Dependently-Typed Montague Semantics in the Proof Assistant Agda-flat

Colin Zwanziger
Department of Philosophy
Carnegie Mellon University
Pittsburgh, PA, USA
zwanzig@cmu.edu

Abstract

We apply the Agda-flat proof assistant (Vezzosi, 2019) to computational semantics. Computational semantics in Agda-flat is distinguished from the approach based on Coq (Chatzikyriakidis and Luo, 2014) in that it allows an implementation of the classical, intensional semantic analyses of Montague (1973). That is, it synthesizes the modern dependent type theory and Montague intensional logic traditions in the computational semantics setting. To demonstrate this, we show how to replicate Montague's analyses in the type theory of Zwanziger (2018), which closely corresponds to the Agda-flat system. Accompanying code type-checks these analyses in Agda-flat.

1 Introduction

Proof assistants, e.g. Coq (Chatzikyriakidis and Luo, 2014), have been applied in computational semantics to support natural language inference. By translating (declarative) sentences of a natural language into the language of a proof assistant, the problem of whether one natural language sentence implies another is reduced to finding a proof of implication in that proof assistant. Ideally, both the translation and the proof search are automated.

While proof assistants like Coq and the similar system Agda are adequate for implementing most semantic analyses given in the modern dependent type theory tradition (Ranta, 1994), they do not provide an obvious implementation of the classical intensional semantic analyses of Montague (1973). The criterion that a computational semantics system be general enough to capture Montague's analyses has been termed the "Montague Test" (Morrill and Valentín, 2016).

The present work introduces computational semantics in the Agda-flat proof assistant. Agda-flat (Vezzosi, 2019) is a variation on ordinary Agda implementing a so-called comonadic modal dependent type theory. Agda-flat was created by Andrea Vezzosi (c. 2017-2019) as a tool for verifying mathematical theorems in type-theoretical foundations. The logic behind the most recent mode of Agda-flat is essentially similar to **CHoTT**, which was isolated by Zwanziger (2018) as a natural dependently-typed, optionally hyperintensional analog of Montague's simply-typed intensional logic. As such, Agda-flat passes (the relevant incarnation of) the Montague Test. Furthermore, **CHoTT** can be used as a readable notation for Agda-flat.

Below, we develop enough of natural language semantics in **CHoTT** and Agda-flat to pass the Montague Test. In Section 2, we review the type theory **CHoTT** of Zwanziger (2018), which closely corresponds to the Agda-flat system, and is seen to provide a dependently-typed version of Montague intensional logic. In Section 3, the Montague Test sentences are translated into **CHoTT**. In Section 4, we discuss the Agda-flat system, and in particular the straightforward rendering of **CHoTT** sentences therein. Pursuant to the translations given in Section 3, an auxiliary repository is provided type-checking the translations of all Montague Test sentences.

2 Comonadic Homotopy Type Theory

We review the type theory **CHoTT** of Zwanziger (2018), which provides a dependently-typed version of the intensional logic **IL** of Montague (1973). To motivate **CHoTT**, a preview of the correspondence between **CHoTT** and **IL** is given in Subsection 2.1. Furthermore, some context for the development of **CHoTT** is provided in Subsection 2.2. The technical discussion of **CHoTT** is in Subsection 2.3.

IL	**CHoTT**	English Gloss
t	Prop	"type of truth values/propositions"
$\langle s, a\rangle$	$\flat A$	"type of intensions of terms of type a/A"
$\hat{}\alpha$	$t^\flat$	"the intension of term α/t"
$\check{}\alpha$	$t_\flat$	"the extension of term α/t"
$\Box\phi$	$\Box\phi$	"necessarily ϕ"

Table 1: An **IL**-to-**CHoTT** Dictionary

2.1 Cheat Sheet for CHoTT

As any extension of intensional logic should, the type theory **CHoTT** will include analogs of the intension, extension, and other operators of **IL**. These are foreshadowed in Table 1. The variant notation from **IL** follows Shulman (2018), and reflects the essentially independent origins of **IL** and Shulman's system.

Pursuant to the correspondence of Table 1, the translations of the Montague Test sentences given in Section 3 will bear an obvious similarity to those of Montague. There will be a few technical points of divergence between **CHoTT** and **IL**, though, discussed in the course of Subsection 2.3. Though these particularities of **CHoTT** can be justified on technical grounds, it is also worthwhile to understand **CHoTT** and its relation to **IL** in the context of the literature.

2.2 Provenance of CHoTT

We here review some historical context and justification for **CHoTT**. A reader less concerned by such matters can skip ahead to Subsection 2.3 for the technical development.

The type theory **CHoTT** is a "comonadic homotopy type theory", combining a version of Montague's intensional logic ("comonadic type theory") with homotopy type theory (HoTT), a version of dependent type theory. We visit each of these aspects in turn.

2.2.1 Intensional Logic and Comonadic Type Theory

The identification of Montague intensional logic with comonadic type theory dates to joint work of the author (Awodey et al., 2015), and is developed in the M.S. thesis of the author (Zwanziger, 2017), in both syntactic and semantic aspects. Comonadic type theory is a variety of modal type theory which was first treated systematically in the 1990's, with approaches due to Bierman and de Paiva (2000) and Davies and Pfenning (Pfenning and Davies, 2001). Montague's intensional logic can thus be understood retrospectively as the original comonadic type theory. However, Montague's syntax lacks β-conversion (AKA λ-conversion) and the usual rules for quantifiers, an issue addressable within these modern approaches (Zwanziger, 2017). The present paper follows the Davies and Pfenning approach, and and in particular Shulman (2018)'s extension to homotopy type theory.

2.2.2 Homotopy Type Theory

HoTT (Univalent Foundations Program, 2013) is a proposed type-theoretic alternative to axiomatic set theory as a foundation for math. HoTT incorporates dependent type theory, a framework which has been fruitfully applied in natural language semantics since Sundholm (1989) and Ranta (1994). Recently, dependent type theory has been called "modern type theory" in the context of natural language semantics, to distinguish it from the simply-typed intensional logic of Montague (Chatzikyriakidis and Luo, 2018). Particularly appealing applications of dependent type theory in natural languages semantics, dating to the early work, include the use of the baked-in bounded quantification to model the bounded quantification of natural language (part of the "common nouns-as-types" viewpoint) (Luo, 2012), and a natural approach to dynamic semantics.

As for the "homotopy" part, it is argued in Zwanziger (2018) that this provides a natural approach to hyperintensional natural language semantics. As a homotopy type theory, **CHoTT** includes two notions of equality: definitional equality, written $\equiv$, and thought of as (hyper)intensional equality, and typal equality, written $=$, and thought of as an *a posteriori* equality. In line with this intuition, $\equiv$ is stronger than $=$. In **CHoTT**, we find 'intensional' operators which respect only $\equiv$, not $=$. Furthermore, this more "granular" intensional equality need not conflate statements with

$$\frac{}{\cdot \mid \cdot \ \mathsf{ctx}}\ \text{ctx-Emp.}$$

$$\frac{\Delta \mid \Gamma \vdash B : U}{\Delta \mid \Gamma, x : B\ \mathsf{ctx}}\ \text{ctx-Ext.}^{e} \qquad \frac{\Delta \mid \Gamma, x : A, \Gamma'\ \mathsf{ctx}}{\Delta \mid \Gamma, x : A, \Gamma' \vdash x : A}\ \text{Var.}^{e}$$

$$\frac{\Delta \mid \cdot \vdash B : U}{\Delta, u : \{\flat\}\ B \mid \cdot\ \mathsf{ctx}}\ \text{ctx-Ext.}^{i} \qquad \frac{\Delta, u : \{\flat\}\ A, \Delta' \mid \Gamma\ \mathsf{ctx}}{\Delta, u : \{\flat\}\ A, \Delta' \mid \Gamma \vdash u : A}\ \text{Var.}^{i}$$

Table 2: The Extensional ($-^{e}$) and Intensional ($-^{i}$) Context Rules

the same truth conditions. A hyperintensional semantics of **CHoTT**, inspired by the homotopy-theoretic semantics of HoTT, is the subject of other research by the author (Zwanziger, 2019).

In summary, **CHoTT** and the closely related Agda-flat, while building directly on a separate tradition than intensional logic, do incorporate intensional logic, together with additional features. Furthermore, these extra features have various uses for the natural language semanticist, rather than being peculiarities inflicted on them by the Agda-flat system. That said, since our intention is to make a beeline for formalizing the Montague Test sentences, these modeling advantages will not be pursued below, with the exception of common nouns-as-types.

2.3 The System CHoTT

We now turn to delineating the system **CHoTT**, with a focus on the similarities and differences with **IL**.

Officially, **CHoTT** is the fragment of Shulman (2018) consisting of the usual notions of homotopy type theory, together with the comonadic type operator $\flat$, which we think of as an intension type operator performing the role of Montague's $\langle s, -\rangle$. In less condensed terms, we have the following:

In the tradition of Pfenning and Davies (2001) and Shulman (2018), **CHoTT** has two variable judgements,

$$u : \{\flat\}\ A$$

and

$$x : A$$

We will say (at variance with prior terminology) that "u is an intensional variable of type A," when $u : \{\flat\}\ A$ and that "x is an extensional variable of type A," when $x : A$.

Remark 1 *Intuitively, having an assumption $u : \{\flat\}\ A$ of an intensional variable of type A is akin to having an assumption $x : \flat A$ of an extensional variable of the type of intensions of terms of type A. Indeed, such assumptions will turn out to be interchangeable.*

The hypothetical judgements of **CHoTT** have the form

$$\Delta \mid \Gamma \vdash t : B$$

and

$$\Delta \mid \Gamma \vdash t \equiv u : B$$

where Δ represents a list of intensional typed variables, and Γ a list of extensional typed variables. This $\Delta \mid \Gamma$ is called the context, and we will have, as usual, that a type appearing in the context may depend only on typed variables to its left. So types in Γ can depend on variables in Δ, but not vice versa.

Due to the two variable judgements, there is a duplication of the rules for context extension and variables, with variants for extensional and intensional variables. These are given in Table 2. In view of Remark 1, the rule Var.^{i} is understood as the principle that an assumption of an intension of a term of type A yields a term of type A. This rule is, naturally, implicated in the derivation of the Montague extension operator, below.

We import the usual homotopy-type-theoretical notions (Univalent Foundations Program, 2013), including $\prod$- and $\sum$-types (analogs of the quantifiers $\forall$ and $\exists$), type universes and universe polymorphism, $=$-types, higher inductive types (HITs), and univalence. However, as a simplifying assumption, the typing rules are assumed to manipulate extensional variables only. For instance, the formation rule for $\prod$ is

$$\frac{\Delta \mid \Gamma \vdash A : U \qquad \Delta \mid \Gamma, x : A \vdash B : U}{\Delta \mid \Gamma \vdash \prod_{x:A} B : U}$$

$$\dfrac{\Delta \mid \cdot \vdash B : U}{\Delta \mid \Gamma \vdash \flat B : U}\ \flat\text{-Form. (Montague's } \langle s, - \rangle) \qquad \dfrac{\Delta \mid \cdot \vdash t : B}{\Delta \mid \Gamma \vdash t^\flat : \flat B}\ \flat\text{-Intro. (Montague's } {}^\wedge(-))$$

$$\dfrac{\Delta \mid \Gamma, x : \flat A \vdash B : U \qquad \Delta \mid \Gamma \vdash s : \flat A \qquad \Delta, u :\{\flat\}\, A \mid \Gamma \vdash t : B[u^\flat/x]}{\Delta \mid \Gamma \vdash (\mathsf{let}\ u^\flat := s\ \mathsf{in}\ t) : B[s/x]}\ \flat\text{-Elim.}$$

$$\dfrac{\Delta \mid \Gamma, x : \flat A \vdash B : U \qquad \Delta \mid \cdot \vdash s : A \qquad \Delta, u :\{\flat\}\, A \mid \Gamma \vdash t : B[u^\flat/x]}{\Delta \mid \Gamma \vdash \mathsf{let}\ u^\flat := s^\flat\ \mathsf{in}\ t \equiv t[s/u] : B[s^\flat/x]}\ \flat\text{-}\beta\text{-Conversion}$$

Table 3: The Rules for $\flat$

$$\dfrac{\dfrac{\Delta \mid \cdot \vdash B : U}{\Delta \mid \Gamma, x : \flat B \vdash B : U}\ \text{Weakening} \qquad \Delta \mid \Gamma \vdash t : \flat B \qquad \dfrac{}{\Delta, u :\{\flat\}\, B \mid \Gamma \vdash u : B}\ \text{Var.}^i}{\Delta \mid \Gamma \vdash (\mathsf{let}\ u^\flat := t\ \mathsf{in}\ u) \equiv: t_\flat : B}\ \flat\text{-Elim.}$$

Figure 1: Derivation of $\flat$-Elim.-Simp. (Montague's ${}^\vee(-)$)

in which $x : A$ is required to be extensional.

Finally, we have a comonad $\flat$ corresponding to Montague's $\langle s, - \rangle$, the rules for which appear as Table 3.

Note that the $\flat$-Form. and $\flat$-Intro. (intension operator) rules only apply "in an intensional context", that is, when no extensional variables are present in the context.[1] Thus, in a well-typed term, any intensional operator ($\flat(-)$ or $(-)^\flat$) that appears must have been adduced during a phase of the derivation with an intensional context.[2] Since variables within the scope of an intensional operator must receive *de re* interpretation (Zwanziger, 2017), and only intensional variables occur within the scope of an intensional operator, intensional variables can be motivated as a technical device for keeping track of which locations within a term receive *de re* interpretation.

The $\flat$-Elim. rule provides a way of substituting a term of type $\flat A$ for an intensional variable of type A via an explicit let-notation. This principle is in keeping with Remark 1. Furthermore, it gives a benign way of "substituting into an intensional context". To illustrate, such a substitution

$$\mathsf{let}\ u^\flat := s\ \mathsf{in}\ t(u^\flat)^\flat \quad ,$$

in which s receives *de re* interpretation, does not in general $\flat$-β-reduce to the *de dicto* form

$$t(s)^\flat \quad ,$$

and thus avoids the usual pitfall of substitution into intensional contexts. As a corollary, β-conversion (for function types) clearly gives

$$(\lambda x.\mathsf{let}\ u^\flat := x\ \mathsf{in}\ t(u^\flat)^\flat)(s) \equiv \\ \mathsf{let}\ u^\flat := s\ \mathsf{in}\ t(u^\flat)^\flat \quad ,$$

not

$$(\lambda x.\mathsf{let}\ u^\flat := x\ \mathsf{in}\ t(u^\flat)^\flat)(s) \equiv t(s)^\flat \quad ,$$

explaining why β-conversion is benign in **CHoTT**.

Though $\flat$-Elim. is subtle, the reader may take heart that we can use it to derive an 'extension' operator corresponding to Montague's ${}^\vee(-)$, which, again following Shulman (2018), we call $(-)_\flat$. That is, the rule

$$\dfrac{\Delta \mid \Gamma \vdash t : \flat B}{\Delta \mid \Gamma \vdash t_\flat : B}\ \flat\text{-Elim.-Simple}$$

is derivable (Figure 1). As foreshadowed above, this derivation also makes crucial use of the rule Var.^{i}.

Furthermore, we have the conversion $(t^\flat)_\flat \equiv t$, familiar from Montague. That is, the principle

$$\dfrac{\Delta \mid \cdot \vdash t : B}{\Delta \mid \Gamma \vdash (t^\flat)_\flat \equiv t : B}\ \flat\text{-}\beta\text{-Conv.-Simp.}$$

[1]While **IL** does not place any similar restriction on the intension operator ${}^\wedge(-)$, Montague does stipulate a constant-domain Kripke semantics for **IL**. The analogous Kripke-Montague semantics for **CHoTT** need not stipulate constant domains. But a type $\flat A$ of intensions will, naturally, have a constant domain interpretation regardless. Remark 1 relates extensional variables of type $\flat A$ with intensional variables of type A.

[2]In the current terminology, then, an intensional context is not, as traditionally, *defined* by the presence of an intensional operator, but rather a more primitive notion that is prerequisite for the presence of an intensional operator.

is derivable, obtained with

$$\begin{aligned}(t^\flat)_\flat &\equiv \mathsf{let}\ u^\flat := t^\flat\ \mathsf{in}\ u && (\text{Def'n. of } (-)_\flat)\\ &\equiv t && (\flat\text{-}\beta\text{-Conv.})\end{aligned}$$

as calculation.

It is with all the comonadic rules in concert that Remark 1 is ultimately justified. The reader may confirm, for instance, that the principles

$$\frac{u : \{\flat\}\ A \mid \cdot \vdash t : B}{\cdot \mid x : \flat A \vdash \mathsf{let}\ u^\flat := x\ \mathsf{in}\ t : \mathsf{let}\ u^\flat := x\ \mathsf{in}\ B}$$

and

$$\frac{\cdot \mid x : \flat A \vdash t : B}{u : \{\flat\}\ A \mid \cdot \vdash t[u^\flat/x] : B[u^\flat/x]}$$

are derivable.

3 Natural Language Semantics in CHoTT

We now set about rendering the Montague Test sentences in **CHoTT**.

To illustrate the issues involved, let's examine what will be the translations of the sentence, "John believes that a fish walks." This sentence is traditionally held to have two construals, a *de dicto* and a *de re*. These construals are to be represented by

$$believe(j, (\exists(x : fish).walk(i(x)))^\flat)$$

and

$$\exists(x : fish).\mathsf{let}\ u^\flat := i(x)\ \mathsf{in}\ believe(j, walk(u^\flat)^\flat) \quad ,$$

respectively. These formulas exhibit several less familiar features which bear discussion, including defined (non-primitive) notation for logic inside HoTT ($\exists$), common nouns-as-types ($x : fish$) and associated type coercions (i), and the handling of *de re* with (let-substitutions for) intensional variables.

In Subsection 3.1, we explicate any prerequisite defined notation and associated theory. In Subsection 3.2, we give the **CHoTT** renderings of the Montague Test sentences.

3.1 Defined Notations

We now develop several notions for eventual use in Subsection 3.2. Unless otherwise specified, the definitions of the present subsection are adapted from Chapter 3 of the HoTT Book (Univalent Foundations Program, 2013).

3.1.1 Predicate Logic

Central to the approach of the current section is the use of the type Prop of "propositions" in lieu of Montague's type t of "truth values". Within HoTT, propositions are identified with those types that have at most one term (up to the equality $=$). Intuitively, such types encode no more (extensional) information than whether they have a term ("are inhabited") or not. In such case, truth is identified with inhabitation. Formally speaking, we have the following definitions:

Definition 2 *For any type* $A : U$, *let* $\mathrm{isProp}(A) :\equiv \prod_{x,y:A} x = y$.

When $\mathrm{isProp}(A)$ is inhabited, we say that A is a proposition, and when such A is inhabited, we say A is true.

Definition 3 $\mathrm{Prop} :\equiv \sum_{A:U} \mathrm{isProp}(A)$

Whereas a sequent of form $\Delta \mid \Gamma, x : A \vdash B : U$ is thought of as a type depending on A, a sequent of form $\Delta \mid \Gamma, x : A \vdash P : \mathrm{Prop}$ may be thought of as a predicate on A, rather than a "proposition depending on A". This notion of predicate gives rise to a notion of predicate logic inside type theory with the following constructs:

Definition 4

$$\begin{aligned}\top &:\equiv 1\\ \bot &:\equiv 0\\ P \wedge Q &:\equiv P \times Q\\ P \vee Q &:\equiv \|P + Q\|\\ P \Rightarrow Q &:\equiv P \to Q\\ \neg P &:\equiv P \Rightarrow \bot\\ \forall(x : A)P(x) &:\equiv \prod_{x:A} P(x)\\ \exists(x : A)P(x) &:\equiv \|\sum_{x:A} P(x)\| \quad ,\end{aligned}$$

where $\Delta \mid \Gamma, x : A \vdash P, Q : \mathrm{Prop}$.

Here, $\|-\|$ denotes "propositional truncation", as defined in Chapter 3 of the HoTT Book (Univalent Foundations Program, 2013), which quotients any type to a proposition.

To the logic of the HoTT Book, we do add one definition:

Definition 5 *Let* $\Delta, u : \{\flat\}\ A \mid \cdot \vdash P : \mathrm{Prop}$. *Then* $\Box P :\equiv \|\flat P\|$.

This pregnant and perhaps surprising definition can be understood in terms of the type-theoretic

$$
\begin{aligned}
E &: U \\
j, b &: \flat E \\
walk, man, talk, fish, woman, unicorn, park &: \flat E \to \mathrm{Prop} \\
believe &: \flat\,\mathrm{Prop} \to \flat E \to \mathrm{Prop} \\
seek, catch, eat, find, love, lose &: \flat(\flat(\flat E \to \mathrm{Prop}) \to \mathrm{Prop}) \to \flat E \to \mathrm{Prop} \\
slowly, try &: \flat(\flat E \to \mathrm{Prop}) \to \flat E \to \mathrm{Prop} \\
in &: \flat(\flat(\flat E \to \mathrm{Prop}) \to \mathrm{Prop}) \to \flat(\flat E \to \mathrm{Prop}) \to \\
&\quad\; \flat E \to \mathrm{Prop}
\end{aligned}
$$

Figure 2: Lexicon for the Montague Test

tendency to conflate truth with inhabitation. In Kripke-Montague semantics, intensions for (the interpretation of) a type A only exist when (the domain interpreting) A at each world is non-empty. So $\flat A$ at a world is non-empty (*i.e.* true) when A is nonempty (*i.e.* true) at every world.[3]

3.1.2 Subtypes and Coercions

We will have reason to view common nouns as types (for the purposes of type-bounded quantification). Yet predicates like "talks" are apt to apply to women, men, donkeys (perhaps), *etc*. Consequently, "talks" is better interpreted as a predicate on a type of entities subsuming women, men, and donkeys. To accommodate both impulses, we require a notion of subtype and type coercion, which will allow us to write logical forms like, *e.g.*, $\forall(x : man).talk(i(x))$ for "Every man talks." This motivates our use of the subtypes of HoTT, defined as follows:

Definition 6 *Let* $\Delta \mid \Gamma, x : A \vdash P : \mathrm{Prop}$. *Then* $\{x : A \mid P(x)\} :\equiv \sum_{x:A} P(x)$. *Furthermore let* $i :\equiv \pi_1 : \sum_{x:A} P(x) \to A$.

This $\{x : A \mid P(x)\}$ is said to be the subtype of A satisfying property P, and i the inclusion or coercion of $\{x : A \mid P(x)\}$ into A.

For readability, we will furthermore coin the following abuse of notation:

Convention 7 *The type* $\{x : A \mid P(x)\}$, *where* $\Delta \mid \Gamma, x : A \vdash P : \mathrm{Prop}$, *may simply be denoted by* P, *where confusion is unlikely.*

[3]It would be more satisfying to have simply $\Box P :\equiv \flat P$, but **CHoTT** is apparently too weak to prove that $\flat P$ is a proposition, as desired of $\Box P$. This gives a hint that **CHoTT**, despite its advantages, may not be the last word in comonadic type theory.

The formula $\forall(x : man).talk(i(x))$ is now understood as using Convention 7, as well as a coercion.

3.2 The Montague Test Sentences

With the benefit of the foregoing, we now render the Montague Test sentences in **CHoTT**. We will treat exactly the sentence suite formulated by Morrill and Valentín (2016) in introducing the Montague Test.

We assume a ground type E, together with a number of constants that serve as the translations of natural language terms, detailed in Figure 2. These are simply taken from Dowty *et al.* (1981), Chapter 7, using the correspondences suggested by Table 1, plus a single important stipulation: wherever the type e appears for Dowty *et al.*, $\flat E$ (rather than E) appears for us. This unfortunate technical device is necessitated by our approach to *de re* readings, which makes use of let-substitution. One can substitute terms of type $\flat E$ using let, but not E.[4]

The **CHoTT** forms for the Montague Test sentences are given in Table 4, below. Any reference numbers come from Chapter 7 of Dowty *et al.* (1981).

Overall, **CHoTT** formulas of Table 4 closely reflect those given by Montague (1973). To illustrate the similarities and differences, we return, at last, to the translations of "John believes that a fish walks." The *de dicto* reading is translated as

$$believe(j, (\exists(x : fish).walk(i(x)))^\flat) \quad .$$

[4]One way around this problem would be to use adjoint type theory (Benton and Wadler, 1996; Licata and Shulman, 2016), in which some ground types already have "constant domain", in lieu of comonadic type theory.

Ref. No.	**Sentence**	**Translation(s)**
7	John walks.	$walk(j)$
16	Every man talks.	$\forall(x : man).talk(i(x))$
19	The fish walks.	$\exists(x : fish).walk(i(x)) \wedge \forall(y : fish).\|y = x\|$
32	Every man walks or talks.	$\forall(x : man).walk(i(x)) \vee talk(i(x))$
34	Every man walks or every man talks.	$(\forall(x : man).walk(i(x))) \vee (\forall(x : man).talk(i(x)))$
39	A woman walks and she talks.	$\exists(x : woman).walk(i(x)) \wedge talk(i(x))$
43 45	John believes that a fish walks.	$believe(j, (\exists(x : fish).walk(i(x)))^\flat)$ $\exists(x : fish).\mathsf{let}\ u^\flat := i(x)\ \mathsf{in}\ believe(j, walk(u^\flat)^\flat)$
48 49 n/a	Every man believes that a fish walks.	$\exists(x : fish).\mathsf{let}\ u^\flat := i_1(x)\ \mathsf{in}$ $\forall(y : man).believe(i_2(y), walk(u^\flat)^\flat)$ $\forall(y : man).\exists(x : fish).\mathsf{let}\ u^\flat := i_1(x)\ \mathsf{in}$ $believe(i_2(y), walk(u^\flat)^\flat)$ $\forall(y : man).believe(i_2(y), (\exists(x : fish).walk(i_1(x)))^\flat)$
57	Every fish such that it walks talks.	$\forall(x : fish \wedge walk).talk(i(x))$
60 62	John seeks a unicorn.	$seek(j, (\lambda(P : \flat(\flat E \to \text{Prop})).\exists(x : unicorn).P_\flat(i(x)))^\flat)$ $\exists(x : unicorn).\mathsf{let}\ u^\flat := i(x)\ \mathsf{in}$ $seek(j, (\lambda(P : \flat(\flat E \to \text{Prop})).P_\flat(u^\flat))^\flat)$
73	John is Bill.	$\|j = b\|$
76	John is a man.	$man(j)$
83	Necessarily, John walks.	$\Box(walk(j))$
86	John walks slowly.	$slowly(walk^\flat)(j)$
91	John tries to walk.	$try(j, walk^\flat)$
94	John tries to catch a fish and eat it.	$try(j, (\lambda(y : \flat E).\mathsf{let}\ u^\flat := y\ \mathsf{in}\ \exists(x : fish).$ $catch'((\lambda(P : \flat(\flat E \to \text{Prop})).P_\flat(u^\flat))^\flat)(i(x)) \wedge$ $eat'((\lambda(P : \flat(\flat E \to \text{Prop})).P_\flat(u^\flat))^\flat)(i(x)))^\flat)$
98	John finds a unicorn.	$\exists(x : unicorn).\mathsf{let}\ u^\flat := i(x)\ \mathsf{in}$ $find((\lambda(P : \flat(\flat E \to \text{Prop})).P_\flat(u^\flat))^\flat)(j)$
105	Every man such that he loves a woman loses her.	$\exists(y : woman).\mathsf{let}\ u^\flat := i_2(y)\ \mathsf{in}$ $\forall(x : man \wedge love((\lambda(P : \flat(\flat E \to \text{Prop})).P_\flat(u^\flat))^\flat)).$ $lose((\lambda(P : \flat(\flat E \to \text{Prop})).P_\flat(u^\flat))^\flat)(i_1(x))$
110	John walks in a park.	$\exists(x : park).\mathsf{let}\ u^\flat := i(x)\ \mathsf{in}$ $in((\lambda(P : \flat(\flat E \to \text{Prop})).P_\flat(u^\flat))^\flat)(walk^\flat)(j)$
116 118	Every man doesn't walk.	$\neg\forall(x : man).walk(i(x))$ $\forall(x : man).\neg walk(i(x))$

Table 4: Montague Test Sentences

Aside from being pleasingly compact, in keeping with dependent type theory's type-bounded quantification, this formula is hardly changed from Montague (1973).

Things get more interesting for the *de re* reading, which is represented by

$$\exists(x : fish).\mathsf{let}\ u^\flat := i(x)\ \mathsf{in}\ believe(j, walk(u^\flat)^\flat) \quad ,$$

making use of the let-substitution not familiar from **IL**. To explicate, it is perhaps helpful to think of $walk(u^\flat)$ as $walk(y)[u^\flat/y]$. That is, $walk(y)$ was altered to have an intensional variable, in keeping with Remark 1. As discussed in Section 2.3, the intensional variable u signals *de re* interpretation, and makes it possible to adduce an intension operator, forming $walk(u^\flat)^\flat$. We later desire a(n extensional) quantification, so we switch back to extensional variables by substituting into the intensional context, forming

$$\mathsf{let}\ u^\flat := i(x)\ \mathsf{in}\ believe(j, walk(u^\flat)^\flat) \quad ,$$

and then quantify.

In sum, this example, as well as the rest of the Montague Test sentences, are well-handled in **CHoTT**, and manifestly similarly to in **IL**.

4 Agda-flat

We now discuss Agda-flat, and the implementation therein of the Montague Test sentences, as rendered in **CHoTT** in the previous section.

4.1 About Agda-flat

Agda-flat (Vezzosi, 2019) is a proof assistant due to Andrea Vezzosi (c. 2017-2019). It is a branch of the Agda proof assistant (Norell, 2007) which grew out of Vezzosi's work on modal dependent type theory (Nuyts et al., 2017). Agda is similar to other dependently-typed proof assistants, like Coq. Agda-flat adds on a modal operator, $\flat$.

Agda-flat was intended as a proof assistant for mathematical theorems. Early mathematical uses include Licata *et al.* (2018) and Wellen (2018). The original version of Agda-flat is largely incompatible with linguistic applications, due to its relatively strong proof theory. Amongst other things, one can prove that $\flat$ is idempotent (that is, $\flat\flat A \simeq \flat A$), which does not hold in the familiar model of Montague (1973). Following a 2019 request from the author, Vezzosi implemented a new mode, called ("no-flat-split"), with a weaker proof theory. It is this "no-flat-split" mode which corresponds closely to **CHoTT**, and thus provides a setting for (optionally hyperintensional) Montague semantics.

4.2 Agda-flat for Computational Semantics

4.2.1 Installation and Configuration

Agda-flat is available for download at: `https://github.com/agda/agda/tree/flat`. Unfortunately, the process of installing Agda and turning on Agda-flat is system-dependent and potentially complex. The Agda Wiki (Agda Collaboration) can provide guidance, and one may wish to avail oneself of a friendly Agda user, including online.

As mentioned in Section 4.1, the "no-flat-split" option is necessary for computational semantics applications. To engage the "no-flat-split" option, place the code

```
{-# OPTIONS --no-flat-split #-}
```

at the beginning of a module.

4.2.2 Use

Agda-flat and **CHoTT** are similar, with both admitting a $\flat$-operator, and in particular having a way of restricting the $\flat$-Form. and $\flat$-Intro operations. Whereas in **CHoTT**, this is achieved by restricting these operations to an intensional context, in Agda, the variable contexts are not made explicit, and instead heavy use is made of $\prod$-types. Of necessity, then, the $\prod$-types of Agda-flat are permitted to be of the form $\prod_{u:\{\flat\}\,A} P$, in which an intensional variable is bound. In "Agda notation", however, $\prod_{u:\{\flat\}\,A} P$ is written `(u :{♭} A) → P`.

Agda-flat allows a similar manipulation of intensional variables by the rules for inductive types. Furthermore, $\flat$ is actually defined as an inductive type in Agda-flat, and uses pattern matching instead of an Elim. rule, as for any inductive types in Agda.

That is, $\flat$ is defined as an inductive type by, in pseudo-code,

```
data ♭ (A :{♭} Type) :  Type where
  int :  (a :{♭} A) → ♭A          .
```

Whenever the $\flat$-Elim. rule would be used in **CHoTT**, we instead use pattern matching. For instance, we define Montague's extension operator thus:

```
ext:  {A :{♭} Type} → (♭A  →  A)
ext (int u) = u               .
```

Once the $\flat$ operator is defined, and the above stylistic differences between **CHoTT** and Agda-flat accounted for, the rendering of **CHoTT** terms in Agda-flat proceeds as expected.

4.2.3 The Montague Test

The logical forms given in Table 4 are rendered in type-checkable code, available on the GitHub of the author at `https://github.com/zwanzigerc/Montague-Test`. This repository makes use of the conventions and codebase of the HoTT-Agda library (Brunerie et al.). In addition to the "official" renderings, there are also alternative ones which use the universe `Type` in lieu of the more complex `Prop`.

Agda-flat thus satisfies the Montague Test, and is seen to accommodate computational dependently typed Montague semantics.

5 Future Work

With the basic setup of intensional computational semantics in Agda-flat achieved, a key goal becomes automation, both of the translation from natural language into Agda-flat, and of proof search. Since Agda is so similar to Coq, it is possible that Agda-flat could be integrated with the translation from Grammatical Framework used in FraCoq (Bernardy and Chatzikyriakidis, 2017). As for proof search, Agda has some limited automation options, such as the command "auto", which should be explored. Ultimately the performance of Agda-flat at natural language inference should be compared to other systems that satisfy the Montague Test, such as CatLog3 (Morrill, 2017).

Agda-flat (without the assumption of intensional =-induction) is furthermore an appropriate system to implement *hyper*intensional semantics in the style of Zwanziger (2018). A full discussion of this is deferred to later work.

Acknowledgments

This paper was made possible by Andrea Vezzosi's programming of Agda-flat. In particular, many thanks are due for the new option, "no-flat-split", which makes the system general enough for the purposes of the present paper. Felix Wellen and Zesen Qian provided valuable help with Agda. And thank you to Steve Awodey for suggesting and supporting my focus on Agda-flat.

The logical forms given for purposes of the Montague Test (which in particular do not use common noun types to model verb selectional restrictions) were influenced by personal communication with Carl Pollard.

Any errors in the present work are my own.

References

The Agda Collaboration. The Agda wiki. https://wiki.portal.chalmers.se/agda/pmwiki.php. Accessed: 2019-04-22.

Steve Awodey, Ulrik Buchholtz, and Colin Zwanziger. 2015. Comonadic categorical semantics of montague's intensional logic. In *Slides from Dynamics Semantics Workshop: Modern Type Theoretic and Category Theoretic Approaches*, Ohio State University, Columbus, Ohio.

Nick Benton and Philip Wadler. 1996. Linear logic, monads and the lambda calculus. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 420–431. IEEE.

Jean-Philippe Bernardy and Stergios Chatzikyriakidis. 2017. A type-theoretical system for the FraCaS test suite: Grammatical framework meets Coq. In *Long Papers of the 12th International Conference on on Computational Semantics (IWCS)*.

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

Guillaume Brunerie, Kuen-Bang Hou (Favonia), Evan Cavallo, Tim Baumann, Eric Finster, Jesper Cockx, Christian Sattler, Chris Jeris, Michael Shulman, et al. Homotopy type theory in Agda. https://github.com/HoTT/HoTT-Agda. Accessed: 2019-04-22.

Stergios Chatzikyriakidis and Zhaohui Luo. 2014. Natural language inference in Coq. *Journal of Logic, Language and Information*, 23(4):441–480.

Stergios Chatzikyriakidis and Zhaohui Luo. 2018. *Formal Semantics in Modern Type Theories*. Wiley and ISTE Science Publishing Ltd.

David R. Dowty, Robert Wall, and Stanley Peters. 1981. *Introduction to Montague Semantics*. Springer Science and Business Media.

Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. 2018. Internal universes in models of homotopy type theory. *arXiv preprint arXiv:1801.07664*.

Daniel R. Licata and Michael Shulman. 2016. Adjoint logic with a 2-category of modes. In *International Symposium on Logical Foundations of Computer Science*, pages 219–235. Springer.

Zhaohui Luo. 2012. Common nouns as types. In *Proceedings of the International Conference on Logical Aspects of Computational Linguistics*, pages 173–185. Springer.

Richard Montague. 1973. The proper treatment of quantification in ordinary English. In K. Hintikka, J. Moravcsik, and Suppes P., editors, *Approaches to Natural Language*, pages 221–242. D. Reidel, Dordrecht.

Glyn Morrill. 2017. Parsing logical grammar: CatLog3. In *Proceedings of the Workshop on Logic and Algorithms in Computational Linguistics 2017*, pages 107–131, Stockholm. Stockholm University.

Glyn Morrill and José Oriol Valentín. 2016. Computational coverage of type logical grammar: The Montague Test. In *Empirical Issues in Syntax and Semantics 11*, pages 141–170.

Ulf Norell. 2007. Towards a practical programming language based on dependent type theory. Ph.D. thesis, Chalmers University of Technology.

Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. 2017. Parametric quantifiers for dependent type theory. In *Proceedings of the ACM on Programming Languages, ICFP*. ACM.

Frank Pfenning and Rowan Davies. 2001. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540.

Aarne Ranta. 1994. *Type-Theoretical Grammar*. Oxford University Press.

Michael Shulman. 2018. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Mathematical Structures in Computer Science*, 28(6):856–941.

Göran Sundholm. 1989. Constructive generalized quantifiers. *Synthese*, 79(1):1–12.

The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, Princeton.

Andrea Vezzosi. 2019. Agda-flat. `https://github.com/agda/agda/tree/flat`. Accessed: 2019-04-22.

Felix Wellen. 2018. Flat. `https://github.com/felixwellen/DCHoTT-Agda/blob/master/Flat.agda`. Code library. Accessed: 2019-04-22.

Colin Zwanziger. 2017. Montague's intensional logic as comonadic type theory. M.S. thesis, Carnegie Mellon University, Pittsburgh.

Colin Zwanziger. 2018. Propositional attitude operators in homotopy type theory: Extended abstract. `https://colinzwanziger.com/wp-content/uploads/2019/06/nlcs-2018-extended-abstract.pdf`. In *Fifth Workshop on Natural Language and Computer Science*, Oxford.

Colin Zwanziger. 2019. Natural model semantics for comonadic and adjoint modal type theory. In *Proceedings of the Second Applied Category Theory Conference*, Compositonality.