension twice should reveal no further internal structure. (This does not match the denotation of this paper.)
  An interesting (and as far as we know unexplored) model of this intuition might be partial equivalence relations (**PER**s), where $\Box A$ takes $A$ and forms the identity PER which is defined where $A$ is defined.[2] Famously, PERs form a cartesian-closed category [AL91, Subsection 3.4.1].

In short: where the reader sees '$\Box A$', they should think 'raw syntax in type $A$'.

[2] Alex Simpson and Paul Levy both independently suggested PERs when the first author sketched the ideas of this paper, and Simpson went further and suggested the specific model discussed above. We are grateful to Levy and Simpson for their comments, which prompted us to be specific about the intuition behind the particular denotation in this paper.

*1.3.2. 'Functions' means functions*

It may be useful now to head off another possible confusion: where the reader sees $A{\to}B$, they should think 'graph of a function'—not 'computable function', 'representable function', 'syntax of a function', or 'code of a function'.

All of these things are also possible, but in this paper our challenge is to create a type system, language, and denotation which are 'epsilon away' from the simply-typed $\lambda$-calculus or (since we admit a type of truth-values) higher-order logic—and it just so happens that we also have modal types making precisely its *own* syntax into first-class data.

So: we are considering a 'foundations-flavoured' theory in which $A{\to}B$ represents all possible functions (in whatever foundation the reader prefers) from $A$ to $B$, and we do not intend this paper to be 'programming-flavoured' in which $A{\to}B$ represents only that function(-code) or normal forms that can exist inside some computational device. And, $\Box A$ should represent, as much as possible, 'the syntax of our language/logic that types as $A$'.

## 2. Syntax and typing of the system with box types

We start by presenting the types, terms, and typing relation for the modal type system. This is the simplest version of the language that we want to give a denotational semantics for.[3]

*2.1. The basic syntax*

**Definition 2.1.** Fix two countably infinite sets of **variables** $\mathbb{A}$ and $\mathbb{X}$. We will observe a **permutative convention** that $a, b, c, \ldots$ will range over distinct variables in $\mathbb{A}$ and $X, Y, Z, \ldots$ will range over distinct variables in $\mathbb{X}$. We call $a, b, c$ **atoms** and $X, Y, Z$ **unknowns**.

**Definition 2.2.** Define **types** inductively by:

$$A ::= o \mid \mathbb{N} \mid A{\to}A \mid \Box A$$

**Notation 2.3.** By convention, if $X$ and $Y$ are sets we will write $Y^X$ for the set of functions from $X$ to $Y$. This is to avoid any possible confusion between $A{\to}B$ (which is a type) and $Y^X$ (which is a set).

**Remark 2.4.**
- $o$ will be a type of **truth values**; its denotation will be populated by truth-values $\{\bot, \top\}$.
- $\mathbb{N}$ will be a type of **natural numbers**; its denotation will be populated by numbers $\{0, 1, 2, \ldots\}$.

[3] This section follows the ideas presented in [PD01]; Definition 2.6 corresponds to [PD01, Subsection 2.2] and Figure 1 corresponds to [PD01, Subsection 2.3].

We use a different syntax and make some other superficial changes to ordering and presentation. So the material is just slightly rewritten, reorganised, and represented from previous work. We also concentrate some examples in Subsection 2.3.

There is one small but interesting technical difference: we write $X_@$ where Pfenning and Davies would just write $X$ (or $u$, in their notation; the point is there is no $\text{-}_@$). This propagates to our definition of substitution, where the $\text{-}_@$ must be explicitly stripped off in Figure 2.

There are two reasons for this: when we come to the contextual system the $\text{-}_@$ generalises to the more non-trivial $\text{-}@(r_1)_1^n$ so even though every $X$ occurs as $X_@$ here we prefer to keep the $\text{-}_@$; and also, in our denotation $\text{-}_@$ corresponds to the non-trivial operation of 'take the tail of the list', see Figure 4.

- $A{\to}B$ is a **function type**; its denotation will be populated by functions.
- $\Box A$ is a **modal type**; its denotation will be populated by syntax.

**Definition 2.5.** Fix a set of **constants** $C$ to each of which is assigned a type $\mathit{type}(C)$. We write $C : A$ as shorthand for '$C$ is a constant and $\mathit{type}(C) = A$'. We insist that constants include the following:

$$\bot : o \qquad \top : o \qquad \mathsf{isapp}_A : (\Box A){\to}o$$

We may also assume constants for $\mathbb{N}$, such as $0 : \mathbb{N}$, $\mathsf{succ} : \mathbb{N}{\to}\mathbb{N}$, $* : \mathbb{N}{\to}\mathbb{N}{\to}\mathbb{N}$ and $+ : \mathbb{N}{\to}\mathbb{N}{\to}\mathbb{N}$, a fixedpoint combinator, we may write $1$ for $\mathsf{succ}(0)$, and so on.[4]

We may omit type subscripts where they are clear from context or do not matter.

**Definition 2.6.** Define **terms** inductively by:

$$r ::= C \mid a \mid X_{@} \mid \lambda a{:}A.r \mid rr \mid \Box r \mid \mathit{let}\ X{=}r\ \mathit{in}\ r$$

Constants $C$ are, as standard in the $\lambda$-calculus, added as desired to represent logic and computational primitives. An atom $a$ plays the role of a standard $\lambda$-calculus variable; it is $\lambda$-abstracted in a typed manner in $\lambda a{:}A.r$. The term $X_{@}$ means intuitively 'evaluate $X$' and $\Box r$ means intuitively 'the syntax $r$ considered itself in the denotation'. Finally $\mathit{let}\ X{=}s\ \mathit{in}\ r$ means intuitively 'set $X$ to be the syntax calculated by $s$, in $r$'. Examples of this in action are given and discussed in Subsection 2.3.

**Remark 2.7.** The effect of $r_{@}$ (which is not syntax) is obtained by $\mathit{let}\ X{=}r\ \mathit{in}\ X_{@}$. Likewise the effect of $\lambda X{:}\Box A.r$ (which is not syntax) is obtained by $\lambda a{:}\Box A.\mathit{let}\ X{=}a\ \mathit{in}\ r$.

We *cannot* emulate $\mathit{let}\ X{=}s\ \mathit{in}\ X_{@}$ using $(\lambda a{:}A.a_{@})r$. The expression '$a_{@}$' would mean 'evaluate the syntax $a$' rather than 'evaluate the syntax linked to $a$'.[5]

**Definition 2.8.** Define **free atoms** $\mathit{fa}(r)$ and **free unknowns** $\mathit{fu}(r)$ by:

$$\begin{array}{rlrl}
\mathit{fa}(C) &= \varnothing & \mathit{fa}(a) &= \{a\}\\
\mathit{fa}(\lambda a{:}A.r) &= \mathit{fa}(r) \setminus \{a\} & \mathit{fa}(rs) &= \mathit{fa}(r) \cup \mathit{fa}(s)\\
\mathit{fa}(\Box r) &= \mathit{fa}(r) & \mathit{fa}(\mathit{let}\ X{=}s\ \mathit{in}\ r) &= \mathit{fa}(r) \cup \mathit{fa}(s)\\
\mathit{fa}(X_{@}) &= \varnothing
\end{array}$$

$$\begin{array}{rlrl}
\mathit{fu}(C) &= \varnothing & \mathit{fu}(a) &= \varnothing\\
\mathit{fu}(\lambda a{:}A.s) &= \mathit{fu}(s) & \mathit{fu}(rs) &= \mathit{fu}(r) \cup \mathit{fu}(s)\\
\mathit{fu}(\Box r) &= \mathit{fu}(r) & \mathit{fu}(\mathit{let}\ X{=}s\ \mathit{in}\ r) &= (\mathit{fu}(r) \setminus \{X\}) \cup \mathit{fu}(s)\\
\mathit{fu}(X_{@}) &= \{X\}
\end{array}$$

If $\mathit{fa}(r) \cup \mathit{fu}(r) = \varnothing$ then we call $r$ **closed**.

[4] ...so we follow the example of PCF [Mit96]. A fixedpoint combinator cannot be interpreted in the denotational semantics of Section 3, which adopts a 'sets-and-functions' flavour following semantics for Higher-Order Logic (type theory).

[5] In addition even if $a_{@}$ were syntax, it would not type in the typing system of Figure 1, because $\mathit{fa}(a_{@})$ would be equal to $\{a\} \neq \varnothing$ (Definition 2.8). Modal types are inhabited by *closed* syntax (Definition 2.8).

$$\frac{}{\Gamma, a : A \vdash a : A}\ (\mathbf{Hyp}) \qquad \frac{}{\Gamma \vdash C : \mathit{type}(C)}\ (\mathbf{Const})$$

$$\frac{\Gamma, a{:}A \vdash r : B}{\Gamma \vdash (\lambda a{:}A.r) : A \to B}\ (\to\mathbf{I}) \qquad \frac{\Gamma \vdash r' : A \to B \quad \Gamma \vdash r : A}{\Gamma \vdash r'r : B}\ (\to\mathbf{E})$$

$$\frac{\Gamma \vdash r : A \quad (\mathit{fa}(r){=}\varnothing)}{\Gamma \vdash \Box r : \Box A}\ (\Box\mathbf{I}) \qquad \frac{\Gamma \vdash s{:}\Box A \quad \Gamma, X{:}\Box A \vdash r{:}B}{\Gamma \vdash \mathit{let}\ X{=}s\ \mathit{in}\ r : B}\ (\Box\mathbf{E})$$

$$\frac{}{\Gamma, X : \Box A \vdash X_@ : A}\ (\mathbf{Ext})$$

**Figure 1:** Modal type theory typing rules

**Definition 2.9.** We take $a$ to be bound in $r$ in $\lambda a{:}A.r$ and $X$ to be bound in $r$ in $\mathit{let}\ X{=}s\ \mathit{in}\ r$, and we take syntax up to $\alpha$-equivalence as usual. We omit definitions but give examples:

- $\lambda a{:}A.a = \lambda b{:}A.b$.
- $\lambda a{:}A.(X_@ a) = \lambda b{:}A.(X_@ b)$.
- $\mathit{let}\ X{=}\Box a\ \mathit{in}\ X_@ b = \mathit{let}\ Y{=}\Box a\ \mathit{in}\ Y_@ b$.

As the use of an equality symbol above suggests, we identify terms up to $\alpha$-equivalence.[6]

*2.2. Typing*

**Definition 2.10.**
- A **typing** is a pair $a : A$ or $X : \Box A$.
- A **typing context** $\Gamma$ is a finite partial function from $\mathbb{A} \cup \mathbb{X}$ to types.
- A **typing sequent** is a tuple $\Gamma \vdash r : A$ of a typing context, a term, and a type.

We use list notation for typing contexts, e.g. $a{:}A, Y{:}B$ is the function mapping $a$ to $A$ and $Y$ to $B$; and $a{:}A \in \Gamma$ means that $\Gamma(a)$ is defined and $\Gamma(a) = A$.

Define the **valid typing sequents** of the modal type system inductively by the rules in Figure 1.

We discuss examples of typable terms in Subsection 2.3. The important rule is $(\Box\mathbf{I})$, which tells us that if we have some syntax $r$ and it has no free atoms, then we can box it as a denotation $\Box r$ of box type—any free unknowns $X$ in $r/\Box r$ get linked to further boxed syntax, which is expressed by $(\Box\mathbf{E})$.

**Notation 2.11.** We may write $\varnothing \vdash r : A$ just as $r : A$.

**Notation 2.12.** If $\Gamma$ is a typing context and $U \subseteq \mathbb{A} \cup \mathbb{X}$ then write $\Gamma|_U$ for $\Gamma$ **restricted** to $U$. This is the partial function which is equal to $\Gamma$ where it is defined, and $\mathit{dom}(\Gamma|_U) = \mathit{dom}(\Gamma) \cap U$.

[6] Using nominal abstract syntax [GP01] this identification can be made consistent with the use of names for bound atoms *and* the inductive definition in Definition 2.6. However, studying how best to define syntax is not the emphasis of this paper.

Proposition 2.13 combines Weakening and Strengthening:

**Proposition 2.13.** *If* $\Gamma \vdash r : A$ *and* $\Gamma'|_{fu(r)\cup fa(r)} = \Gamma|_{fu(r)\cup fa(r)}$ *then* $\Gamma' \vdash r : A$.

*Proof.* By a routine induction on $r$. □

### 2.3. *Examples of terms typable in the modal system*

We are now ready to discuss intuitions about this syntax; for a more formal treatment see Section 3 which develops the denotational semantics. We start with some short examples and then consider more complex terms.

#### 2.3.1. *Short examples*

1. Assume constants $\neg : o{\to}o$ and $\wedge : o{\to}o{\to}o$, where $\wedge$ is written infix as usual. Then we can type

   $$\varnothing \vdash \lambda a{:}\Box o.\mathit{let}\ X{=}a\ \mathit{in}\ \Box(\neg X_@)\ :\ \Box o{\to}\Box o.$$
   $$\varnothing \vdash \lambda a{:}\Box o.\lambda b{:}\Box o.\mathit{let}\ X{=}a\ \mathit{in}\ \mathit{let}\ Y{=}b\ \mathit{in}\ \Box(X_@ \wedge Y_@)\ :\ \Box o{\to}\Box o{\to}\Box o.$$
   $$\varnothing \vdash \lambda a{:}\Box o.\mathit{let}\ X{=}a\ \mathit{in}\ \Box(X_@ \wedge X_@)\ :\ \Box o{\to}\Box o.$$

   Intuitively these represents the syntax transformations $P \mapsto \neg P$, $P, Q \mapsto P \wedge Q$, and $P \mapsto P \wedge P$.
2. This program takes syntax of type $A$ and evaluates it:

   $$\varnothing \vdash \lambda a{:}\Box A.\mathit{let}\ X{=}a\ \mathit{in}\ X_@\ :\ \Box A{\to}A$$

   This corresponds to the modal logic axiom $(\mathbf{T})$.
3. Expanding on the previous example, this program takes syntax for a function and an argument, evaluates the syntax and applies the function to the argument:

   $$\varnothing \vdash \lambda a{:}\Box(A{\to}B).\lambda b{:}A.(\mathit{let}\ X{=}a\ \mathit{in}\ X_@)b\ :\ \Box(A{\to}B) \to (A{\to}B)$$
4. This program takes syntax of type $A$ tagged with $\Box$, and adds an extra $\Box$ so that it becomes syntax of type $\Box A$:

   $$\varnothing \vdash \lambda a{:}\Box A.\mathit{let}\ X{=}a\ \mathit{in}\ \Box\Box X_@\ :\ \Box A{\to}\Box\Box A$$

   This corresponds to the modal logic axiom $(\mathbf{4})$.

#### 2.3.2. *There is no natural term of type* $A{\to}\Box A$

We can try to give $\lambda a{:}o.\Box a$ the type $A{\to}\Box A$, but we fail because the typing context $a{:}o$ does not satisfy $fa(a) = \varnothing$.

Our denotation of Figures 3 and 4 illustrates that it is not in general possible to invert the evaluation map from Subsection 2.3.1 and thus map $A$ to $\Box A$. This is Corollary 3.15.[7] So

- there is a canonical map $\Box A{\to}A$ (syntax to denotation)—we saw this map in part 1 of this example—but
- not in general an inverse map $A{\to}\Box A$ (denotation to syntax).

[7] For sufficiently 'small' types this may be possible by specific constructions; see Example 3.16.

*2.3.3. A term for* Axiom K

*Axiom K*, also called the *normality axiom* [BdRV01, Definition 1.39, Subsection 1.6]; its type is $\Box(A{\to}B){\to}\Box A{\to}\Box B$.

We can write a term of this type. Intuitively, the term below takes syntax for a function and syntax for an argument, and produces syntax for the function applied to the argument:

$$\varnothing \vdash \lambda a{:}\Box(A{\to}B).\lambda b{:}\Box A.\mathit{let}\ Y{=}b\ \mathit{in}\ \mathit{let}\ X{=}a\ \mathit{in}\ \Box(X_{@}Y_{@}) : \Box(A{\to}B){\to}\Box A{\to}\Box B$$

**Remark 2.14.** We exhibited terms of type $\Box A{\to}A$, $\Box A{\to}\Box\Box A$, and $\Box(A{\to}B){\to}\Box A{\to}\Box B$, so Figure 1 implements (at least) the deductive power of an intuitionistic variant of S4 [BdRV01, Subsection 4.1, page 194].[8]

The reader familiar with category theory may also ask whether $\Box$ can be viewed as a *comonad*, since $\Box A{\to}A$ and $\Box A{\to}\Box\Box A$ look like the types of a *counit* and *comultiplication* (and perhaps $\Box(A{\to}B){\to}\Box A{\to}\Box B$ looks like the action of a functor). We return to this in Section 8.

*2.3.4. The example of exponentiation*

This is a classic example of meta-programming: write a function that takes a number $n$ and returns syntax for the function $x \in \mathbb{N} \mapsto x^n$.

Assuming a combinator for primitive recursion over natural numbers and using some standard sugar, the following term implements exponentiation:

$$\begin{array}{c} \mathit{exp}\ 0 \Rightarrow \Box\lambda b{:}\mathbb{N}.1 \\ \mathit{exp}\ (\mathit{succ}(n)) \Rightarrow \mathit{let}\ X{=}\mathit{exp}\ n\ \mathit{in}\ \big(\Box\lambda b{:}\mathbb{N}.b * (X_{@}b)\big). \end{array}$$

However, the term above generates $\beta$-reducts. The reader can see this because of the '$\Box\lambda b{:}\mathbb{N}.b{*}(X_{@}b)$' above. This application $X_{@}b$ is trapped under a $\Box$ and *will not* reduce.

Looking ahead to the reduction relation in Figure 5, $\mathit{exp}\ 2$ reduces to

$$\Box(\lambda b{:}\mathbb{N}.b * (\lambda b{:}\mathbb{N}.b * ((\lambda b{:}\mathbb{N}.1)b)b)) \quad \text{and not to} \quad \Box(\lambda b{:}\mathbb{N}.(b * b * 1)).$$

Looking ahead to the denotation of Figure 4, the denotation of $\mathit{exp}\ 2$ will likewise be $\Box(\lambda b{:}\mathbb{N}.b{*}(\lambda b{:}\mathbb{N}.b * ((\lambda b{:}\mathbb{N}.1)b)b))$ in a suitable sense. We indicate the calculation in Subsection 3.3.4.

The contextual system of Section 5 deals with this particular issue; see Subsection 6.2.2.

*2.4. Substitution*

**Definition 2.15.** An **(atoms-)substitution** $\sigma$ is a finite partial function from atoms $\mathbb{A}$ to terms. $\sigma$ will range over atoms-substitutions.

Write $\mathit{dom}(\sigma)$ for the set $\{a \mid \sigma(a)\ \text{defined}\}$.

Write $\mathit{id}$ for the **identity** substitution, such that $\mathit{dom}(\sigma) = \varnothing$.

Write $[a{:=}t]$ for the map taking $a$ to $t$ and undefined elsewhere.

An **(unknowns-)substitution** $\theta$ is a finite partial function from unknowns $\mathbb{X}$ to terms such that for every $X$, if $X \in \mathit{dom}(\theta)$ then $\theta(X) = \Box r$ for some $r$ with $\mathit{fa}(r) = \varnothing$.

$\theta$ will range over unknowns-substitutions.

We write $\mathit{dom}(\theta)$, $\mathit{id}$, and $[X{:=}t]$ just as for atoms-substitutions.

[8] The list of axioms of [BdRV01, page 194] uses $\Diamond$ instead of $\Box$.

A most remarkable family of theorems of Kripke semantics for modal logic relates geometric properties of the Kripke frame's *accessibility relation* with logical properties of the modalities. Axiom (**K**) is satisfied by all frames. Axiom (**T**) expresses geometrically that accessibility is reflexive. Axiom (**4**) expresses that accessibility is transitive.

$$
\begin{array}{ccc}
C\sigma = C & a\sigma = \sigma(a) & (a \in dom(\sigma)) \\
(rs)\sigma = (r\sigma)(s\sigma) & a\sigma = a & (a \notin dom(\sigma)) \\
(\Box r)\sigma = \Box(r\sigma) & (\lambda c{:}A.r)\sigma = \lambda c{:}A.(r\sigma) & (c \notin fa(\sigma)) \\
X_{@}\sigma = X_{@} & (let\, Y{=}s\ in\, r)\sigma = let\, Y{=}s\sigma\ in\, r\sigma & (Y \notin fu(\sigma)) \\
 & & \\
C\theta = C & a\theta = a & \\
(rs)\theta = (r\theta)(s\theta) & X_{@}\theta = s' & (\theta(X) = \Box s') \\
(\Box r)\theta = \Box(r\theta) & X_{@}\theta = X_{@} & (X \notin dom(\theta)) \\
(\lambda c{:}A.r)\theta = \lambda c{:}A.(r\theta) & (let\, Y{=}s\ in\, r)\theta = let\, Y{=}s\theta\ in\, r\theta & (Y \notin fu(\theta))
\end{array}
$$

**Figure 2:** Substitution actions for atoms and unknowns

**Definition 2.16.** Define

$$
\begin{array}{l}
fa(\sigma) = dom(\sigma) \cup \bigcup\{fa(\sigma(a)) \mid a{\in}dom(\sigma)\} \\
fu(\sigma) = \bigcup\{fu(\sigma(a)) \mid a{\in}dom(\sigma)\} \\
fu(\theta) = dom(\theta) \cup \bigcup\{fu(\theta(X)) \mid X{\in}dom(\theta)\}.
\end{array}
$$

**Remark 2.17.** Where $\theta$ is defined, it maps $X$ specifically to terms of the form $\Box r$ with $fa(r) = \varnothing$.

This is because '$\Box r$ with $fa(r) = \varnothing$' is the syntax inhabiting modal types. If we consider another class of syntax (e.g. in the contextual system of Section 5 onwards), then the corresponding notion of unknowns-substitution changes in concert with that.

Definition 2.18 describes how atoms and unknowns get instantiated. We discuss it in Remark 2.20 but one point is important above all others: if $\theta(X) = \Box s'$ then $X_{@}\theta$ is equal to $s'$. A very simple reduction/computation is built in to the substitution action for unknowns, that $(\Box s')_{@} \to s'$.[9]

**Definition 2.18.** Define **atoms** and **unknowns** substitution actions $r\sigma$ and $r\theta$ inductively by the rules in Figure 2.

Lemma 2.19 illustrates a nice corollary of the point discussed in Remark 2.17. It will be useful later in Proposition 3.13.

**Lemma 2.19.** $fa(r\theta) = fa(r)$.

*Proof.* By a routine induction on $r$ using our assumption of Definition 2.15 that if $X \in dom(\theta)$ then $fa(\theta(X)) = \varnothing$. □

**Remark 2.20.** A few comments on Definition 2.18:

- The two capture-avoidance side-conditions $c \notin fa(\sigma)$ and $Y \notin fu(\theta)$ can always be guaranteed by renaming.

[9] $(\Box s')_{@}$ is not actually syntax, but if it were, then $(\Box s')_{@} \to s'$ would be its reduction.

- We write $(\Box r)\sigma = \Box(r\sigma)$. This is computationally wasteful in the sense that the side-condition $fa(r) = \varnothing$ on $(\Box\mathbf{I})$ (Figure 1) guarantees that for typable terms (which is what we care about) $r\sigma = r$.
  We prefer to keep basic definitions orthogonal from such optimisations, but this is purely a design choice (and see the next item in this list).
- We write $(\lambda c{:}A.r)\theta = \lambda c{:}A.(r\theta)$ without any side-condition that $c$ should avoid capture by atoms in $\theta$. This is because Definition 2.15 insists that $fa(\theta(X)) = \varnothing$ always, so there can be no capture to avoid.

Recall the definition of $[a{:=}s]$ from Definition 2.15. Lemma 2.21 is a standard lemma which will be useful later:

**Lemma 2.21.** *If $a \notin fa(r)$ then $r[a{:=}s] = r$.*

*Proof.* By a routine induction on $r$. □

Definition 2.22 and Proposition 2.23 are needed for Proposition 3.13.

**Definition 2.22.** Suppose $\Gamma$ is a typing context and $\theta$ is an unknowns-substitution. Write $\Gamma \vdash \theta$ when if $X \in dom(\theta)$ then $X{:}\Box A \in \Gamma$ for some $A$ and $\Gamma \vdash \theta(X) : \Box A$.

Proposition 2.23 is needed for Theorem 3.14 (soundness of the denotation). It is slightly unusual that soundness of typing under substitution should be needed for soundness under taking denotations. But the syntax is going to be *part of* the denotational semantics—that is its point—and so substitution is part of how this denotation is calculated (see the case of $\Box r$ in Figure 4).

**Proposition 2.23.** *Suppose $\Gamma$ is a typing context and $\theta$ is an unknowns-substitution and suppose $\Gamma \vdash \theta$ (Definition 2.22). Then $\Gamma \vdash r : A$ implies $\Gamma \vdash r\theta : A$.*

*Proof.* By a routine induction on the typing of $r$. We consider four cases:

- *The case of* $(\Box\mathbf{I})$. Suppose $\Gamma \vdash r : A$ and $fa(r) = \varnothing$ so that $\Gamma \vdash \Box r : \Box A$ by $(\Box\mathbf{I})$. By inductive hypothesis $\Gamma \vdash r\theta : A$. By Lemma 2.19 also $fa(r\theta) = \varnothing$. We use $(\Box\mathbf{I})$ and the fact that $(\Box r)\theta = \Box(r\theta)$, and Proposition 2.13.
- *The case of* $(\mathbf{Ext})$ *for* $X \in dom(\theta)$. By assumption in Definition 2.15, $\theta(X) = \Box r'$ for some $r'$ with $fa(r') = \varnothing$. By assumption in Definition 2.22 $\varnothing \vdash \theta(X) : \Box A$. By Definition 2.18 $(X_{@})\theta = r'$. By Proposition 2.13 $\Gamma \vdash r' : A$ as required.
- *The case of* $(\to\mathbf{I})$. Suppose $\Gamma, a{:}A \vdash r : B$ so that by $(\to\mathbf{I})$ $\Gamma \vdash \lambda a{:}A.r : A{\to}B$. By inductive hypothesis $\Gamma, a{:}A \vdash r\theta : B$. We use $(\to\mathbf{I})$.
- *The case of* $(\Box\mathbf{E})$. Suppose $\Gamma, X{:}\Box A \vdash r : B$ and $\Gamma \vdash s : \Box A$ so that by $(\Box\mathbf{E})$ $\Gamma \vdash \mathit{let}\ X{=}s\ \mathit{in}\ r : B$. Renaming if necessary, suppose $X \notin dom(\theta)$. By inductive hypothesis $\Gamma, X{:}\Box A \vdash r\theta : B$ and $\Gamma \vdash s\theta : \Box A$. We use $(\Box\mathbf{E})$ and the fact that $(\mathit{let}\ X{=}s\ \mathit{in}\ r)\theta = \mathit{let}\ X{=}s\theta\ \mathit{in}\ r\theta$.

□

## 3. Denotational semantics for types and terms of the modal type system

We now develop a denotational semantics of the types and terms from Definitions 2.2 and 2.6. The main definitions are in Figures 3 and 4. The design is subtle, so there follows an extended discussion of the definition.

$$
\begin{aligned}
[\![o]\!] &= \{\top^{\mathcal{H}}, \bot^{\mathcal{H}}\} && \textit{truth-values}\\
[\![\mathbb{N}]\!] &= \{0, 1, 2, \dots\} && \textit{natural numbers}\\
[\![A{\to}B]\!] &= [\![B]\!]^{[\![A]\!]} && \textit{function-spaces}\\
[\![\Box A]\!] &= \{\Box r \mid \varnothing \vdash \Box r : \Box A\} \times [\![A]\!] && \textit{closed syntax \& purported denotation}
\end{aligned}
$$

**Figure 3:** Denotational semantics of modal types

### *3.1. Denotation of types*

**Definition 3.1.** Define $[\![A]\!]$ the **interpretation** of types by induction in Figure 3.

**Remark 3.2.** $[\![o]\!]$ is a pair of truth-values, and $[\![\mathbb{N}]\!]$ is the set of natural numbers. $[\![B]\!]^{[\![A]\!]}$ is a function-space.[10] No surprises here.

$z \in [\![\Box A]\!]$ is a pair $(\Box r, x)$. We suggest the reader think of this as

- some syntax $\Box r$ and[11]
- its purported denotation $x$.

We say 'purported' because there is no restriction that $x$ actually be a possible denotation of $r$. For instance, it is a fact that $\Box(0+1) :: 2 \in [\![\Box\mathbb{N}]\!]$, and $\Box(0+1) :: 2$ will not be the denotation of any $r$ such that $\varnothing \vdash r : \mathbb{N}$ (to check this, unpack Definition 3.11 below).

So our semantics inflates: there are usually elements in $[\![\Box A]\!]$ that are not the denotation of any closed term. The reader should remain calm; there are also usually elements in function-spaces that are not the denotation of any closed term. The inflated elements in our semantics are an important part of our design.

**Notation 3.3.** We will want to talk about nested pairs of the form $(x_1, (x_2, \dots, (x_n, x_{n+1})))$. Accordingly we will use list notation, writing $x_1 :: x_2$ for $(x_1, x_2)$ and $x_1 :: \cdots :: x_n :: x_{n+1}$ for $(x_1, (x_2, \dots, (x_n, x_{n+1})))$. See for instance Remark 3.4, Figure 4, and Subsection 3.3.2.

**Remark 3.4.** Note that as standard, distinct syntax may have equal denotation. For instance, $\Box(0+1) :: 1$ and $\Box(1+0) :: 1$ are not equal in $[\![\mathbb{N}]\!]$.

**Remark 3.5.** Why do we inflate? Surely it is both simpler and more intuitive to take $[\![\Box A]\!]$ to be $\{\Box r \mid \varnothing \vdash \Box r : \Box A\}$.

We could do this, but then later on in Definition 3.11 we would not be able to give a denotation to terms by induction on their syntax.

[10] We could restrict this to computable functions or some other smaller set but we have our logician's hat on here, not our programmer's hat on: we *want* the larger set. This will make Corollary 3.15 work. If we chose a smaller, more sophisticated, and more complex notion of function-space here, then this would actually *weaken* the results we then obtain from the semantics.

[11] We could drop the $\Box$ and just write $(r, x)$, but when we build the contextual system in Section 5 the $\Box$ will fill with bindings (see Definition 5.4) and cannot be dropped, so we keep it here.

The problem is that our types, and terms, are designed to permit generation of syntax at modal type. Thus, our design brief is to allow dynamic (runtime) generation of syntax. With the 'intuitive' definition above, there is no guarantee of an inductively decreasing quantity; the runtime can generate syntax of any size. To see this in detail, see Subsection 3.3.3.

The design of $[\![\Box A]\!]$ in Figure 3 gets around this by insisting, at the very moment we assert some denotation of a term $r$ of type $\Box A$—i.e. some syntax $r'$ of type $A$—to simultaneously volunteer a denotation for $r'$—i.e. an element in the denotation of $A$. (As mentioned in Remark 3.2 this denotation might be in some sense mistaken, but perhaps surprisingly that will not matter.)

*3.2. Denotation of terms*

We now set about interpreting terms in the denotation for types from Definition 3.1. The main definition is Definition 3.11. First, however, we need:

- some tools to handle the 'syntax and purported denotation' design of $[\![\Box A]\!]$ (Definition 3.6); and
- a suitable notion of valuation (Definition 3.7).

We then discuss the design of the definitions.

Recall from Notation 3.3 that we may use list notation and write $\Box r :: x$ for $(\Box r, x)$.

**Definition 3.6.** We define $hd$ and $tl$ on $x \in [\![A]\!]$ (Definition 3.1) as follows:

- If $x \in [\![o]\!]$ or $[\![\mathbb{N}]\!]$ or $[\![A{\to}B]\!]$ then $hd(x) = x$ and $tl(x)$ is undefined.
- If $(\Box r, x) \in [\![\Box A]\!]$ then $hd((\Box r, x)) = \Box r$ (first projection) and $tl((\Box r, x)) = x$ (second projection).

**Definition 3.7.** A **valuation** $\varsigma$ is a finite partial function on $\mathbb{A} \cup \mathbb{X}$. Write $\varsigma[X{:=}x]$ for the valuation such that:

- $(\varsigma[X{:=}x])(X) = x$.
- $(\varsigma[X{:=}x])(Y) = \varsigma(Y)$ if $\varsigma(Y)$ is defined, for all $Y$ other than $X$.
- $(\varsigma[X{:=}x])(a) = \varsigma(a)$ if $\varsigma(a)$ is defined.
- $(\varsigma[X{:=}x])$ is undefined otherwise.

Define $\varsigma[a{:=}x]$ similarly.

**Definition 3.8.** Suppose $\Gamma$ is a typing context and $\varsigma$ a valuation. Write $\Gamma \vdash \varsigma$ when:

1. $dom(\Gamma) = dom(\varsigma)$.
2. If $a \in dom(\varsigma)$ then $a{:}A \in \Gamma$ for some $A$ and $\varsigma(a) \in [\![A]\!]$.
3. If $X \in dom(\varsigma)$ then $X{:}\Box A \in \Gamma$ for some $A$ and $\varsigma(X) \in [\![\Box A]\!]$.

**Remark 3.9.** Unpacking Definition 3.1, clause 3 of Definition 3.8 (the one for $X$) means that $\varsigma(X) = \Box r' :: x$ where $\varnothing \vdash \Box r' : \Box A$ and $x \in [\![A]\!]$. Note also that by the form of the derivation rules in Figure 1, it follows that $\varnothing \vdash r' : A$. So an intuition for $\varsigma(X)$ (cf. Remark 3.2) is this—

> "$\varsigma(X)$ is some closed syntax $r'$ (presented as $\Box r' \in [\![\Box A]\!]$), and a candidate denotation for it $x \in [\![A]\!]$",

$\varsigma[a{:=}x]$ and $\varsigma[X{:=}x]$ from Definition 3.7. $\varsigma_{\mathbb{X}}$ from Definition 3.10.

$$
\begin{aligned}
[\![\top]\!]_\varsigma &= \top^{\mathcal{H}} \\
[\![\bot]\!]_\varsigma &= \bot^{\mathcal{H}} \\
[\![a]\!]_\varsigma &= \varsigma(a) && (a \in dom(\varsigma)) \\
[\![\lambda a{:}A.r]\!]_\varsigma &= (x{\in}[\![A]\!] \mapsto [\![r]\!]_{\varsigma[a:=x]}) \\
[\![r'r]\!]_\varsigma &= [\![r']\!]_\varsigma\, [\![r]\!]_\varsigma \\
[\![\Box r]\!]_\varsigma &= (\Box(r\varsigma_{\mathbb{X}})) :: [\![r]\!]_\varsigma \\
[\![X_@]\!]_\varsigma &= tl(\varsigma(X)) \\
[\![let\ X{=}s\ in\ r]\!]_\varsigma &= [\![r]\!]_{\varsigma[X:=[\![s]\!]_\varsigma]} \\
[\![\mathsf{isapp}_A]\!]_\varsigma(\Box(r'r'') :: x) &= \top^{\mathcal{H}} \\
[\![\mathsf{isapp}_A]\!]_\varsigma(\Box(r) :: x) &= \bot^{\mathcal{H}} && (\forall r', r''.r \neq r'r'') \\
[\![C]\!]_\varsigma &= C^{\mathcal{H}}
\end{aligned}
$$

**Figure 4:** Denotational semantics of terms of the modal type system

—or more concisely this:

> "$\varsigma(X)$ is a pair of syntax and denotation."

**Definition 3.10.** Write $\varsigma_{\mathbb{X}}$ for the unknowns-substitution (Definition 2.15) such that

$$\varsigma_{\mathbb{X}}(X) = hd(\varsigma(X))$$

if $\varsigma(X)$ is defined, and $\varsigma_{\mathbb{X}}$ is undefined otherwise.

**Definition 3.11.** For each constant $C : A$ other than $\top$, $\bot$, and isapp fix some interpretation $C^{\mathcal{H}}$ which is an element $C^{\mathcal{H}} \in [\![A]\!]$. Suppose $\Gamma \vdash \varsigma$ and $\Gamma \vdash r : A$.

An **interpretation** of terms $[\![r]\!]_\varsigma$ is defined in Figure 4.

In Subsection 3.3 we discuss the design of $[\![r]\!]_\varsigma$, with examples. In Subsection 3.4 we prove some results about it.

### *3.3. Discussion of the denotation*

#### *3.3.1. About the term-formers*

The denotations of $\top$ and $\bot$ are as expected. To give a denotation to an atom $a$, we just look it up using $\varsigma$, also as expected. The definitions of $\lambda a{:}A.r$ and $r'r$ are also as standard.

As promised in Subsection 3.1, $[\![\Box r]\!]_\varsigma$ returns a pair of a syntax and its denotation.

$\mathsf{isapp}_A$ is there to illustrate concretely how we can express programming on syntax of box types: it takes a syntax argument and checks whether it is a syntactic application.[12] Of course many other such functions are possible, and if we want them we can add them as further constants (just as we might add $+$, $*$, and/or recursion as constants, given a type for numbers).

[12] We know non-trivial pattern-matching on applications exists in our meta-logic because our meta-logic is English; $[\![\mathsf{isapp}]\!]_\varsigma$ is a function on a set of syntax and we can define whatever operation we can define, on that set.

*3.3.2. Example: denotation of let $X{=}\Box(1+2)$ in $\Box\Box X_@$*

To illustrate how Figure 4 works, we calculate the denotation of $\mathit{let}\ X{=}\Box(1+2)\ \mathit{in}\ \Box\Box X_@$. We reason as follows, where for compactness and clarity we write $\varsigma$ for the valuation $[X{:=}\Box(1{+}2) :: 3]$:

$$\begin{aligned}
[\![\mathit{let}\ X{=}\Box(1{+}2)\ \mathit{in}\ \Box\Box X]\!]_\varnothing &= [\![\Box\Box X_@]\!]_{[X:=[\![\Box(1+2)]\!]_\varnothing]} \\
&= [\![\Box\Box X_@]\!]_\varsigma \\
&= \Box((\Box X_@)[X{:=}\Box(1{+}2)]) :: [\![\Box X_@]\!]_\varsigma \\
&= \Box\Box(1{+}2) :: [\![\Box X_@]\!]_\varsigma \\
&= \Box\Box(1{+}2) :: \Box(X_@[X{:=}\Box(1{+}2)]) :: [\![X_@]\!]_\varsigma \\
&= \Box\Box(1{+}2) :: \Box(1{+}2) :: [\![X_@]\!]_\varsigma \\
&= \Box\Box(1{+}2) :: \Box(1{+}2) :: \mathit{tl}(\Box(1{+}2) :: 3) \\
&= \Box\Box(1{+}2) :: \Box(1{+}2) :: 3
\end{aligned}$$

We leave it to the reader to verify that $[\![\Box(1{+}2)]\!]_\varnothing = \Box(1{+}2) :: 3$ and that $X_@[X{:=}\Box(1{+}2)] = 1{+}2$.

Note that '$1+2$' and '$\Box(1+2)$' are different; $1+2$ denotes 3 whereas $\Box(1+2)$ denotes the pair 'The syntax $1+2$, with associated extension 3'. In some very special cases where the set of possible denotations is rather small (finite or countable), the distinction between terms and their denotations can be hard to see, though it is still there. Usually sets of denotations are 'quite large' and sets of syntax are 'quite small', but sometimes this relationship is reversed: there are 'somewhat more' terms denoting numbers, than numbers[13] (but much fewer terms denoting functions from numbers to numbers than functions from numbers to numbers). See Corollary 3.15 and Example 3.16.

Note also the difference between the valuation $\varsigma = [X{:=}\Box(1{+}2) :: 3]$ and the substitution $[X{:=}\Box(1{+}2)]$. The first is a valuation because it maps $X$ to $[\![\Box\mathbb{N}]\!]$, the second is a substitution because it makes $X$ to a term of type $\Box\mathbb{N}$.

Sometimes a mapping can be both valuation and substitution; for instance $[a{:=}3]$ is a valuation ($a$ maps to an element of $[\![\mathbb{N}]\!]$), and is also a substitution.

*3.3.3. Why the natural version does not work*

Natural versions of Definitions 3.1 and 3.11 take

- the denotation of box type to be just boxed syntax rather than a pair of boxed syntax and denotation $[\![\Box A]\!] = \{\Box r \mid \varnothing \vdash \Box r : \Box A\}$, and
- $[\![\Box r]\!]_\varsigma = \Box(r\varsigma_{\mathbb{X}})$ and
- $[\![X_@]\!]_\varsigma = [\![r]\!]_\varnothing$ where $\varsigma(X) = \Box r$.

However, this seems not to work; $\varsigma(X)$ need not necessarily be a smaller term than $X$ so the 'definition' above is not inductive. This is not just a hypothetical issue: a term of the form $[\![\mathit{let}\ X{=}s\ \mathit{in}\ r]\!]_\varsigma$ may cause $\varsigma(X)$ to be equal to $[\![s]\!]_\varsigma$, and $s$ might generate syntax of any size.

The previous paragraph is not a mathematical proof; aside from anything else we have left the notion 'size of term' unspecified. The reader can experiment with different candidates: obvious 'subterm of', 'depth of', and 'number of symbols of' are all vulnerable to the problem described above, as is a more sophisticated notion of size which gives $X$ size $\omega$ the least infinite cardinal—since we can generate multiple copies of terms of the form $\mathit{let}\ X{=}r\ \mathit{in}\ s$, and even if this is closed it can contain bound copies of $X$.

[13] $6+5$ and $5+6$ denote the same number, whose calculation we leave as an exercise to the energetic reader.

*3.3.4. Example: denotation of $exp\,2$*

Recalling Subsection 2.3.4, we calculate the denotation of $[\![exp\,2]\!]_\varnothing$ where $exp$ is specified by:

$$exp\ 0 \Rightarrow \Box\lambda b{:}\mathbb{N}.1$$
$$exp\ (succ(n)) \Rightarrow let\ X{=}exp\ n\ in\ \Box(\lambda b{:}\mathbb{N}.b * (X_@ b)).$$

We sketch part of the calculation:

$$\begin{aligned}
[\![exp\,(succ\,(succ\,0))]\!]_\varnothing &= [\![let\ X{=}exp\,(succ\,0)\ in\ \Box(\lambda b{:}\mathbb{N}.b * (X_@ b))]\!]_\varnothing \\
&= [\![\Box(\lambda b{:}\mathbb{N}.b * (X_@ b))]\!]_{[X:=[\![exp\,(succ\,0)]\!]_\varnothing]} \\
&= \Box(\lambda b{:}\mathbb{N}.b * (X_@ b))[X{:=}hd[\![exp\,(succ\,0)]\!]_\varnothing] \\
&\quad :: [\![\lambda b{:}\mathbb{N}.b * (X_@ b)]\!]_{[X:=[\![exp\,(succ\,0)]\!]_\varnothing]} \\
&\cdots \\
&= \Box(\lambda b{:}\mathbb{N}.b * (\lambda b{:}\mathbb{N}.b * ((\lambda b{:}\mathbb{N}.1)b)b)) :: (x \in \mathbb{N} \mapsto x * x)
\end{aligned}$$

*3.3.5. Example: denotation of terms for axioms $(\mathbf{T})$ and $(\mathbf{4})$*

In Subsection 2.3.1 we considered the terms

$$\lambda a{:}\Box A.let\ X{=}a\ in\ X_@\ :\ \Box A{\to}A \quad \text{and}$$
$$\lambda a{:}\Box A.let\ X{=}a\ in\ \Box\Box X_@\ :\ \Box A{\to}\Box\Box A$$

which implement the modal logic axioms $(\mathbf{T})$ and $(\mathbf{4})$. We now describe their denotations, without working:

- $[\![\lambda a{:}\Box A.let\ X{=}a\ in\ X_@]\!]_\varnothing$ maps $\Box r :: tl \in [\![\Box A]\!]$ to $tl$.
- $[\![\lambda a{:}\Box A.let\ X{=}a\ in\ \Box\Box X_@]\!]_\varnothing$ maps $\Box r :: tl \in [\![\Box A]\!]$ to $\Box\Box r :: \Box r :: tl$.

*3.4. Results about the denotation*

We need a technical result and some notation for Proposition 3.13:

**Lemma 3.12.** *If $\Gamma \vdash \varsigma$ (Definition 3.8) then $\Gamma \vdash \varsigma_{\mathbb{X}}$ (Definition 2.22).*

*Proof.* If $X \notin dom(\varsigma)$ then $X \notin dom(\varsigma_{\mathbb{X}})$.

Suppose $X \in dom(\varsigma)$. By Definition 3.10 $\varsigma_{\mathbb{X}}(X) = hd(\varsigma(X))$. By Definition 3.8 $\varsigma_{\mathbb{X}}(X) \in [\![\Box A]\!]$ for some $A$. Unpacking Figure 3 this implies that $\varsigma_{\mathbb{X}}(X) = \Box r$ for some $\varnothing \vdash r : A$, and we are done. □

Proposition 3.13 relies on a dual role played by syntax in $\varsigma_{\mathbb{X}}$. It is coerced between denotation and syntax in $(\Box\mathbf{I})$, and 'in the other direction' in $(\mathbf{Ext})$. Proposition 3.13 expresses this important dynamic in the mathematics of the paper. Technically, the result is needed for the case of $(\Box\mathbf{I})$ in the proof of Theorem 3.14. Recall the notation $\Gamma|_U$ from Notation 2.12.

**Proposition 3.13.** *Suppose $\Gamma \vdash r : A$ and $\Gamma \vdash \varsigma$. Then $\Gamma|_{\mathbb{A}} \vdash r\,\varsigma_{\mathbb{X}} : A$.*
*($\varsigma_{\mathbb{X}}$ is defined in Definition 3.10; its action on $r$ is defined in Definition 2.18.)*

*Proof.* By Lemma 3.12 $\Gamma \vdash \varsigma_{\mathbb{X}}$. By Proposition 2.23 $\Gamma \vdash r\,\varsigma_{\mathbb{X}} : A$. By Lemma 2.19 $fa(r\varsigma_{\mathbb{X}}) = fa(r)$. Now it is a fact that $fa(r) \subseteq dom(\Gamma|_{\mathbb{A}})$, so by Proposition 2.13 $\Gamma|_{\mathbb{A}} \vdash r\,\varsigma_{\mathbb{X}} : A$ as required. □

**Theorem 3.14** (Soundness). *If* $\Gamma \vdash r : A$ *and* $\Gamma \vdash \varsigma$ *then* $[\![r]\!]_\varsigma$ *is defined and* $[\![r]\!]_\varsigma \in [\![A]\!]$.

*Proof.* By induction on the derivation of $\Gamma \vdash r : A$. Most of the rules follow by properties of sets and functions. We consider the interesting cases:

- *Rule* ($\Box\mathbf{I}$). Suppose $\Gamma \vdash r : A$ and $fa(r){=}\varnothing$ so that by ($\Box\mathbf{I}$) $\Gamma \vdash \Box r : A$.
  Suppose $\Gamma \vdash \varsigma$. Then by inductive hypothesis $[\![r]\!]_\varsigma \in [\![A]\!]$. Also, by Proposition 3.13 $\varnothing \vdash r\varsigma_{\mathbb{X}} : A$
  It follows by Definition 3.1 that
  $$[\![\Box r]\!]_\varsigma = (\Box(r\varsigma_{\mathbb{X}})) :: [\![r]\!]_\varsigma \in [\![\Box A]\!]$$
  as required.
- *Rule* ($\Box\mathbf{E}$). Suppose $\Gamma, X{:}\Box A \vdash r : B$ and $\Gamma \vdash s : \Box A$ so that by ($\Box\mathbf{E}$) $\Gamma \vdash \mathit{let}\ X{=}s\ \mathit{in}\ r : A$.
  Suppose $\Gamma \vdash \varsigma$. By inductive hypothesis for $\Gamma \vdash s : \Box A$ we have $[\![s]\!]_\varsigma \in [\![\Box A]\!]$ and so there is some term $s'$ and some $x \in [\![A]\!]$ such that $(\Box s') :: x = [\![s]\!]_\varsigma$ and $\varnothing \vdash \Box s' : \Box A$. Unpacking Definition 3.8, $\Gamma, X{:}\Box A \vdash \varsigma[X{:=}(\Box s') :: x]$. By inductive hypothesis for $\Gamma, X{:}\Box A \vdash r : B$ we have
  $$[\![r]\!]_{\varsigma[X:=(\Box s') :: x]} \in [\![B]\!]$$
  and using Definition 3.11 we have
  $$[\![\mathit{let}\ X{=}s\ \mathit{in}\ r]\!]_\varsigma = [\![r]\!]_{\varsigma[X:=(\Box s') :: x]} \in [\![B]\!]$$
  as required.
- *Rule* ($\mathbf{Ext}$). By ($\mathbf{Ext}$) $\Gamma, X{:}\Box A \vdash X_@{:}\Box A$.
  Suppose $\Gamma, X{:}\Box A \vdash \varsigma$. Unpacking Definition 3.8, this means that $\varsigma(X) = (\Box s') :: x$ for some $s'$ and $x$ such that $\varnothing \vdash \Box s' : \Box A$ and $x \in [\![A]\!]$. From Definition 3.11 $[\![X_@]\!]_\varsigma = x \in [\![A]\!]$ as required.
- *Rule* ($\mathbf{Hyp}$). Suppose $\Gamma, a{:}A \vdash \varsigma$. By Definition 3.8 this means that $\varsigma(a) \in [\![A]\!]$. By Definition 3.11 $[\![a]\!]_\varsigma = \varsigma(a)$. The result follows.

□

**Corollary 3.15.** *There is no term* $s$ *such that* $\varnothing \vdash s : (\mathbb{N}{\to}\mathbb{N}){\to}\Box(\mathbb{N}{\to}\mathbb{N})$ *is typable and such that the map* $\big(x{\in}\mathbb{N}^{\mathbb{N}} \mapsto hd([\![s]\!]_\varnothing\, x)\big) \in hd([\![\Box(\mathbb{N}{\to}\mathbb{N})]\!])^{[\![\mathbb{N}\to\mathbb{N}]\!]}$ *is injective.*

*Proof.* $[\![\mathbb{N}{\to}\mathbb{N}]\!]$ is an uncountable set whereas $hd([\![\Box(\mathbb{N}{\to}\mathbb{N})]\!]) = \{r \mid \varnothing \vdash r : \mathbb{N}{\to}\mathbb{N}\}$ is countable. The result follows from Theorem 3.14. □

**Example 3.16.** By Corollary 3.15 there can be no term representing a function which reifies an element of $[\![A]\!]$ to corresponding syntax.

Of course, there might be a term which reifies those elements of $[\![A]\!]$ that are representable by syntax. For specific 'sufficiently small' $A$, this might even include all of $[\![A]\!]$.

For example, if $A = \mathbb{N}$ then the following function does the job:

$$\begin{array}{c} \mathit{reifyNat}\ 0 \Rightarrow \Box 0 \\ \mathit{reifyNat}\ (\mathsf{succ}(n)) \Rightarrow \mathit{let}\ X{=}\mathit{reifyNat}(n)\ \mathit{in}\ \Box(X_{@}{+}1). \end{array}$$

**Remark 3.17.** Similar arguments to those used in Corollary 3.15 and Example 3.16 also justify why the Haskell programming language has a *Show* function for certain types, but not for function types.[14] We chose full function-spaces in Figure 4, so that the models for which we prove soundness in Theorem 3.14 would be large, and we did that so that the proof of Corollary 3.15 would become relatively easy. Careful consideration has gone into the precise designs of $[\![B]\!]^{[\![A]\!]}$ and $[\![\Box A]\!]$.

We will later on in Corollary 6.11 prove a similar result for the contextual system, and then later still in Corollary 7.8 surprisingly leverage this to a result which even works for functions to all of $[\![\Box(\mathbb{N}{\to}\mathbb{N})]\!]$ rather than just to the (much smaller) $hd([\![\Box(\mathbb{N}{\to}\mathbb{N})]\!])$.

## 4. Reduction

We have Theorem 3.14 (soundness) and Corollary 3.15 (impossibility in general of reifying denotation to syntax). The other major property of interest is that typing and denotation are consistent with a natural notion of reduction on terms.

So we now turn our attention to the lemmas leading up to Proposition 4.10 and Theorem 4.11.

### *4.1. Results concerning substitution on atoms*

Recall from Definition 2.18 the definition of the atoms-substitution action. Lemma 4.1 is a counterpart to Proposition 3.13. We had to prove Proposition 3.13 earlier because calculating the denotation $[\![\Box r]\!]_\varsigma$ in Figure 4 involves calculating $r\varsigma_{\mathbb{X}}$ (an unknowns-substitution applied to a term).[15] Now we are working towards reduction, and $\beta$-reduction can generate atoms-substitution, so we need Lemma 4.1.

**Lemma 4.1.** *Suppose* $\Gamma, a{:}B \vdash r : A$ *and* $\Gamma \vdash s : B$. *Then* $\Gamma \vdash r[a{:=}s] : A$.

*Proof.* By a routine induction on the typing of $r$. We consider three cases:

- *The case of* $(\Box\mathbf{I})$. Suppose $\Gamma, a{:}B \vdash r : A$ and $fa(r){=}\varnothing$ so that $\Gamma, a{:}B \vdash \Box r : \Box A$ by $(\Box\mathbf{I})$. But then by Lemma 2.21 $r[a{:=}s] = r$, and the result follows from Proposition 2.13.
- *The case of* $(\mathbf{Ext})$ is similar to that of $(\Box\mathbf{I})$.
- *The case of* $(\Box\mathbf{E})$. Using the fact from Definition 2.18 that

$$(\mathit{let}\ X{=}s'\ \mathit{in}\ r)[a{:=}s] \ = \ \mathit{let}\ X{=}s'[a{:=}s]\ \mathit{in}\ r[a{:=}s].$$

□

[14] See haskell.org/haskellwiki/Show_instance_for_functions, retrieved on January 20, 2012.

[15] In the contextual system, calculating the denotation will involve atoms-substitution as well.

**Lemma 4.2.** *Suppose* $\Gamma, a{:}B \vdash r : A$ *and* $\Gamma \vdash s : B$*, and suppose* $\Gamma \vdash \varsigma$*. Then* $[\![r[a{:=}s]]\!]_\varsigma = [\![r]\!]_{\varsigma[a:=[\![s]\!]_\varsigma]}$.

*Proof.* By a routine induction on the derivation of $\Gamma, a{:}B \vdash r : A$ (Figure 1). We consider three cases:

- *The case of* $(\Box\mathbf{I})$. We use Lemma 2.21 and Proposition 2.13 (as in the case of $(\Box\mathbf{I})$ in the proof of Lemma 4.1).
- *The case of* $(\mathbf{Ext})$. By $(\mathbf{Ext})$ $\Gamma, a{:}B, X{:}A \vdash X_@ : A$. By definition $X_@[a{:=}s] = X_@$. We use Proposition 2.13.
- *The case of* $(\mathbf{Hyp})$ *for* $a$. By $(\mathbf{Hyp})$ $\Gamma, a{:}B \vdash a : B$. By assumption $\Gamma, a{:}B \vdash \varsigma$ so unpacking Definition 3.8, $\varsigma(a) \in [\![B]\!]$. By Figure 4 $\varsigma(a) = [\![a]\!]_\varsigma$, and we are done.

□

Proposition 4.3 can be viewed as a denotational counterpart of Proposition 2.13:

**Proposition 4.3.** *Suppose* $\Gamma \vdash r : A$ *and* $\Gamma \vdash \varsigma$ *and* $\Gamma \vdash \varsigma'$*. Suppose* $\varsigma(a) = \varsigma'(a)$ *for every* $a \in fa(r)$ *and* $\varsigma(X) = \varsigma'(X)$ *for every* $X \in fa(r)$.
*Then* $[\![r]\!]_\varsigma = [\![r]\!]_{\varsigma'}$.

*Proof.* By a routine induction on $r$. □

**Lemma 4.4.** *Suppose* $\Gamma, a{:}A \vdash r : B$ *and* $\Gamma \vdash s : A$*, and suppose* $\Gamma \vdash \varsigma$*. Then*

$$[\![(\lambda a{:}A.r)s]\!]_\varsigma = [\![r]\!]_{\varsigma[a:=[\![s]\!]_\varsigma]}.$$

*Proof.* We unpack the cases of $\lambda$ and application in Definition 3.11. □

*4.2. Results concerning substitution on unknowns*

**Lemma 4.5.** *Suppose* $\Gamma \vdash (let\ X{=}s\ in\ r) : A$ *and* $\Gamma \vdash \varsigma$*. Then*

$$[\![let\ X{=}s\ in\ r]\!]_\varsigma = [\![r]\!]_{\varsigma[X:=[\![s]\!]_\varsigma]}.$$

*Proof.* We just unpack the clause for $let\ X{=}s\ in\ r$ in Figure 4 (well-definedness is from Theorem 3.14). □

**Lemma 4.6.** *Suppose* $\theta$ *is an unknowns-substitution (Definition 2.15). Suppose* $X \notin dom(\theta)$ *and suppose* $fu(\theta(Z)) = \varnothing$ *for every* $Z \in dom(\theta)$.
*Then* $r[X{:=}\Box s]\theta = r\theta[X{:=}\Box(s\theta)]$.

*Proof.* By a routine induction on $r$. The interesting case is $X_@$, for which it is easy to check that:

$$X_@\theta[X{:=}(\Box s)\theta] = s\theta \quad \text{and} \quad X_@[X{:=}\Box s]\theta = s\theta.$$

□

**Lemma 4.7.** *Suppose* $\Gamma, X{:}\Box B \vdash r : A$ *and* $\Gamma \vdash \Box s : \Box B$*, and suppose* $\Gamma \vdash \varsigma$*. Then* $[\![r[X{:=}\Box s]]\!]_\varsigma = [\![r]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]}$.

*Proof.* By induction on the derivation of $\Gamma, X{:}\Box B \vdash r : A$.

$r[a{:=}s]$ and $r[X{:=}s]$ from Definition 2.18.

$$\frac{}{(\lambda a{:}A.r)r' \to_\beta r[a{:=}r']}\ (\beta) \qquad \frac{}{let\ X{=}\Box s\ in\ r \to_\beta r[X{:=}\Box s]}\ (\beta_\Box)$$

$$\frac{r \to_\beta r' \quad s \to_\beta s'}{rs \to_\beta r's'}\ (\mathbf{cnga}) \qquad \frac{r \to_\beta r'}{\lambda a{:}A.r \to_\beta \lambda a{:}A.r'}\ (\mathbf{cngl})$$

$$\frac{r \to_\beta r' \quad s \to_\beta s'}{let\ X{=}s\ in\ r \to_\beta let\ X{=}s'\ in\ r'}\ (\mathbf{cnge})$$

$$\frac{}{\mathsf{isapp}\,\Box(r'r) \to_\beta \top}\ (\mathbf{isapp}\top) \qquad \frac{(r \text{ not of the form } r'r)}{\mathsf{isapp}\,\Box(r) \to_\beta \bot}\ (\mathbf{isapp}\bot)$$

**Figure 5:** Reduction rules for the modal system

- *The case of* $(\Box\mathbf{I})$. Suppose $\Gamma, X{:}\Box B \vdash r : A$ and $fa(r) = \varnothing$ so that by $(\Box\mathbf{I})$ $\Gamma, X{:}\Box B \vdash \Box r : \Box A$. We sketch the necessary reasoning:

$$\begin{aligned}
[\![(\Box r)[X{:=}\Box s]]\!]_\varsigma &= [\![\Box(r[X{:=}\Box s])]\!]_\varsigma && \text{Definition 2.18}\\
&= \Box(r[X{:=}\Box s])\varsigma_{\mathbb{X}} :: [\![r[X{:=}\Box s]]\!]_\varsigma && \text{Figure 4}\\
&= \Box(r[X{:=}\Box s])\varsigma_{\mathbb{X}} :: [\![r]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]} && \text{Ind. Hyp.}\\
&= \Box(r\varsigma_{\mathbb{X}}[X{:=}\Box(s\varsigma_{\mathbb{X}})]) :: [\![r]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]} && \text{Lemma 4.6}\\
[\![\Box r]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]} &= (\Box r)(\varsigma[X{:=}[\![\Box s]\!]_\varsigma])_{\mathbb{X}} :: [\![r]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]} && \text{Figure 4}\\
&= (\Box r)\varsigma_{\mathbb{X}}[X{:=}\Box(s\varsigma_{\mathbb{X}})] :: [\![r]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]} && \text{Figure 4}\\
&= \Box(r\varsigma_{\mathbb{X}}[X{:=}\Box(s\varsigma_{\mathbb{X}})]) :: [\![r]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]} && \text{Definition 2.18}
\end{aligned}$$

- *The case of* $(\mathbf{Ext})$ *for* $X$. By $(\mathbf{Ext})$ $\Gamma, X{:}\Box B \vdash X_@ : B$. Then we reason as follows:

$$\begin{aligned}
[\![X_@]\!]_{\varsigma[X:=[\![\Box s]\!]_\varsigma]} &= tl([\![\Box s]\!]_\varsigma) && \text{Figure 4}\\
&= [\![s]\!]_\varsigma && \text{Figure 4}\\
[\![X_@[X{:=}\Box s]]\!]_\varsigma &= [\![s]\!]_\varsigma && \text{Definition 2.18}
\end{aligned}$$

□

*4.3. Reduction*

**Definition 4.8.** Define $\beta$**-reduction** $r \to_\beta r'$ inductively by the rules in Figure 5.

**Remark 4.9.** We do not have a rule that if $r \to_\beta r'$ then $\Box r \to_\beta \Box r'$. This would be wrong because it does not respect the integrity of the syntax of a term; syntax, in denotation, does not inherently reduce.

We do however allow reduction under a $\lambda$. This is purely a design choice; we are interested in making as many terms as possible $\beta$-convertible, and less immediately interested in this

paper in finding nice notions of $\beta$-normal form. If we did not have a denotational semantics then we might have to be more sensitive to such questions (because normal forms are important for consistency)—because we *do* have a denotational semantics, we obtain consistency via soundness and the precise notion of normal form is not so vital.

**Proposition 4.10.** *If* $\Gamma \vdash r : A$ *and* $r \to r'$ *then* $\Gamma \vdash r' : A$.

*Proof.* By a routine induction on $r$. The case of $(\beta)$ $(\lambda a{:}A.r)r' \to_\beta r[a{:=}r']$ follows by Lemma 4.1; that of $(\beta_\Box)$ follows by Proposition 3.13. □

**Theorem 4.11.** *Suppose* $\Gamma \vdash r : A$ *and* $\Gamma \vdash \varsigma$. *Suppose* $r \to_\beta r'$. *Then* $[\![r]\!]_\varsigma = [\![r']\!]_\varsigma$.

*Proof.* By induction on the derivation of $r \to_\beta r'$.

- *The case of* $(\beta)$ follows by Lemmas 4.2 and 4.4.
- *The case of* $(\beta_\Box)$ follows by Lemmas 4.5 and 4.7.

□

## 5. Syntax and typing of the system with contextual types

The modal type system is beautiful, but is a little too weak for some applications. The issue is that $X$ ranges over *closed* syntax. If we are working under some $\lambda$-abstractions, we may well find this limiting; we want to work with *open* syntax so that we can refer to the enclosing binder. This really matters, because it affects the programs we can write. For instance in the example of exponentiation from Subsection 2.3.4, the issue of working under a $\lambda$-abstraction forced us to generate unwanted $\beta$-redexes.

The contextual system is one way to get around this. Syntax is still closed, but the notion of closure is liberalised by introducing a context into the modality; to see the critical difference, compare the $([]\mathbf{I})$ rule in Figure 6 with the $(\Box\mathbf{I})$ rule from Figure 1. The interested reader can see how this allows us to write a nicer program for exponentiation, which does not generate $\beta$-redexes, in Subsection 6.2.2.

### 5.1. Syntax of the contextual system

**Notation 5.1.** The contextual system needs many vectors of types and atoms-and-types. For clarity, we write these vectors subscripted, for instance:

- $(a_i{:}A_i)_1^n$ is shorthand for $\{a_1{:}A_1, \ldots, a_n{:}A_n\}$.
- $[A_i]_1^n A$ is shorthand for $[A_1, \ldots, A_n]A$.
- $(A_i)_1^n \to A$ is shorthand for $A_1 \to (A_2 \to \ldots (A_n \to A))$.
- $\{a_i\}_1^n$ is shorthand for $\{a_1, \ldots, a_n\}$.
- $[a_i{:=}x_i]_1^n$ will be shorthand for the map taking $a_i$ to $x_i$ for $1 \leq i \leq n$ and undefined elsewhere (Definition 5.10).

We may omit the interval where it is understood or irrelevant, so for instance $\{a_i\}$ and $\{a_i\}_i$ are both shorthand for the same thing: "$\{a_1, \ldots, a_n\}$ for some $n$ whose precise value we will never need to reference", and $(A_i) \to A$ is shorthand for "$(A_i)_1^n \to A$ for some $n$ whose precise value we will never need to reference".

We take atoms and unknowns as in Definition 2.1.

**Definition 5.2.** Define **types** inductively by:

$$A ::= o \mid \mathbb{N} \mid A \to A \mid [A_i]_1^n A$$

$o$ (truth-values), $\mathbb{N}$ (numbers), and $A{\to}B$ (functions) are as in Definition 2.2. $[A_i]_1^n A$ is a *contextual type*. Think of this as generalising the modal types of Definition 5.2 by 'allowing bindings in the box'.

**Definition 5.3.** Fix a set of **constants** $C$ to each of which is assigned a type $type(C)$. We write $C : A$ as shorthand for '$C$ is a constant and $type(C) = A$'. We insist that constants include the following:

$$\bot : o \qquad \top : o \qquad \mathsf{isapp}_A : ([\,]A){\to}o$$

We may omit the type subscripts where they are clear from context or do not matter.

**Definition 5.4.** Define **terms** inductively by:

$$r ::= C \mid a \mid \lambda a{:}A.r \mid rr \mid [a_i{:}A_i]r \mid X@(r_i)_1^n \mid \mathit{let}\ X{=}r\ \mathit{in}\ r$$

**Remark 5.5.** The syntax of the modal type system in Definition 2.6 injects naturally into that of Definition 5.4, if we map $\Box$- to $[\,]$- (the empty context) and -$_@$ to -$@()$.

The important extra complexity is in $X@(r_i)_1^n$; when $X$ is instantiated by a substitution $\theta$, this triggers an atoms-substitution of the form $[a_i{:=}r_i]_1^n$. See Definition 5.11.

**Definition 5.6.** Define **free atoms** $fa(r)$ and **free unknowns** $fu(r)$ by:

$$\begin{array}{rl} fa(C) = \varnothing & \qquad fa(a) = \{a\} \\ fa(\lambda a{:}A.r) = fa(r) \setminus \{a\} & \qquad fa(rs) = fa(r) \cup fa(s) \\ fa([a_i{:}A_i]_1^n r) = fa(r) \setminus \{a_1, \ldots, a_n\} & \qquad fa(\mathit{let}\ X{=}s\ \mathit{in}\ r) = fa(r) \cup fa(s) \\ fa(X@(s_i)_i) = \bigcup_i fa(s_i) & \end{array}$$

$$\begin{array}{rl} fu(C) = \varnothing & \qquad fu(a) = \varnothing \\ fu(\lambda a{:}A.r) = fu(r) & \qquad fu(rs) = fu(r) \cup fu(s) \\ fu([a_i{:}A_i]r) = fu(r) & \qquad fu(\mathit{let}\ X{=}s\ \mathit{in}\ r) = (fu(r){\setminus}\{X\}) \cup fu(s) \\ fu(X@(s_i)_i) = \{X\} \cup \bigcup_i fu(s_i) & \end{array}$$

**Definition 5.7.** We take $a$ to be bound in $r$ in $\lambda a{:}A.r$ and $a_1, \ldots, a_n$ to be bound in $r$ in $[a_i{:}A]_1^n r$, and we take $X$ to be bound in $r$ in $\mathit{let}\ X{=}s\ \mathit{in}\ r$. We take syntax up to $\alpha$-equivalence as usual. For example:

- $\lambda a{:}A.a = \lambda b{:}A.b$
- $\lambda a{:}A.[b{:}B]((X@(b))a) = \lambda b{:}A.[a{:}B]((X@a)b) \neq \lambda b{:}A.[b{:}B]((X@(b))b)$
- $\mathit{let}\ X{=}[a{:}A]a\ \mathit{in}\ (X@(b))$
  $= \mathit{let}\ Y{=}[a{:}A]a\ \mathit{in}\ (Y@(b))$
  $= \mathit{let}\ Y{=}[b{:}A]b\ \mathit{in}\ (Y@(b))$

$$\frac{}{\Gamma, a : A \vdash a : A}\ (\mathbf{Hyp}) \qquad \frac{}{\Gamma \vdash C : \mathit{type}(C)}\ (\mathbf{Const})$$

$$\frac{\Gamma, a{:}A \vdash r : B}{\Gamma \vdash (\lambda a{:}A.r) : A \to B}\ (\to\mathbf{I}) \qquad \frac{\Gamma \vdash r' : A \to B \quad \Gamma \vdash r : A}{\Gamma \vdash r'r : B}\ (\to\mathbf{E})$$

$$\frac{\Gamma, (a_i{:}A_i)_i \vdash r : A \quad (\mathit{fa}(r) \subseteq \{a_i\}_i)}{\Gamma \vdash [a_i{:}A_i]r : [A_i]A}\ ([]\mathbf{I}) \qquad \frac{\Gamma, X{:}[A_i]A \vdash r{:}B \ \ \Gamma \vdash s{:}[A_i]A}{\Gamma \vdash \mathit{let}\ X{=}s\ \mathit{in}\ r : B}\ ([]\mathbf{E})$$

$$\frac{\Gamma, X : [A_i]_1^n A \vdash r_j : A_j \quad (1 \leq j \leq n)}{\Gamma, X : [A_i]_1^n A \vdash X@(r_i)_1^n : A}\ (\mathbf{Ext})$$

**Figure 6:** Contextual modal type theory typing rules

*5.2. Typing for the contextual system*

**Definition 5.8.** A **typing** is a pair $a : A$ or $X : [A_i]_i A$. A **typing context** $\Gamma$ is a finite partial function from $\mathbb{A} \cup \mathbb{X}$ to types (as in Definition 2.10, except that unknowns have contextual types instead of just box types $\Box A$).

A **typing sequent** is a tuple $\Gamma \vdash r : A$ of a typing context, a term, and a type.

Define the **valid typing sequents** of the contextual modal type system by the rules in Figure 6.

Recall the notation $\Gamma|_U$ from Notation 2.12. Proposition 5.9 repeats Proposition 2.13 for the contextual system:

**Proposition 5.9.** *If* $\Gamma \vdash r : A$ *and* $\Gamma'|_{\mathit{fu}(r) \cup \mathit{fa}(r)} = \Gamma|_{\mathit{fu}(r) \cup \mathit{fa}(r)}$ *then* $\Gamma' \vdash r : A$.

*Proof.* By a routine induction on $r$. □

*5.3. Substitution*

Definition 5.10 reflects Definition 2.15 for the richer syntax of terms:

**Definition 5.10.** An **(atoms-)substitution** $\sigma$ is a finite partial function from atoms $\mathbb{A}$ to terms. $\sigma$ will range over atoms-substitutions.

Write $\mathit{dom}(\sigma)$ for the set $\{a \mid \sigma(a) \text{ defined}\}$

Write $\mathit{id}$ for the **identity** substitution, such that $\mathit{dom}(\sigma) = \varnothing$.

Write $[a_i{:=}t_i]_1^n$ for the map taking $a_i$ to $t_i$ for $1 \leq i \leq n$ and undefined elsewhere.

An **(unknowns-)substitution** $\theta$ is a finite partial function from unknowns $\mathbb{X}$ to terms such that if $\theta(X)$ is defined then $\theta(X) = [a_i{:}A_i]_1^n r$ for some $r$ with $\mathit{fa}(r) \subseteq \{a_1, \ldots, a_n\}$ (so $\mathit{fa}(\theta(X)) = \varnothing$ for every $X \in \mathit{dom}(\theta)$).

$\theta$ will range over unknowns-substitutions.

We write $\mathit{dom}(\theta)$, $\mathit{id}$, and $[X_i{:=}t_i]_1^n$ just as for atoms-substitutions (we will be most interested in the case that $n = 1$).

$$
\begin{array}{rrl}
C\sigma = C & a\sigma = \sigma(a) & (a \in dom(\sigma)) \\
(rs)\sigma = (r\sigma)(s\sigma) & a\sigma = a & (a \notin dom(\sigma)) \\
(X@(r_i)_i)\sigma = X@(r_i\sigma)_i & (\lambda c{:}A.r)\sigma = \lambda c{:}A.(r\sigma) & (c \notin fa(\sigma)) \\
(let\ Y{=}s\ in\ r)\sigma = let\ Y{=}s\sigma\ in\ r\sigma & ([a_i{:}A_i]r)\sigma = [a_i{:}A_i](r\sigma) & (a_i \notin fa(\sigma) \text{ all } i) \\
 & & \\
C\theta = C & a\theta = a & \\
(rs)\theta = (r\theta)(s\theta) & (X@(r_i))\theta = s'[a_i{:=}r_i\theta] & (\theta(X){=}[a_i{:}A_i]s') \\
([a_i{:}A_i]r)\theta = [a_i{:}A_i](r\theta) & (X@(r_i))\theta = X@(r_i\theta) & (X \notin dom(\theta)) \\
(\lambda c{:}A.r)\theta = \lambda c{:}A.(r\theta) & (let\ Y{=}s\ in\ r)\theta = let\ Y{=}s\theta\ in\ r\theta & (Y \notin fu(\theta))
\end{array}
$$

**Figure 7:** Substitution actions for atoms and unknowns (contextual syntax)

We also reflect Definition 2.16 and write $fa(\sigma)$ and $fu(\theta)$, but using the notions of 'free atoms' and 'free unknowns' from Definition 5.6. The definition is formally identical:

$$
\begin{array}{l}
fa(\sigma) = dom(\sigma) \cup \{fa(\sigma(a)) \mid a \in dom(\sigma)\} \quad \text{and} \\
fu(\theta) = dom(\theta) \cup \{fu(\theta(X)) \mid X \in dom(\theta)\}
\end{array}
$$

**Definition 5.11.** Define substitution actions $r\sigma$ and $r\theta$ by the rules in Figure 7.

**Remark 5.12.** The capture-avoidance side-conditions of Definition 5.11 (of the form '$* \notin fa(\sigma)$' or '$* \notin fu(\theta)$') can be guaranteed by $\alpha$-renaming.

The case of $(X@(r_i)_1^n)\theta$ may generate an atoms-substitution $s'[a_i{:=}r_i\theta]$. In the case $n = 0$ it is easy to check that this collapses the the previous case of $X_@$ from Figure 2, because the substitution becomes empty.

Note that $r\sigma$ and $r\theta$ are both defined by straightforward induction on $r$; even if $s'$ is larger than $X@(r_i)_1^n$—so that apparently the inductive quantity gets larger—still $s'[a_i{:=}r_i\theta]$ is an *atoms*-substitution, not an unknowns-substitution. Meanwhile, atoms-substitution never triggers unknowns-substitution so circularity is avoided.

Strictly speaking the case of $(X@(r_i)_1^n)\theta$ introduces a partiality into the notion of substitution action; we assume that $\theta(X) = [a_i{:}A_i]_1^m s'$ and *for this to make sense* it must be that $n = m$; if $n \neq m$ then the definition is not well-defined. However, for well-typed syntax this is guaranteed not to happen, and since this is the only case we will care about, we will never notice this.

We conclude this section with some important definitions and results about the interaction of substitution and typing, which will be needed for Theorem 6.10.

Definition 5.13 reflects Definition 2.22, but we need $\Gamma \vdash \sigma$ as well as $\Gamma \vdash \theta$:

**Definition 5.13.** Write $\Gamma \vdash \theta$ when if $X \in dom(\theta)$ then $X{:}[A_i]A \in \Gamma$ for some $[A_i]A$ and $\Gamma \vdash \theta(X) : [A_i]A$.

Similarly write $\Gamma \vdash \sigma$ when if $a \in dom(\sigma)$ then $a{:}A \in \Gamma$ for some $A$ and $\Gamma \vdash \sigma(a) : A$.

**Lemma 5.14.** *$fa(r\theta) = fa(r)$ where $r\theta$ is defined.*

$$
\begin{aligned}
[\![o]\!] &= \{\top^{\mathcal{H}}, \bot^{\mathcal{H}}\} \\
[\![\mathbb{N}]\!] &= \{0, 1, 2, \dots\} \\
[\![A{\to}B]\!] &= [\![B]\!]^{[\![A]\!]} \\
[\![[A_i]_1^n A]\!] &= \{[a_i{:}A_i]_1^n r \mid \varnothing \vdash [a_i{:}A_i]_1^n r : [A_i]_1^n A\} \times [\![A]\!]^{\Pi_{i=1}^n [\![A_i]\!]}
\end{aligned}
$$

**Figure 8:** Denotational semantics for CMTT types

*Proof.* By a routine induction on $r$ using our assumption of Definition 5.10 that if $X \in dom(\theta)$ then $fa(\theta(X)) = \varnothing$. □

Lemma 5.15 reflects Lemma 4.1. However, unlike was the case for the modal system, it is needed for Proposition 5.16/2.23 because the case of $(X@(r_i))\theta$ in Definition 5.11 triggers an atoms-substitution.

**Lemma 5.15.** *Suppose* $\Gamma \vdash r : A$ *and* $\Gamma \vdash \sigma$. *Then* $\Gamma \vdash r\sigma : A$.

*Proof.* By routine inductions on the derivation of $\Gamma \vdash r : A$. □

Proposition 5.16 reflects Proposition 2.23 and is needed for soundness of the denotation. The proof is significantly more complex, because of the atoms-substitution that can be introduced by the case of $(X@(s_j))\theta$. This is handled in the proof below using Lemma 5.15.

**Proposition 5.16.** *Suppose* $\Gamma \vdash r : A$ *and* $\Gamma \vdash \theta$. *Then* $\Gamma \vdash r\theta : A$.

*Proof.* By a routine induction on the typing of $r$. We consider two cases:

- *The case of* $([]\mathbf{I})$. Suppose $\Gamma, (b_j{:}B_j) \vdash r : A$ and $fa(r){\subseteq}\{b_j \mid j\}$ so that $\Gamma \vdash [b_j{:}B_j]r : [B_j]A$ by $([]\mathbf{I})$. By inductive hypothesis $\Gamma, (b_j{:}B_j) \vdash r\theta : A$. By Lemma 5.14 $fu(r\theta){\subseteq}\{b_j \mid j\}$. We use $([]\mathbf{I})$ and the fact that $([b_j{:}B_j]r)\theta = [b_j{:}B_j](r\theta)$.
- *The case of* $(\mathbf{Ext})$ *for* $X \in dom(\theta)$. Suppose $\Gamma, X{:}[A_j]_1^m A \vdash s_j : A_j$ for each $1{\leq}j{\leq}m$ so that by $(\mathbf{Ext})$ $\Gamma, X{:}[A_j]_1^m A \vdash X@(s_j)_1^m{:}A$. By inductive hypothesis $\Gamma \vdash s_j\theta : A_j$ for each $j$. By assumption $\theta(X) = [a_j{:}A_j]r'$ for some $r'$ such that $(a_j{:}A_j)_1^m \vdash r' : A$. By Lemma 5.15 $\Gamma \vdash r'[a_j{:=}s_j\theta] : A$. By the definitions $(X@(s_j))\theta = r'[a_j{:=}s_j\theta]$, so we are done.

□

We could now give a theory of reduction for the contextual system, following the definition of reduction for the modal system in Subsection 4.3. However, we will skip over this; the interested reader is referred elsewhere [NP05]. What is more interesting, from the point of view of this paper, is the models we define for the contextual system, which we come to next.

## 6. Contextual models

### *6.1. Denotational semantics*

Definition 6.1 is like Definition 3.1, except that instead of box types, we have contextual types:

**Definition 6.1.** Define $[\![A]\!]$ the **interpretation** of types by induction in Figure 8.

$$
\begin{aligned}
[\![\top]\!]_\varsigma &= \top^{\mathcal{H}} \\
[\![\bot]\!]_\varsigma &= \bot^{\mathcal{H}} \\
[\![a]\!]_\varsigma &= \varsigma(a) \\
[\![\lambda a{:}A.r]\!]_\varsigma &= (x{\in}[\![A]\!] \mapsto [\![r]\!]_{\varsigma[a:=x]}) \\
[\![r'r]\!]_\varsigma &= [\![r']\!]_\varsigma\,[\![r]\!]_\varsigma \\
[\![[a_i{:}A_i]_1^n r]\!]_\varsigma &= [a_i{:}A_i]_1^n(r\,\varsigma_{\mathbb{X}}) :: \left((x_i{\in}[\![A_i]\!])_1^n \mapsto [\![r]\!]_{\varsigma[a_i:=x_i]_1^n}\right) \\
[\![X@(r_i)_1^n]\!]_\varsigma &= tl(\varsigma(X))\,([\![r_i]\!]_\varsigma)_1^n \\
[\![let\ X{=}s\ in\ r]\!]_\varsigma &= [\![r]\!]_{\varsigma[X:=[\![s]\!]_\varsigma]} \\
[\![\mathsf{isapp}_A]\!]_\varsigma([a_i{:}A_i](r'r)) &= \top^{\mathcal{H}} \\
[\![\mathsf{isapp}_A]\!]_\varsigma([a_i{:}A_i](r)) &= \bot^{\mathcal{H}} \quad \text{otherwise} \\
[\![C]\!]_\varsigma &= C^{\mathcal{H}}
\end{aligned}
$$

**Figure 9:** Denotational semantics for terms of the contextual system

**Definition 6.2.** A **valuation** $\varsigma$ is a finite partial function on $\mathbb{A} \cup \mathbb{X}$.

We define $\varsigma[X{:=}x]$ and $\varsigma[a{:=}x]$ just as in Definition 3.7.

**Definition 6.3.** Write $\varsigma_{\mathbb{X}}$ for the substitution (Definition 5.10) such that $\varsigma_{\mathbb{X}}(X) = hd(\varsigma(X))$ if $\varsigma(X)$ is defined, and $\varsigma_{\mathbb{X}}(X)$ is undefined if $\varsigma(X)$ is undefined.

**Definition 6.4.** If $\Gamma$ is a typing context then write $\Gamma \vdash \varsigma$ when:

1. $dom(\Gamma) = dom(\varsigma)$.
2. If $a \in dom(\varsigma)$ then $\Gamma(a) = A$ for some $A$ and $\varsigma(a) \in [\![A]\!]$.
3. If $X \in dom(\varsigma)$ then $\Gamma(X) = [A_i]A$ and $\varsigma(X) \in [\![[A_i]A]\!]$.

**Remark 6.5.** Unpacking Definition 6.1, clause 3 (the one for $X$) means that $\varsigma(X) = [a_i{:}A_i]r'$ and $\varnothing \vdash [a_i{:}A_i]r' : [A_i]A$. Following the typing rules of Figure 6, this is equivalent to $(a_i{:}A_i)_i \vdash r' : A$.

**Definition 6.6.** For each constant $C : A$ other than $\top$, $\bot$, and $\mathsf{isapp}$ fix some interpretation $C^{\mathcal{H}}$ which is an element $C^{\mathcal{H}} \in [\![A]\!]$. Suppose $\Gamma \vdash \varsigma$ and $\Gamma \vdash r : A$.

An **interpretation** of terms $[\![r]\!]_\varsigma$ is defined in Figure 9.

**Remark 6.7.** Definition 6.6 is in the same spirit as Definition 3.11, but now the modal types are contextual; the modal box contains a context $a_1{:}A_1, \ldots, a_n{:}A_n$. When we calculate $[\![X@(r_i)_1^n]\!]_\varsigma$ the denotation of $X@(r_i)_1^n$, the denotations of the terms $r_i$ provide denotations for the variables in that context.

**Lemma 6.8.** *If* $\Gamma \vdash \varsigma$ *then* $\Gamma \vdash \varsigma_{\mathbb{X}}$.

*Proof.* If $X \notin dom(\varsigma)$ then $X \notin dom(\varsigma_{\mathbb{X}})$.

Suppose $X \in dom(\varsigma)$. By Definition 6.3 $\varsigma_{\mathbb{X}}(X) = hd(\varsigma(X))$. By Definition 6.4 $\varsigma_{\mathbb{X}}(X) \in [\![[A_i]_1^n A]\!]$ for some $[A_i]_1^n A$. Unpacking Figure 8 this implies that $\varsigma_{\mathbb{X}}(X) = [a_i{:}A_i]_1^n r$ for some $\varnothing \vdash [a_i{:}A_i]_1^n r : [A_i]_1^n A$, and we are done. □

**Lemma 6.9.** *Suppose* $\Gamma \vdash r : A$ *and* $\Gamma \vdash \varsigma$. *Then* $\Gamma|_{\mathbb{A}} \vdash r\varsigma_{\mathbb{X}} : A$.

*Proof.* By Lemma 6.8 $\Gamma \vdash \varsigma_{\mathbb{X}}$. By Proposition 5.16 $\Gamma \vdash r\,\varsigma_{\mathbb{X}} : A$. By Lemma 5.14 $fa(r\varsigma_{\mathbb{X}}) = fa(r)$. It is a fact that $fa(r) \subseteq dom(\Gamma|_{\mathbb{A}})$, so by Proposition 2.13 $\Gamma|_{\mathbb{A}} \vdash r\,\varsigma_{\mathbb{X}} : A$ as required. □

**Theorem 6.10** (Soundness). *If* $\Gamma \vdash r : A$ *and* $\Gamma \vdash \varsigma$ *then* $[\![r]\!]_\varsigma \in [\![A]\!]$.

*Proof.* By induction on the the derivation of $\Gamma \vdash r : A$. Most of the rules follow by properties of sets and functions. We consider the interesting cases:

- *Rule* $([]\mathbf{I})$. Suppose $\Gamma, (a_i{:}A_i)_1^n \vdash r : A$ so that by $([]\mathbf{I})$ $\Gamma \vdash [a_i{:}A_i]r : [A_i]A$. Suppose $fa(r){\subseteq}\{a_1, \ldots, a_n\}$ and $\Gamma \vdash \varsigma$. Using Lemma 6.9 $\varnothing \vdash [a_i{:}A_i](r\varsigma_{\mathbb{X}}) : [A_i]A$.
  Suppose $x_i \in [\![A_i]\!]$ for $1{\leq}i{\leq}n$. By Definition 6.4
  $$\Gamma, (a_i{:}A_i)_1^n \vdash \varsigma[a_i{:=}x_i]_1^n$$
  so by inductive hypothesis for the derivation of $\Gamma, (a_i{:}A_i)_1^n \vdash r : A$ it follows that
  $$[\![r]\!]_{\varsigma[a_i:=x_i]_1^n} \in [\![A]\!].$$
  Now this was true for arbitrary $x_i$ and it follows from Definition 6.1 that $[\![[a_i{:}A_i]r]\!]_\varsigma \in [\![[A_i]A]\!]$ as required.
- *Rule* $([]\mathbf{E})$. Suppose $\Gamma, X{:}[A_i]A \vdash r : B$ and $\Gamma \vdash s : [A_i]A$ so that by $([]\mathbf{E})$ $\Gamma \vdash \mathit{let}\ X{=}s\ \mathit{in}\ r : B$.
  Suppose $\Gamma \vdash \varsigma$. By inductive hypothesis for $\Gamma \vdash s : [A_i]A$ we have $[\![s]\!]_\varsigma \in [\![[A_i]A]\!]$.
  It follows by Definition 6.4 that $\Gamma, X{:}[A_i]A \vdash \varsigma[X{:=}[\![s]\!]_\varsigma]$ so by inductive hypothesis for $\Gamma, X{:}[A_i]A \vdash r : B$ we have $[\![r]\!]_{\varsigma[X:=[\![s]\!]_\varsigma]} \in [\![B]\!]$. We now observe by Definition 6.6 that
  $$[\![\mathit{let}\ X{=}s\ \mathit{in}\ r]\!]_\varsigma = [\![r]\!]_{\varsigma[X:=[\![s]\!]_\varsigma]} \in [\![B]\!].$$
- *Rule* $(\mathbf{Ext})$. Suppose $\Gamma, X{:}[A_i]_1^n A \vdash r_i : A_i$ for $1{\leq}i{\leq}n$ so that by $(\mathbf{Ext})$ $\Gamma, X{:}[A_i]_1^n A \vdash X@(r_i)_1^n{:}A$.
  By inductive hypothesis for the typings $\Gamma, X{:}[A_i]_1^n A \vdash r_i : A_i$ we have $[\![r_i]\!]_\varsigma \in [\![A_i]\!]$ for $1{\leq}i{\leq}n$.
  Suppose $\Gamma, X{:}[A_i]A \vdash \varsigma$. By Definitions 6.4 and 6.6 this means that $\varsigma(X) = ([a_i{:}A_i]_1^n r') :: f$ for some $\varnothing \vdash [a_i{:}A_i]r' : [A_i]A$ and some $f \in (\Pi_{i=1}^n [\![A_i]\!]){\to}[\![A]\!]$. It follows that $f\,([\![r_i]\!]_\varsigma)_1^n \in [\![A]\!]$ as required.
- *Rule* $(\mathbf{Hyp})$. Suppose $\Gamma, a{:}A \vdash \varsigma$. By Definition 6.4 this means that $\varsigma(a) \in [\![A]\!]$. By Definition 6.6 $[\![a]\!]_\varsigma = \varsigma(a)$. The result follows.
- *Rule* $(\to\mathbf{I})$. Suppose $\Gamma, a{:}A \vdash r : B$ so that by $(\to\mathbf{I})$ $\Gamma \vdash \lambda a{:}A.r : A{\to}B$. Suppose $\Gamma \vdash \varsigma$ and choose any $x \in [\![A]\!]$. It follows that $\Gamma, a{:}A \vdash \varsigma[a{:=}x]$ and so by inductive hypothesis that $[\![r]\!]_{\varsigma[a:=x]} \in [\![B]\!]$.
  Since $x \in [\![A]\!]$ was arbitrary, by Definition 6.6 we have that
  $$[\![\lambda a{:}A.r]\!]_\varsigma = (x \in [\![A]\!] \mapsto [\![r]\!]_{\varsigma[a:=x]}) \in [\![A{\to}B]\!].$$

□

**Corollary 6.11.** 1. *There is no term $s$ such that $\varnothing \vdash s : (\mathbb{N}{\to}\mathbb{N}){\to}[\,](\mathbb{N}{\to}\mathbb{N})$ is typable and such that the map $\big(x{\in}\mathbb{N}^{\mathbb{N}} \mapsto hd([\![s]\!]_{\varnothing}\, x)\big) \in hd([\![[\,](\mathbb{N}{\to}\mathbb{N})]\!])^{[\![\mathbb{N}{\to}\mathbb{N}]\!]}$ is injective.*

2. *There is no term $s$ such that $\varnothing \vdash s : (\mathbb{N}{\to}\mathbb{N}){\to}[\mathbb{N}]\mathbb{N}$ is typable and such that the map $\big(x{\in}\mathbb{N}^{\mathbb{N}} \mapsto hd([\![s]\!]_{\varnothing}\, x)\big) \in hd([\![[\mathbb{N}]\mathbb{N}]\!])^{[\![\mathbb{N}{\to}\mathbb{N}]\!]}$ is injective.*

*Proof.* $hd[\![[\,](\mathbb{N}{\to}\mathbb{N})]\!]$ and $hd[\![[\mathbb{N}]\mathbb{N}]\!]$ are both countable sets whereas $[\![\mathbb{N}{\to}\mathbb{N}]\!] = \mathbb{N}^{\mathbb{N}}$ is uncountable. □

*6.2. Typings and denotations in the contextual system*

The examples from Subsection 2.3 transfer to the contextual system if we translate □- to $[\,]$- and -$_{@}$ to -@() (cf. Remark 5.5). So the reader can look to Subsection 2.3 for some simpler examples.

We now consider some slightly more advanced ideas.

*6.2.1. Moving between $[A]B$ and $[\,](A{\to}B)$*

We can move between the types $[A]B$ and $[\,](A{\to}B)$ using terms $f : [A]B{\to}[\,](A{\to}B)$ and $g : [\,](A{\to}B){\to}[A]B$ defined as follows:

$$\begin{array}{ll} \varnothing \vdash f = \lambda c{:}[A]B.\mathit{let}\ X{=}c\ \mathit{in}\ [\,]\lambda a{:}A.X@(a) & : [A]B{\to}[\,](A{\to}B) \\ \varnothing \vdash g = \lambda c{:}[\,](A{\to}B).\mathit{let}\ X{=}c\ \mathit{in}\ [a{:}A]((X@())a) & : [\,](A{\to}B){\to}[A]B \end{array}$$

It is routine to check that the typings above are derivable using the rules in Figure 6.

Intuitively, we can write the following:

- $f$ maps $[a{:}A]r$ to $[\,]\lambda a{:}A.r$.
- $g$ maps $[\,]\lambda a{:}A.r$ to $[a{:}A]((\lambda a{:}A.r)a)$ (so $g$ introduces a $\beta$-redex).

This can be made formal as follows:

$$\begin{array}{rl} hd[\![f\ ([a{:}A]r)]\!]_{\varsigma} = [\,]\lambda a{:}A.(r\varsigma_{\mathbb{X}}) & \text{and} \\ hd[\![g\ ([\,]\lambda a{:}A.r)]\!]_{\varsigma} = [a{:}A]((\lambda a{:}A.(r\varsigma_{\mathbb{X}}))a) & \end{array}$$

The fact that $g$ introduces a $\beta$-redex reflects the fact that we have given our language facilities to build up syntax—but not to destroy it. We can build a precise inverse to $f$ if we give ourselves an explicit destructor for $\lambda$-abstraction.

So for instance, we can give ourselves option types and then admit a constant symbol $\mathsf{match_lam} : [\,](A{\to}A){\to}\mathsf{option}([A]B)$ with intended behaviour as follows:

$$\mathsf{match_lam}\,(t) = \begin{cases} \mathsf{some}\,([a{:}A]\,r) & \text{if } [\![t]\!] = ([\,]\lambda a{:}A.r)::_ \\ \mathsf{none} & \text{otherwise} \end{cases}$$

Using $\mathsf{match_lam}$ we could map from $[\,](A{\to}B)$ to $[A]B$ in a manner that is inverse to $f$.[16]

[16] We do not promote this language directly as a practical programming language, any more than one would promote the pure $\lambda$-calculus. We should add constants for the operations we care about.

The point is that in this language, there are things we can do using the modal types that cannot be expressed directly in the pure $\lambda$-calculus, no matter how many constants we might add.

*6.2.2. The example of exponentiation, revisited*

Recall from Subsection 2.3.4 the discussion of exponentiation and how in the modal system the natural term to meta-program exponentiation introduced $\beta$-reducts.

The following term implements exponentiation:

$$\begin{array}{rl} exp\ 0 & \Rightarrow [b{:}\mathbb{N}]1 \\ exp\ (\mathsf{succ}\ n) & \Rightarrow let\ X{=}[b{:}\mathbb{N}]exp\ n\ in\ [b{:}\mathbb{N}](b * (X@(b))) \end{array}$$

This term does not generate $\beta$-reducts in the way we noted of the corresponding term from Subsection 2.3.4. For instance,

$$hd[\![exp\ 2]\!]_{\varnothing} = [b{:}\mathbb{N}](b * b * 1).$$

Compare this with Subsection 3.3.4.

Think of the $[b{:}\mathbb{N}]$ in $[b{:}\mathbb{N}]r$ as a 'translucent lambda', and think of $X@(r_i)$ as a corresponding application. We can use these to carry out computation—a rather weak computation; just a few substitutions as formalised in the clause for $X@(r_i)_i$ in Figure 7—but this computation occurs *inside* a modality, which we could not do with an ordinary $\lambda$-abstraction.

Now might be a good moment to return to the clause for $[a_i{:}A_i]r$ in Figure 9:

$$[\![[a_i{:}A_i]_1^n r]\!]_{\varsigma} = [a_i{:}A_i]_1^n(r\ \varsigma_{\mathbb{X}}) :: \big((x_i{\in}[\![A_i]\!])_1^n \mapsto [\![r]\!]_{\varsigma[a_i:=x_i]_1^n}\big)$$

We see the $\lambda$-abstraction in the semantics, and we also see its 'translucency': the $\lambda$-abstraction appears in the extension, but is also associated with a non-functional intension.

*6.2.3. Syntax to denotation*

There is a schema of *unpack* programs, parameterised over $(a_i{:}A_i)_1^n$ which evaluates syntax with $n$ free atoms:

$$unpack = \lambda b{:}[A_i]_1^n B.let\ X{=}b\ in\ \lambda(a_i{:}A_i)_1^n.X@(a_i)_1^n \qquad : [A_i]_1^n B{\to}((A_i)_1^n{\to}B)$$

We can express the following connection between $unpack$ (which is a term) and $tl$ (which is a function on denotations):

**Lemma 6.12.** *Suppose* $\Gamma \vdash [a_i{:}A_i]s : [A_i]A$ *and* $\Gamma \vdash \varsigma$. *Then*

$$[\![unpack\ [a_i{:}A_i]s]\!]_{\varsigma} = tl[\![[a_i{:}A_i]s]\!]_{\varsigma}.$$

*Proof.* By long but routine calculations unpacking Figure 9. □

As an aside, note that if we have diverging terms $\omega_i : A_i$ then we can combine this with $unpack$ to obtain a term $\varnothing \vdash \lambda a{:}[A_i]A.unpack\ a\ (\omega_i) : [A_i]A{\to}A$. In a call-by-name evaluation strategy, this loops forever if evaluation tries to refer to one of the (diverging) arguments.

*6.2.4. Modal-style axioms*

As in Subsection 2.3.1 we can write functions corresponding to axioms from the necessity fragment of S4:

$$\begin{array}{rll} T = \lambda a{:}[\,]A.let\ X{=}a\ in\ X@() & & : [\,]A{\to}A \\ 4 = \lambda x.let\ X{=}x\ in\ [\,][\,]X@() & & : [\,]A{\to}[\,][\,]A \\ K = \lambda f.\lambda x.let\ F{=}f\ in\ let\ X{=}x\ in\ F@()\ X@() & & : [\,](A{\to}B){\to}[\,]A{\to}[\,]B \end{array}$$

(Of course, $T$ is just a special case of $unpack$ above.)

*6.2.5. More general contexts*

Versions of the terms $4$ and $K$ exist for non-empty contexts. For example, we can have a schema of $4_\Gamma$ axioms, for any context $\Gamma$:

$$4_\Gamma = \lambda x{:}[\Gamma]A.let\ X{=}x\ in\ [\,][\Gamma]X@(id_\Gamma) \qquad : [\Gamma]A{\to}[\,][\Gamma]A$$

Here and below we abuse notation by putting $[\Gamma]$ in the type; we intend the *types* in $\Gamma$, with the variables removed.

Above, $id_\Gamma$ is the identity substitution defined inductively on $\Gamma$ by

$$id_\cdot = \cdot \qquad \text{and} \qquad id_{\Gamma,x:A} = id_\Gamma, x.$$

Note that the terms realising $4_\Gamma$ are not uniform, because the substitution $id_\Gamma$ is not a term in the language; it is a meta-level concept, producing different syntax depending on $\Gamma$.

Similarly, we have a schema of $K_\Gamma$ terms:

$$K_\Gamma = \lambda f.\lambda x.let\ F{=}f\ in\ let\ X{=}x\ in\ [\Gamma]F@id_\Gamma\ X@id_\Gamma{:}\ [\Gamma](A{\to}B){\to}[\Gamma]A{\to}[\Gamma]B$$

...and terms exposing the structural rules of contexts:

$$\begin{aligned} weaken_{\Gamma_1,\Gamma_2} &= \lambda z.let\ Z{=}z\ in\ [\Gamma_1,\Gamma_2](Z@(id_{\Gamma_1})) && : [\Gamma_1]A{\to}[\Gamma_1,\Gamma_2]A \\ contract_B &= \lambda z.let\ Z{=}z\ in\ [x{:}B](Z@(x,x)) && : [B,B]A{\to}[B]A \\ exchange_{B,C} &= \lambda z.let\ Z{=}z\ in\ [y{:}C,x{:}B](Z@(x,y)) && : [B,C]A{\to}[C,B]A \end{aligned}$$

We give $weaken$ in full generality and then for brevity $contract$ and $exchange$ only for two-element contexts. If we think in terms of multimodal logic [GKWZ03] these terms 'factor', 'fuse', and 'rearrange' contexts/modalities.

## 7. Shapeliness

We have seen semantics to both the modal and contextual type systems. We have also noted that, like function-spaces, our semantics *inflates*. We discussed why in Remark 3.5 and Subsection 3.3.3.

In this section we delve deeper into the fine structure of the denotation to isolate a property of those parts of the denotation that can be described by syntax (Definition 7.1). This is an attractive well-formedness/well-behavedness property in its own right, and furthermore, we can exploit it to strengthen Corollaries 3.15 and 6.11 (see Corollary 7.8).

**Definition 7.1.** Define the **shapely** $x \in [\![A]\!]$ inductively by the rules in Figure 10.

Call $\varsigma$ **shapely** when:

- $\varsigma(X)$ is shapely for every $X \in dom(\varsigma)$.
- $\varsigma(a)$ is shapely for every $a \in dom(\varsigma)$.

**Remark 7.2.** In words, the condition $x = [\![hd(x)]\!]_\varnothing$ in (**Shape**$[\,]$) reads: "$x$ is equal to the extension/denotation of the intension/syntax of $x$". So, if $x$ is in $[\![[A_i]A]\!]$ and we extract the syntax from this, which is $hd(x)$, then if we calculate the denotation of that syntax we get $x$ back again.

Elements in $[\![\mathbb{B}]\!]$, $[\![\mathbb{N}]\!]$, or $[\![\mathbb{N}{\to}\mathbb{N}]\!]$ are automatically shapely, because they carry no intension/syntax.

Conversely, $x$ is not shapely when it has an intension and an extension—and they do not match up. The paradigmatic non-shapely element is $[\,]0 :: 1$, since the intension 'the syntax $0$' does not match the extension 'the number $1$'.

$$\frac{tl(x) \in [\![(A_i)_1^n \to A]\!] \text{ is shapely} \quad x = [\![hd(x)]\!]_\varnothing}{x \in [\![[A_i]_1^n A]\!] \text{ is shapely}} \ (\mathbf{Shape}[\,])$$

$$\frac{\forall y \in [\![B]\!].\, y \text{ is shapely} \Rightarrow xy \in [\![A]\!] \text{ is shapely}}{x \in [\![B \to A]\!] \text{ is shapely}} \ (\mathbf{ShapeFun})$$

$$\frac{(x \in [\![\mathbb{B}]\!])}{x \in [\![\mathbb{B}]\!] \text{ is shapely}} \ (\mathbf{Shape}\mathbb{B}) \qquad \frac{(x \in [\![\mathbb{N}]\!])}{x \in [\![\mathbb{N}]\!] \text{ is shapely}} \ (\mathbf{Shape}\mathbb{N})$$

**Figure 10:** Shapeliness

**Lemma 7.3.** 1. *If $x \in [\![B \to A]\!]$ is shapely and $y \in [\![B]\!]$ is shapely, then so is $xy \in [\![A]\!]$.*
2. *If $x \in [\![[A_i]A]\!]$ is shapely then $x = [\![hd(x)]\!]_\varnothing$.*
3. *Every $f \in \mathbb{N}^\mathbb{N}$ is shapely.*

*Proof.* The first two parts follow from the form of the inductive definition in Figure 10. The third part is a simple application of (**ShapeFun**), noting that by (**Shape**$\mathbb{N}$) every $n \in \mathbb{N}$ is shapely. □

We can combine Lemmas 7.3 and 6.12 to get a nice corollary of shapeliness (*unpack* is from Subsection 6.2.3):

**Corollary 7.4.** *If $x \in [\![[A_i]A]\!]$ is shapely then $tl(x) = [\![unpack\ hd(x)]\!]_\varnothing$.*

*Proof.* Suppose $x \in [\![[A_i]A]\!]$ is shapely, so that by part 2 of Lemma 7.3 $x = [\![hd(x)]\!]_\varnothing$. We apply $tl$ to both sides and use Lemma 6.12. □

**Lemma 7.5.** *Suppose $\Gamma, X : [B_i]B \vdash r : A, \Gamma \vdash [a_i{:}B_i]s : [B_i]B$, and $\Gamma \vdash \varsigma$. Then $[\![r[X{:=}[a_i{:}B_i]s]]\!]_\varsigma = [\![r]\!]_{\varsigma[X:=[\![[B_i]s]\!]_\varsigma]}$.*

*Proof.* By a routine induction on the derivation of $\Gamma \vdash r : A$, similar to the proof of Lemma 4.7. □

**Corollary 7.6.** *Suppose $\Gamma \vdash r : A$, $\Gamma \vdash \varsigma$, and $\varsigma$ is shapely. Then $[\![r]\!]_\varsigma = [\![r\, \varsigma_\mathbb{X}]\!]_{\varsigma|_\mathbb{A}}$.*

*Proof.* First, we note that the effect of $\varsigma_\mathbb{X}$ can be obtained by concatenating $[X{:=}hd(\varsigma(X))]$ for every $X \in fu(r)$. The order does not matter because by construction $hd(\varsigma(X))$ is closed syntax (no free variables). Furthermore since $\varsigma$ is shapely, $\varsigma(X) = [\![hd(\varsigma(X))]\!]_\varnothing$ so we can write $\varsigma$ as

$$\varsigma|_\mathbb{A} \cup [X{:=}[\![hd(\varsigma(X))]\!]_\varsigma \mid X \in dom(\varsigma)],$$

where here $[X{:=}x_X \mid X \in \mathcal{X}]$ is the map taking $X$ to $x_X$ for every $X \in \mathcal{X}$.[17] We now use Lemma 7.5 for $[X{:=}\varsigma(X)]$ for each $X \in fu(r)$, and Proposition 5.9. □

**Proposition 7.7.** *Suppose $\Gamma \vdash r : A$ and suppose $\Gamma \vdash \varsigma$. Then if $\varsigma$ is shapely then so is $[\![r]\!]_\varsigma$.*

*Proof.* By induction on the typing $\Gamma \vdash r : A$ (Figure 6).

[17] Strictly speaking we also need a version of Proposition 4.3 for the contextual system; this is not hard.

- *The case of* (**Hyp**) is immediate because by assumption $\varsigma(a)$ is shapely.
- *The case of* (**Const**) is also immediate (provided that all semantics for constants are shapely).
- *The case of* ($\to$**I**). Suppose $\Gamma, a{:}A \vdash r : B$ so that by ($\to$**I**) $\Gamma \vdash \lambda a{:}A.r : A{\to}B$. Suppose $x \in [\![A]\!]$ is shapely. Then so is $\varsigma[a{:=}x]$ and by inductive hypothesis so is $[\![r]\!]_{\varsigma[a:=x]}$. It follows by (**ShapeFun**) that
$$[\![\lambda a{:}x.r]\!]_\varsigma = \big(x \in [\![A]\!] \mapsto [\![r]\!]_{\varsigma[a:=x]}\big)$$
is shapely.
- *The case of* ($\to$**E**). Suppose $\Gamma \vdash r' : A{\to}B$ and $\Gamma \vdash r : A$ so that by ($\to$**E**) $\Gamma \vdash r'r : B$. By inductive hypothesis $[\![r']\!]_\varsigma$ and $[\![r]\!]_\varsigma$ are both shapely. By part 1 of Lemma 7.3 so is $[\![r'r]\!]_\varsigma = [\![r']\!]_\varsigma[\![r]\!]_\varsigma$.
- *The case of* ($[\,]$**I**). Suppose $\Gamma, (a_i{:}A_i) \vdash r : A$ and $fa(r) \subseteq \{a_i\}$ so that by ($[\,]$**I**) $\Gamma \vdash [a_i{:}A_i]r : [A_i]A$.
By inductive hypothesis $[\![r]\!]_{\varsigma'}$ is shapely for every shapely $\varsigma'$ such that $\Gamma, (a_i{:}A_i) \vdash \varsigma'$ and it follows that $tl[\![[a_i{:}A_i]r]\!]_\varsigma = [\![\lambda(a_i{:}A_i).r]\!]_\varsigma$ is shapely.
Also unpacking definitions
$$hd[\![[a_i{:}A_i]r]\!]_\varsigma = [a_i{:}A_i](r\varsigma_{\mathbb{X}}).$$
So it suffices to verify that $[\![[a_i{:}A_i]r]\!]_\varsigma = [\![[a_i{:}A_i](r\varsigma_{\mathbb{X}})]\!]_\varnothing$. This follows from Corollary 7.6.

□

Corollary 6.11 proved that denotations cannot be reified to syntax in general, by general arguments on cardinality. But our denotational semantics is inflated; $[\![[\,](A{\to}B)]\!]$ and $[\![A{\to}B]\!]$ have the same cardinality even if $hd([\![[\,](A{\to}B)]\!])$ and $[\![A{\to}B]\!]$ do not. Corollary 7.8 tells us that we cannot in general even reify denotation to the 'inflated' denotations, even if they are large enough. In this sense, inflation is 'not internally detectable':

**Corollary 7.8.** 1. *There is no term $s$ such that $\varnothing \vdash s : (\mathbb{N}{\to}\mathbb{N}){\to}[\,](\mathbb{N}{\to}\mathbb{N})$ is typable and such that $[\![s]\!]_\varnothing \in [\![[\,](\mathbb{N}{\to}\mathbb{N})]\!]^{[\![\mathbb{N}\to\mathbb{N}]\!]}$ is injective.*
2. *There is no term $s$ such that $\varnothing \vdash s : (\mathbb{N}{\to}\mathbb{N}){\to}[\mathbb{N}]\mathbb{N}$ is typable and such that $[\![s]\!]_\varnothing \in [\![[\mathbb{N}]\mathbb{N}]\!]^{[\![\mathbb{N}\to\mathbb{N}]\!]}$ is injective.*

*Proof.* By Proposition 7.7 $s$ is shapely, so by part 1 of Lemma 7.3 it maps shapely elements of $\mathbb{N}^{\mathbb{N}} = [\![\mathbb{N}{\to}\mathbb{N}]\!]$ to shapely elements of $[\![[\,](\mathbb{N}{\to}\mathbb{N})]\!]/[\![[\mathbb{N}]\mathbb{N}]\!]$. By part 3 of Lemma 7.3 and the fact that $\mathbb{N}^{\mathbb{N}}$ is uncountable, the number of shapely elements of $\mathbb{N}^{\mathbb{N}}$ is uncountable. By part 2 of Lemma 7.3 and the fact that syntax is countable, the number of shapely elements of $[\![[\,](\mathbb{N}{\to}\mathbb{N})]\!]$ and $[\![[\mathbb{N}]\mathbb{N}]\!]$ is countable. The result follows. □

It is clear that part 1 of Corollary 7.8 can be directly adapted to the modal system from Section 2.

## 8. □ as a (relative) comonad

We noted as early as Remark 2.14 that □ looks like a comonad. In this section, we show that this is indeed the case.

Before doing this, we would like to convince the reader that this is obviously impossible.

True, we have natural maps $\Box A \to A$ (evaluation) and $\Box A \to \Box\Box A$ (quotation). However, if $\Box$ is a comonad then it has to be a functor on some suitable category, so we would expect some natural map in $(A{\to}B) \to (\Box A{\to}\Box B)$. This seems unlikely because if we had this, then we could take $A$ to be a unit type (populated by one element) and $B = (\mathbb{N}{\to}\mathbb{N})$ and thus generate a natural map from $\mathbb{N}{\to}\mathbb{N}$ to $\Box(\mathbb{N}{\to}\mathbb{N})$. But how would we do this in the light of Corollaries 6.11 and Corollary 7.8? Even where closed syntax exists for a denotation, there may be many different choices of closed syntax to represent the same denotation, further undermining our chances of finding *natural* assignments. '$\Box$ as a comonad' seems doomed.

This problem is circumvented by the 'trick' of considering a category in which each denotation must be associated with syntax; we do not insist that the syntax and denotation match. This is essentially the same idea as *inflation* in Remark 3.2 (but applied in the other direction; in Remark 3.2 we inflated by adding a purported denotation to every syntax; here we are inflating by adding a purported syntax to every denotation). In the terminology of Definition 7.1 we can say that we do not insist on *shapeliness*. We simply insist that some syntax be provided.

Modulo this 'trick', $\Box$ becomes a well-behaved comonad after all.

### *8.1. $\Box$ as a comonad*

**Notation 8.1.** Write $\pi_1$ for *first projection* and $\pi_2$ for *second projection*.

That is, $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$.

**Definition 8.2.** Suppose $f \in [\![\Box B]\!]^{[\![\Box A]\!]}$. Define a function $\boxdot f \in [\![\Box\Box B]\!]^{[\![\Box\Box A]\!]}$ by sending

$$\Box\Box s :: x \quad \text{to} \quad \Box\pi_1(f(\Box s :: [\![s]\!]_\varnothing)) :: f(x)$$

where $x \in [\![\Box A]\!]$ and $s : A$.

**Remark 8.3.** It may be useful to unpack what $\boxdot f$ does. Suppose

$$f(\Box r :: x) = \Box r' :: x' \quad \text{and} \quad f(\Box s :: y) = \Box s' :: y'$$

where $x \in [\![A]\!]$ and $y = [\![s]\!]_\varnothing$. Then $\Box f$ sends $\Box\Box s :: \Box r :: x$ to $\Box\Box s' :: \Box r' :: x'$.

**Definition 8.4.** Define a category $\mathcal{J}$ by:

- Objects are types $A$.[18]
- Arrows from $A$ to $B$ are functions from $[\![\Box A]\!]$ to $[\![\Box B]\!]$ (not from $[\![A]\!]$ to $[\![B]\!]$; as promised above, some syntax must be provided).

Composition of arrows is given by composition of functions.

**Definition 8.5.** Define an endofunctor $\boxdot$ on $\mathcal{J}$ mapping

- an object $A$ to $\boxdot A = \Box A$ and
- an arrow $f : A \to B$ to $\boxdot f : \boxdot A \to \boxdot B$ from Definition 8.2.

[18] The reader might prefer to take objects to be $[\![A]\!]$. This is fine; the assignment $A \longmapsto [\![A]\!]$ is injective, so it makes no difference whether we take objects to be $A$ or $[\![A]\!]$.

So $\Box$ is a type-former acting on types and $\boxdot$ is a functor acting on objects and arrows. Objects happen to be types, and $\boxdot$ acts on objects just by prepending a $\Box$. Arrows are functions on sets, and the action on $\boxdot$ on these functions is more complex as defined above.

**Definition 8.6.**
- Write $id_A$ for the identity on $[\![\Box A]\!]$ for each $A$.
- Write $\delta_A$ for the arrow from $\boxdot A$ to $A$ given by the function mapping $[\![\Box\Box A]\!]$ to $[\![\Box A]\!]$ taking $\Box\Box r :: x$ to $x$ (where $x \in [\![\Box A]\!]$). This will be the **counit** of our comonad.
- Write $\epsilon_A$ for the arrow from $\boxdot A$ to $\boxdot\boxdot A$ given by the function mapping $[\![\Box\Box A]\!]$ to $[\![\Box\Box\Box A]\!]$ taking $\Box\Box r :: x$ to $\Box\Box\Box r :: \Box\Box r :: x$ (where $x \in [\![\Box A]\!]$). This will be the **comultiplication** of our comonad.

**Lemma 8.7.** *$\boxdot$ from Definition 8.5 is a functor.*

*Proof.* It is routine to verify that $\boxdot id_A = id_{\boxdot A}$ and if $f : A \to B$ and $g : B \to C$ then $\boxdot g \circ \boxdot f = \boxdot(g \circ f)$. □

**Lemma 8.8.**
- *$\delta_A$ is a natural transformation from $\boxdot$ to $id_{\mathcal{J}}$ (the identity functor on $\mathcal{J}$).*
- *$\epsilon_A$ is a natural transformation from $\boxdot$ to $\boxdot\boxdot$.*

*Proof.* Suppose $f : A \to B$. For the first part, we need to check that $f \circ \delta_A = \delta_B \circ \boxdot f$. This is routine:

$$(f \circ \delta_A)(\Box\Box r :: x) = f(x) \quad \text{and}$$
$$\delta_B \circ \boxdot f = \pi_2\big(\Box\pi_1(f(\Box r :: [\![r]\!]_\varnothing)) :: f(x)\big) = f(x)$$

The second part is similar and no harder. □

Note that $\boxdot\delta_A$ is an arrow from $\boxdot\boxdot A$ to $\boxdot A$.

**Lemma 8.9.** *$\boxdot\delta_A$ maps $\Box\Box\Box s :: \Box\Box r :: x \in [\![\Box\Box\Box A]\!]$ to $\Box\Box s :: x \in [\![\Box\Box A]\!]$.*

*Proof.* By a routine calculation on the definitions:

$$\begin{aligned}
\Box\delta_A(\Box\Box\Box s :: \Box\Box r :: x) &= \Box\pi_1(\delta_A(\Box\Box\Box s :: [\![\Box\Box s]\!]_\varnothing)) :: \delta_A(\Box\Box r :: x) && \text{Definition 8.2}\\
&= \Box\pi_1(\delta_A(\Box\Box\Box s :: [\![\Box\Box s]\!]_\varnothing)) :: x && \text{Definition 8.6}\\
&= \Box\pi_1([\![\Box\Box s]\!]_\varnothing) :: x && \text{Definition 8.6}\\
&= \Box\Box s :: x && \text{Figure 4}
\end{aligned}$$

□

**Proposition 8.10.** *$\boxdot$ is a comonad.*

*Proof.* We need to check that

- $\boxdot\epsilon_A \circ \epsilon_A = \epsilon_{\boxdot A} \circ \epsilon_A$ and
- $\delta_{\boxdot A} \circ \epsilon_A = id_A = \boxdot\delta_A \circ \epsilon_A$.

Both calculations are routine. We consider just the second one. Consider $\Box\Box s :: \Box r :: x \in [\![\Box\Box A]\!]$. Then

$$\begin{aligned}
(\delta_{\boxdot A} \circ \epsilon_A)(\Box\Box s :: \Box r :: x) &= \delta_{\boxdot A}(\boxed{\Box\Box\Box s ::}\ \Box\Box s :: \Box r :: x)\\
&= \Box\Box s :: \Box r :: x
\end{aligned}$$

$$\begin{aligned}
(\boxdot\delta_A \circ \epsilon_A)(\Box\Box s :: \Box r :: x) &= \boxdot\delta_A(\Box\Box\Box s :: \boxed{\Box\Box s ::}\ \Box r :: x)\\
&= \Box\Box s :: \Box r :: x
\end{aligned}$$

The shaded part is the part that gets 'deleted'. In the second case we use Lemma 8.9. □

*8.2. $\Box$ as a relative comonad*

Recall that in the previous subsection we represented $\Box$ as a comonad on a category with the 'trick' of associating syntax to every denotation.

It is possible to put this in a broader context using the notion of *relative comonad*.

**Definition 8.11.** Following [ACU10, Definition 1], a **relative comonad** consists of the following information:

- Two categories $\mathcal{J}$ and $\mathcal{C}$ and a functor $J : \mathcal{J} \to \mathcal{C}$.[19]
- A functor $T : \mathcal{J} \to \mathcal{C}$.
- For every $X \in \mathcal{J}$ an arrow $\delta_X : TX \to JX \in \mathcal{C}$ (the **unit**).
- For every $X, Y \in \mathcal{J}$ and arrow $k : TX \to JY \in \mathcal{C}$, an arrow $k^* : TX \to TY$ (the **Kleisli extension**).

Furthermore, we insist on the following equalities:

- If $X, Y \in \mathcal{J}$ and $k : X \to Y \in \mathcal{J}$ then $k = k^* \circ \delta$.
- If $X \in \mathcal{J}$ then $\delta^*_X = id_{TX}$.
- If $X, Y, Z \in \mathcal{J}$ and $k : TX \to JY$ and $l : TY \to JZ$ then $l^* \circ k^* = (l^* \circ k)^*$.

**Definition 8.12.** Take $\mathcal{C}$ to have objects types $A$ and arrows elements of $[\![B]\!]^{[\![A]\!]} = [\![A \to B]\!]$— this is simply the natural category arising from the denotational semantics of Figure 3.

Take $\mathcal{J}$ to be the category of Definition 8.4.

Take $J$ to map $A \in \mathcal{J}$ to $\Box A \in \mathcal{C}$ and to map $f \in [\![\Box B]\!]^{[\![\Box A]\!]}$ to itself.

Take $T$ to map $A \in \mathcal{J}$ to $\Box\Box A \in \mathcal{C}$ and to map $f \in [\![\Box B]\!]^{[\![\Box A]\!]}$ to $\boxdot f$ from Definition 8.2.

**Proposition 8.13.** *Definition 8.12 determines a relative comonad on $\mathcal{C}$.*

It is slightly simplified, but accurate, to describe relative (co)monads as being for the case where we have an operator that is nearly (co)monadic but the category in question has 'too many objects'. By that view, $\Box$ is a comonad on the full subcategory of $\mathcal{C}$ over modal types.

Now the intuition of modal types $\Box A$ is 'closed syntax', so it may be worth explicitly noting here that this full subcategory is *not* just a category of syntax. Each $[\![\Box A]\!]$ contains for each term $\varnothing \vdash r : A$ also a copy of $[\![A]\!]$, because we inflate.

## 9. Conclusions

The intuition realised by the denotation of $\Box A$ in this paper is 'typable closed syntax of the same language, of type $A$'. This is difficult to get right because it is self-referential; the denotation of a term of modal type can be the syntax of another term, which might have larger size than the term that denoted it. If we are careless, then this fact might make it impossible to define denotation by induction on syntax. We noted this in Subsection 3.3.3.

For that reason we realised this intuition by an 'inflated' reading of $\Box A$ as 'closed syntax, and purported denotation of that syntax'. If the system constructs a term inhabiting the denotation of a modal type, then it is required to provide at the same time some denotation

[19] The clash with the $\mathcal{J}$ from Definition 8.4 is deliberate: this is the only $\mathcal{J}$ we will care about in this paper. The definition of relative comonad from [ACU10] is general in the source category.

for that term—even if that denotation is 'silly'. As noted in Remark 3.2, there is no restriction that $\Box r :: x \in [\![\Box A]\!]$ needs to match up, in that $r$ must have denotation $x$.

When $r$ and $x$ *do* match up we say that $\Box r :: x$ is *shapely*. This is Definition 7.1, and we use this notion for our culminating result in Corollary 7.8, which entails that there is no uniform family of terms of type $A \to \Box A$.

The proof of this involves a beautiful interplay between syntax and denotation, which also illustrates the usefulness of denotational techniques; we can use a sound model to show that certain things *cannot* happen in the syntax, because if they did, they would have to happen in the model.

*Future work.* One avenue for future work is to note that our denotation is *sets based*, and so this invites generalisation to *nominal* sets semantics [GP01].

Perhaps we could leverage this to design a language which combines the simplicity of the purely modal system with the expressivity of contextual terms. Specifically, nominal sets are useful for giving semantics to open terms [GM11, Gab11] and we hope to develop a language in which we can retain the modal type system but relax the condition that $fa(r) = \varnothing$ in $(\Box\mathbf{I})$ in Figure 1 (much as the contextual system does, but in the 'nominal' approach we would not add types to the modality).

The underlying motivation here is that the contextual system is 'eager' in accounting for free variables—we need to express all the variables we intend to use in the contextual modal type, by putting their types in the modality. We might prefer to program on open syntax in a 'lazy' fashion, by stating that the syntax may be open, but not specifying its free variables explicitly in the type.

Note that this is not the same thing as programming freely on open syntax. Free variables would still be accounted for in the typing context (leading to some form of *dynamic linking* as and when open syntax is unboxed and evaluated; for an example of a $\lambda$-calculus view of dynamic linking, though not meta-programming, see [AFZ03]). So all variables would be eventually accounted for in the typing context, but they would not need to be listed in the type.

This is another reason for the specific design of our denotational semantics and taking the denotation of $\Box A$ to be specifically closed syntax; we hope to directly generalise this using nominal techniques so that $\Box A$ can also denote (atoms-)open syntax. This is future work.

*On the precise meaning of Corollary 7.8.* Corollary 7.8 depends on the fact that we admitted no constants of type $(\mathbb{N}\to\mathbb{N})\to\Box(\mathbb{N}\to\mathbb{N})$. We may be able to admit such a constant, representing a function that takes denotation and associates to it some 'dummy syntax' chosen in some fixed but arbitrary manner.

So Corollary 7.8 does not (and should not) prove that terms of type $(\mathbb{N}\to\mathbb{N})\to\Box(\mathbb{N}\to\mathbb{N})$ are completely impossible—only that they do not arise from the base system and cannot exist unless we explicitly choose to put them in there.

*Technical notes on the jump in complexity from modal to contextual system.* We noted in the introduction that Sections 2 and 5, and Sections 3 and 6 are parallel developments of the syntax and examples of the modal and contextual systems.

We briefly survey technical details of how these differences manifest themselves.

- The contextual system enriches the modal system with types in the modality. The increase in expressivity is exemplified in Subsection 6.2.2.

- In the contextual system and not in the modal system, instantiation of unknowns can trigger an atoms-substitution (see Definition 5.11) leading to a kind of 'cascade effect'. This turns out to be terminating, well-behaved, and basically harmless—but this has to be verified, and that brings some specific technical material forward in the proofs for the contextual case that is not so prominent in the purely modal case (notably, Lemma 5.15).
- A clear view of exactly where the extra complexity of the contextual system 'lives' in the denotation can be obtained by comparing the denotational semantics of $\Box A$ and $[A_i]A$ in Figures 3 and 8.

*Related work*

$\Box$ *and monads.* Famously, Moggi proposed to model computation using a monad [Mog91]. Let us write it as $\bigcirc A$. This type is intuitively populated by 'computations of type $A$'. The unit arrow $A \to \bigcirc A$ takes a value of type $A$ and returns the trivial computation that just returns $A$.

Pfenning and Davies [PD01, Section 7, page 21] have shown that the type $\bigcirc A$ can be encoded in the modal system as $\Diamond\Box A$, while the function-space $A \Rightarrow B$ of a monadic system is encoded as $\Box A \to B$ of the modal system.

The difference from the *co*monad of this paper in that our $\Box A$ is populated by *closed syntax*, and not by *computation*.

If we have an element of $\mathbb{N}^{\mathbb{N}}$ then it is easy to build a computation that just returns that value; it is however not easy—and may be impossible—to exhibit closed syntax to represent this computation.

We could add a constant to our syntax for each of the uncountably many functions from natural numbers to natural numbers. This would be mathematically fine—but not particularly implementable. We do not assume this.

Closed syntax is of course related to computation, and we can make this formal: Given an element in $\Box(\mathbb{N}\to\mathbb{N})$ we can map it to a computation, just by executing it. So intuitively there is an arrow $\Box A \to \bigcirc A$.

See also [Kob97, BdP00, AMdPR01], where the $\Box$ operator of several constructive variants of S4 (not equivalent to the version we presented here) is modeled as comonads.

*Brief survey of applications of* $\Box$ *calculi.* Logic and denotation, not implementation, are the focus of this paper, but the '$\Box$-calculi' considered in this paper have their motivation in implementation and indeed they were specifically designed to address implementational concerns. We therefore give a brief survey of how (contextual) modal types have been useful in the more applied end of computer science.

The connection of the modal $\Box$ calculus with partial evaluation and staged computation was noticed by Davies and Pfenning [DP01, PD01], and subsequently used as a language for run-time code generation by Wickline et al. [WLP98]. The contextual variant of $\Box$ as a basis for meta-programming and modeling of higher-order abstract syntax was proposed by Nanevski and Pfenning [NP05], and subsequently used to reason about optimised implementation of higher-order unification in Twelf [PP03], which could even be scaled to dependent types [NPP08].

Recently, the contextual flavor of the system has been used in meta-programming applications for reasoning and programming with higher-order abstract syntax by Pientka and collaborators [Pie08, PD08, FP10, CP12].

*Relationship between the formulation with meta-variables and labeled natural deductions.* The syntax of terms from Definition 2.6 does not follow instantly from the syntax of types from Definition 2.2; in particular, the use of a two-level syntax (also reminiscent of the two levels of nominal terms [UPG04]) is a design choice, not an inevitability.

The usual way to present inference systems based on modal logic is to have a propositional (or variable) context where each proposition is labeled by the 'world' at which it is true [Sim94].

When S4 is considered, we take advantage of reflexivity and transitivity of the Kripke frame to simplify the required information to two kinds of facts:

1. What holds at the current world, but not necessarily in all future worlds.
2. What holds in the current world and also in all future worlds.

By this view, the first kind of fact corresponds to atoms $a$, and the second kind of fact corresponds to unknowns $X$. So this can be seen as the origin of the two-level structure of our syntax in this paper.

The interested reader can find the modal (non-contextual) version of our type system presented using the labeled approach in a paper by Davies and Pfenning [DP01], and each stage of computation is indeed viewed as world in a Kripke frame.

*CMTT and nominal terms.* Nominal terms were developed in [UPG03, UPG04] and feature a two-level syntax, just like CMTT. That is made very clear in this paper, where the first author imported the nominal terms terminology of *atoms* and *unknowns*.

The syntax of this paper is not fully nominal—the $[a_i]r$ of the contextual system may look like a nominal abstraction, but there are no suspended permutation $\pi \cdot X$ (instead, we have types in the modality). One contribution of this paper is to make formal, by a denotation, the precise status of the two levels of variable in CMTT.

So we can note that the abstraction for atoms is functional abstraction in CMTT whereas the abstraction for atoms in nominal terms is nominal atoms-abstraction;[20] unknowns of nominal terms range over elements of nominal sets, whereas unknowns of CMTT range over ordinary sets functionally abstracted over finitely many arguments; the notion of *equivariance* (symmetry up to permuting atoms) characteristic of all nominal techniques is absent in CMTT (the closest we get is a term like $exchange_{B,C}$ in Subsection 6.2.4); and in contrast the self-reflective character of CMTT is absent from nominal terms and the logics built out of it [Gab12]. So in spite of some structural parallels between CMTT and nominal terms in that both are two-level, there are also significant differences.

As noted above, there is a parallel between CMTT and Kripke structures, that is made more explicit in [DP01]. A direct connection between nominal terms and Kripke semantics has never been made, but the first author at least has been aware of it as a possibility, where 'future worlds' corresponds to 'more substitutions arriving'. Also as discussed above, an obvious next step is to develop a modified modal syntax which takes on board more 'nominal' ideas, applied to the modal intuitions which motivate the $\lambda$-calculus of this paper. This is future work.

[20] In [GM09] we translate nominal terms to higher-order terms, and atoms-abstraction gets translated to functional abstraction. However, this does *not* mean that atoms-abstraction is a 'special case' of functional abstraction, any more than translating e.g. Java to machine binary means that method invocation is a special case of logic gates.

*The syntax of this paper, and previous work.* The modal and contextual systems which we give semantics to in this paper, are taken from previous work. Specifically, Definition 2.6 corresponds to [PD01, Subsection 2.2], Definition 5.4 corresponds to the term section opening [NPP08, Subsection 4.1], Figure 1 corresponds to [PD01, Subsection 2.3] and Figure 6 to [NPP08, Figure 3].

We feel that this paper does make some contribution in terms of presentation, and the exposition and definitions here may be tailored to a slightly different community.


### Acknowledgements

This paper was supported by Spanish MICINN Project TIN2010-20639 Paran10; AMAROUT grant PCOFUND-GA-2008-229599; Ramon y Cajal grants RYC-2010-0743 and RYC-2006-002131; and the Leverhulme Trust.


### References


[ACU10] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In *Foundations of software science and computation structures, 13th International Conference (FOSSACS 2010)*, volume 6014 of *Lecture Notes in Computer Science*, pages 297–311. Springer, 2010.

[AFZ03] Davide Ancona, Sonia Fagorzi, and Elena Zucca. A calculus for dynamic linking. In *ICTCS*, pages 284–301, 2003.

[AL91] Andréa Asperti and Giuseppe Longo. *Categories, types, and structures: an introduction to category theory for the working computer scientist*. Foundations of computing. MIT Press, 1991. Available online from the University of Michigan, digitised November 2007.

[AMdPR01] Natasha Alechina, Michael Mendler, Valeria de Paiva, and Eike Ritter. Categorical and Kripke semantics for Constructive S4 modal logic. In *Computer Science Logic, CSL'01*, volume 2142 of *Lecture Notes in Computer Science*, pages 292–307, 2001.

[BdP00] Gavin M. Bierman and Valeria C. V. de Paiva. On an intuitionistic modal logic. *Studia Logica*, 65(3):383–416, 2000.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

[CP12] Andrew Cave and Brigitte Pientka. Programming with binders and indexed data-types. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*, pages 413–424. ACM Press, 2012.

[DP01] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[FP10] Amy Felty and Brigitte Pientka. Reasoning with higher-order abstract syntax and contexts: A comparison. In *Interactive Theorem Proving*, volume 6172 of *Lecture Notes in Computer Science*, pages 227–242, 2010.

[Gab11] Murdoch J. Gabbay. Stone duality for First-Order Logic: a nominal approach. In *Howard Barringer Festschrift*. December 2011.

[Gab12] Murdoch J. Gabbay. Nominal terms and nominal logics: from foundations to meta-mathematics. In *Handbook of Philosophical Logic*, volume 17. Kluwer, 2012.

[GKWZ03] Dov M. Gabbay, Agnes Kurucz, Frank Wolter, and Michael Zakharyaschev. *Many-dimensional modal logics: theory and applications*, volume 148 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2003.

[GM09] Murdoch J. Gabbay and Dominic P. Mulligan. Universal algebra over lambda-terms and nominal terms: the connection in logic between nominal techniques and higher-order variables. In *Proceedings of the 4th International Workshop on Logical Frameworks and Meta-Languages (LFMTP 2009)*, pages 64–73. ACM, August 2009.

[GM11] Murdoch J. Gabbay and Dominic Mulligan. Nominal Henkin Semantics: simply-typed lambda-calculus models in nominal sets. In *Proceedings of the 6th International Workshop on Logical Frameworks and Meta-Languages (LFMTP 2011)*, volume 71 of *EPTCS*, pages 58–75, September 2011.

[GP01] Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3–5):341–363, July 2001.

[Kob97] Satoshi Kobayashi. Monad as modality. *Theoretical Computer Science*, 175(1):29–74, 1997.

[Mit96] John C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[NP05] Aleksandar Nanevski and Frank Pfenning. Staged computation with names and necessity. *Journal of Functional Programming*, 15(6):893–939, 2005.

[NPP08] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

[PD01] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4), 2001.

[PD08] Brigitte Pientka and Joshua Dunfield. Programming with proofs and explicit contexts. In *Proceedings of the 10th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP 2008)*, pages 163–173, 2008.

[Pie08] Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM Press, 2008.

[PP03] Brigitte Pientka and Frank Pfenning. Optimizing higher-order pattern unification. In *Proceedings of the International Conference on Automated Deduction (CADE'03)*, volume 2741 of *Lecture Notes in Computer Science*, pages 473–487, 2003.

[Sim94] Alex K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.

[UPG03] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal Unification. In *CSL*, volume 2803 of *Lecture Notes in Computer Science*, pages 513–527. Springer, December 2003.

[UPG04] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal Unification. *Theoretical Computer Science*, 323(1–3):473–497, September 2004.

[WLP98] Philip Wickline, Peter Lee, and Frank Pfenning. Run-time code generation and Modal-ML. In *Programming Language Design and Implementation (PLDI'98)*, pages 224–235. ACM Press, 1998.