Noname manuscript No.
(will be inserted by the editor)

Dependent Types for Pragmatics

Darryl McAdams · Jonathan Sterling

Received: date / Accepted: date

Abstract This paper proposes the use of dependent types for pragmatic phenomena such as pronoun binding and presupposition resolution as a type-theoretic alternative to formalisms such as Discourse Representation Theory and Dynamic Semantics.

Keywords Semantics · Pragmatics · Pronouns · Presuppositions · Type Theory · Dependent Types

1 Introduction

In this paper, we discuss the possible use of dependent types as a metatheory for semantics which gives rise to very natural solutions to problems in pragmatics such as pronominal reference and presuppositions. The approach also gives a simple account of donkey anaphora without resorting to exotic scope extension of the sort used in Discourse Representation Theory and Dynamic Semantics, thanks to the proof-relevant nature of type theory.

Section 2 briefly covers the differences between simple types and dependent types for those unfamiliar. Section 3 discusses the sorts of meanings that would be possible under dependent types, together with an extension to the

Darryl McAdams
...
...
...
...
...

Jonathan Sterling
...
...
...
...

type theory that makes it possible to assign pronouns a meaning that is free of syntactic indices. Section 4 wraps up with a discussion of further extensions that could be made to the framework, on both theoretical and empirical grounds.

2 Dependent Types

Dependent types are a type theoretic approach to first-order and higher-order logic. In line with standard type theoretic approaches to logic, dependent type theory justifies its introduction and elimination rules by demonstrating local soundness and completeness in the form of β reduction and η expansion rules.

To make the explanation of the two main dependent connectives simpler, consider their non-dependent counterparts. A typical type system will have pair types (e.g. $A \times B$) and function types (e.g. $A \to B$). The formation, introduction, and elimination rules for the $\times$ connective are given by the following inference rules:

$$\frac{\Gamma \vdash A\ \mathit{type} \qquad \Gamma \vdash B\ \mathit{type}}{\Gamma \vdash A \times B\ \mathit{type}}\ \times\mathrm{F}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N\rangle : A \times B}\ \times\mathrm{I}$$

$$\frac{\Gamma \vdash P : A \times B}{\Gamma \vdash \mathsf{fst}(P) : A}\ \times\mathrm{E}_1$$

$$\frac{\Gamma \vdash P : A \times B}{\Gamma \vdash \mathsf{snd}(P) : A}\ \times\mathrm{E}_2$$

These are justified by the reductions

$$\mathsf{fst}(\langle M, N\rangle) \Rightarrow_\beta M \qquad \mathsf{snd}(\langle M, N\rangle) \Rightarrow_\beta N$$

which witnesses local soundness, and the expansion

$$P \Rightarrow_\eta \langle \mathsf{fst}(P), \mathsf{snd}(P)\rangle$$

which witnesses local completeness. Similarly, the introduction and elimination rules for $\to$ are

$$\frac{\Gamma \vdash A\ \mathit{type} \qquad \Gamma \vdash B\ \mathit{type}}{\Gamma \vdash A \to B\ \mathit{type}}\ \to\mathrm{F}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\, M : A \to B}\ \to\mathrm{I}$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ \to\mathrm{E}$$

with reduction and expansion rules

$$(\lambda x.\, M)\, N \Rightarrow_\beta [N/x]M \qquad M \Rightarrow_\eta \lambda x.\, Mx$$

where $[N/x]M$ stands for the result of substituting N for x in M.

Together, these rules consist of a definition of the simply typed λ calculus with pairs and functions, usually called $\lambda^{\times,\to}$. The extension to dependent pairs and functions is relatively straightforward. The judgment A *type* above, which is used to specify that something is a type, will be replaced by a typing judgment of the form $A : \mathsf{Set}$, where Set is the type of types.[1] The dependent pair inference rules are:[2]

$$\frac{\Gamma \vdash A : \mathsf{Set} \qquad \Gamma, x : A \vdash B : \mathsf{Set}}{\Gamma \vdash (x : A) \times B : \mathsf{Set}} \times\mathrm{F}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : [M/x]B}{\Gamma \vdash \langle M, N\rangle : (x : A) \times B} \times\mathrm{I}$$

$$\frac{\Gamma \vdash P : (x : A) \times B}{\Gamma \vdash \mathsf{fst}(P) : A} \times\mathrm{E}_1$$

$$\frac{\Gamma \vdash P : (x : A) \times B}{\Gamma \vdash \mathsf{snd}(P) : [\mathsf{fst}(P)/x]B} \times\mathrm{E}_2$$

The formation rule modifies the non-dependent case by letting the type B be defined in terms of a variable $x : A$, that will refer ultimately to the first element of a pair of the type $(x : A) \times B$. An element of $(x : A) \times B$ is just a pair as usual, but where the second element N has the type gotten from specifying x to be M (the first element) in B. The elimination rules are correspondingly augmented to reflect the dependency of the intro.

As an example of such a pair, imagine we extended our system to have a type of natural numbers $\mathbb{N}$, with inhabitants given by numerals(i.e. $0 : \mathbb{N}$, $1 : \mathbb{N}$, $27 : \mathbb{N}$, etc.), and a predicate $\mathit{Prime} : \mathbb{N} \to \mathsf{Set}$, such that $\mathit{Prime}\, n$ is inhabited by $\star$ if and only if n is prime. That is to say, we expect it to hold that $\star : \mathit{Prime}\, 2$ and $\star : \mathit{Prime}\, 7$ but not $\star : \mathit{Prime}\, 4$. Then the type $(n : \mathbb{N}) \times \mathit{Prime}\, n$ is the dependent type analog to the proposition $\exists x : \mathbb{N}.\mathit{Prime}\, n$. Inhabitants of this proposition, such as the pairs $\langle 2, \star\rangle$ and $\langle 7, \star\rangle$ are simple witnesses to the existence, paired with *proofs that the proposition holds of the witness*. As such, it cannot hold that $\langle 4, \star\rangle$ inhabits the type $(n : \mathbb{N}) \times \mathit{Prime}\, n$.

The β and η rules for dependent pairs are identity to the reduction rules for non-dependent pairs. Just in case x is not free in B, we can take the syntax $A \times B$ to be syntactic sugar for $(x : A) \times B$.

Analogous to the modifications to pairs, the generalization of functions to the dependent case is given via the following rules:

[1] This will lead to inconsistency if $\mathsf{Set} : \mathsf{Set}$, due to Girard's Paradox (Hurkens 1995), but standard techniques for avoiding this can be applied, such as universe hierarchies (Martin-Löf 1984). For simplicity of presentation, we embrace temporary inconsistency.

[2] This paper opts to use the notation $(x : A) \times B$ and $(x : A) \to B$ in place of the more common $\Sigma x : A.B$ and $\Pi x : A.B$, respectively, in order to emphasize that these are merely dependent versions of pairs and functions.

$$\frac{\Gamma \vdash A : \mathsf{Set} \qquad \Gamma, x : A \vdash B : \mathsf{Set}}{\Gamma \vdash (x : A) \to B : \mathsf{Set}} \to\mathrm{F}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\, M : (x : A) \to B} \to\mathrm{I}$$

$$\frac{\Gamma \vdash M : (x : A) \to B \qquad \Gamma \vdash N : B}{\Gamma \vdash MN : [N/x]B} \to\mathrm{E}$$

Just as dependent pairs were pairs where the type of the second element was dependent on the value of the first element, a dependent function is a function where the return type is dependent on the argument value. Continuing with the previous example, the type $(n : \mathbb{N}) \to \mathit{Prime}\, n$ corresponds to the proposition $\forall n : \mathbb{N}.\mathit{Prime}\, n$, which would be proved by a function sending each natural number n to a proof that n is prime. Of course no such function exists, so this type should be uninhabited.

As with pairs, the β and η rules remain unchanged, and we use the notation $A \to B$ as syntactic sugar for $(x : A) \to B$ just in case x is not free in B.

3 Dependent Types for Pragmatics

In a standard Dynamic Semantics approach to pronouns, the discourse "A man walked in. He sat down." would be represented by a proposition such as

$$(\exists x : \mathsf{E}.\mathit{Man}\, x \wedge \mathit{WalkedIn}\, x) \wedge \mathit{SatDown}\, x$$

In standard presentations of semantics, of course, this would be a malformed proposition, because x is out of scope in the right conjunct, however in Dynamic Semantics, the scope of existentials is extended to make this a well-formed proposition. Using a dependently typed formalism, such a sentence would have the semantics

$$(p : (x : \mathsf{E}) \times \mathit{Man}\, x \times \mathit{WalkedIn}\, x) \times \mathit{SatDown}\, (\mathsf{fst}(p))$$

Rather than modifying the behavior of existentials, which in dependent types become pairs, we instead use a dependent pair type in place of the conjunction. Conjunctions would become pair types regardless, but by using an explicitly dependent pair, the dependent framework allows the right conjunct to refer to not only the propositional content of the left conjunct, but also to the *witnesses* to the existentially quantified proposition, by way of $\mathsf{fst}()$.

The semantics for *a*, *man*, *walked in*, and *sat down* are, in simplified form, just direct translations from the usual semantic representations:

$$\begin{aligned}
&[\![\mathrm{a}]\!] : (\mathsf{E} \to \mathsf{Set}) \to (\mathsf{E} \to \mathsf{Set}) \to \mathsf{Set} \\
&[\![\mathrm{a}]\!] = \lambda P.\, \lambda Q.\, (x : \mathsf{E}) \times P\, x \times Q\, x
\end{aligned}$$

$$[\![\text{man}]\!] : \mathsf{E} \to \mathsf{Set}$$
$$[\![\text{man}]\!] = \mathit{Man}$$

$$[\![\text{walked in}]\!] : \mathsf{E} \to \mathsf{Set}$$
$$[\![\text{walked in}]\!] = \mathit{WalkedIn}$$

$$[\![\text{sat down}]\!] : \mathsf{E} \to \mathsf{Set}$$
$$[\![\text{sat down}]\!] = \mathit{SatDown}$$

The semantics of conjunction (in the form of sentence sequencing) and the pronoun *he* will be discussed briefly, however, for now the following definitions will suffice:

$$[\![\text{S}_1.\ \text{S}_2.]\!] : \mathsf{Set}$$
$$[\![\text{S}_1.\ \text{S}_2.]\!] = (p : [\![\text{S}_1]\!]) \times [\![\text{S}_2]\!]$$

$$[\![\text{he}]\!] : \mathsf{E}$$
$$[\![\text{he}]\!] = \mathsf{fst}(p)$$

We will need to return to these because they will not work on the face of it. How, for instance, does *he* know that it should mean $\mathsf{fst}(p)$ and not $\mathsf{snd}(p)$? Using $\mathsf{fst}(p)$ works for these carefully constructed examples, but in general it evidently will not. This deficiency will be resolved, but only after the shape of the semantics has been further discussed.

Consider now the discourse "A man walked in. The man (then) sat down." The use of *the man* in the right conjunct, instead of *he* introduces presuppositional content via the definite determiner. Ideally, the semantics of this should be nearly identical to those of the previous example (modulo β reduction). By giving *the* a dependently typed meaning, we can do this relatively simply:

$$[\![\text{the}]\!] : (P : \mathsf{E} \to \mathsf{Set}) \to (x : \mathsf{E}) \to P\,x \to E$$
$$[\![\text{the}]\!] = \lambda P.\,\lambda x.\,\lambda p.\,x$$

The first argument to *the* is simple the predicate, which in this case will be *Man*. The second argument is an entity, and the third is an inhabitant of the type $P\,x$, i.e. a proof that $P\,x$ holds. Therefore we would want

$$[\![\text{the man}]\!] = (\lambda P.\,\lambda x.\,\lambda p.\,x)\ (\mathit{Man}\,(\mathsf{fst}(p)))\ (\mathsf{fst}(\mathsf{snd}(p))) =_\beta \mathsf{fst}(p)$$

The term $\mathsf{fst}(p) : \mathsf{E}$ is of course the aforementioned man from the left conjunct. $\mathsf{snd}(p)$ is a proof that he is in fact a man, and that he walked in, and so $\mathsf{fst}(\mathsf{snd}(p))$ is the proof that he is a man. The argument $\mathsf{fst}(p)$ is, in effect, the witness to the existence component of the presupposition that *the* has, and $\mathsf{fst}(\mathsf{snd}(p))$ is the proof that the propositional component of the presupposition holds.

This definition will again be returned to later, because we have similar problems as with *he*, namely, how do we pick the right term for x and for the proof of $P\,x$? But for this example, it suffices that we pick them as shown.

The next two pairs of examples go hand in hand. Consider the classic donkey anaphora sentences "If a farmer owns a donkey, he beats it." and "Every farmer who owns a donkey beats it." A typical Dynamic Semantics approach might assign these sentences the meaning

$$\forall x : \mathsf{E}.\mathit{Farmer}\,x \wedge (\exists y : \mathsf{E}.\mathit{Donkey}\,y \wedge \mathit{Owns}\,x\,y) \Rightarrow \mathit{Beats}\,x\,y$$

In the dependently typed setting, we can assign a similar meaning, but which has a more straightforward connection to the syntax:

$$\begin{aligned}(p : (x : \mathsf{E}) \times \mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Own}\,x\,y)\\ \to \mathit{Beat}\,(\mathsf{fst}(p))\,(\mathsf{fst}(\mathsf{snd}(\mathsf{snd}(p))))\end{aligned}$$

For convenience, we can define a subscript notation p_i which projects the i-th element of a (right nested) tuple; now we have:

$$(p : (x : \mathsf{E}) \times \mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Owns}\,x\,y) \to \mathit{Beats}\,p_1\,p_3$$

The lexical entries for the content words and pronouns should be obvious at this point, but for *if*, *a*, and *every* we can define:

$$\begin{aligned}[\![\text{if}]\!] &: \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set}\\ [\![\text{if}]\!] &= \lambda P.\,\lambda Q.\,(p : P) \to Q\end{aligned}$$

$$\begin{aligned}[\![\text{a}]\!] &: (\mathsf{E} \to \mathsf{Set}) \to (\mathsf{E} \to \mathsf{Set}) \to \mathsf{Set}\\ [\![\text{a}]\!] &= \lambda P.\,\lambda Q.\,(x : \mathsf{E}) \times P\,x \times Q\,x\end{aligned}$$

$$\begin{aligned}[\![\text{every}]\!] &: (\mathsf{E} \to \mathit{Type}) \to (\mathsf{E} \to \mathsf{Set}) \to \mathsf{Set}\\ [\![\text{every}]\!] &= \lambda P.\,\lambda Q.\,(p : (x : \mathsf{E}) \times P\,x) \to Q(\mathsf{fst}(p))\end{aligned}$$

With these, we can get:

$[\![$a farmer owns a donkey$]\!]$
$= (x : \mathsf{E}) \times \mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Owns}\,x\,y$

$[\![$if a farmer owns a donkey$]\!]$
$= \lambda Q.\,(p : (x : \mathsf{E}) \times \mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Owns}\,x\,y) \to Q$

$[\![$farmer who owns a donkey$]\!]$
$= \lambda x.\,\mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Owns}\,x\,y$

$[\![$every farmer who owns a donkey$]\!]$
$= \lambda Q.\,(p : (x : \mathsf{E}) \times \mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Owns}\,x\,y)$
$\to Q\,p_1$

$[\![$beats it$]\!]$
$= \lambda z.\,\mathit{Beats}\,z\,p_3$

$[\![$he beats it$]\!]$
$= \mathit{Beats}\,p_1\,p_3$

Again, the pronouns need to have the right meanings, which here is just given as a temporary, just-so solution.

3.1 Terms for Presuppositions

Providing that the pronouns and definite determiners can be assigned the correct terms, the meanings given above work just as well as a standard DRT or dynamic meaning, but in a type safe way. But can these meanings in fact be gotten?

A number of possible solutions exist to do precisely this sort of thing in the programming language literature. Haskell's type class constraints (Marlow 2010) and Agda's instance arguments (Devriese and Piessens 2011) provide very similar functionality but for somewhat different purposes, so one option would be to steal those ideas. Unfortunately, doing so would complicate the theory, which is already complex enough due to dependency. The approach we will take here involves a new term that binds variables for presupposed parts of a proposition. Terms and contexts are defined as

$$
\begin{array}{llrl}
\textit{Terms} & M, N, A, B & ::= & x \mid \mathsf{Set} \\
 & & \mid & (x : A) \to B \mid \lambda x.\, M \mid MN \\
 & & \mid & (x : A) \times B \mid \langle M, N \rangle \mid \mathsf{fst}(M) \mid \mathsf{snd}(M) \\
 & & \mid & \mathsf{require}\ x : A\ \mathsf{in}\ M \\
\textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : A \\
\textit{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, x : A
\end{array}
$$

The new term $\mathsf{require}\ x : A\ \mathsf{in}\ M$ should be understood to mean roughly "find some $x : A$ in the context and make it available in M."

Lexical constants (e.g. Man, Own, etc.) are to be contained in a *signature* Σ, whereas the context Γ is reserved for local hypotheses. The use of signatures to carry the constants of a theory originates from the Edinburgh Logical Framework, where individual logics were represented as signatures of constants which represented their syntax, judgments and rule schemes (Harper, Honsell & Plotkin 1993). Then the basic forms of judgment are as follows:

$$
\begin{array}{rl}
\vdash\ \Sigma\ \mathrm{sig} & \Sigma \text{ is a valid signature} \\
\vdash_\Sigma \Gamma\ \mathrm{ctx} & \Gamma \text{ is a valid context} \\
\Gamma \vdash_\Sigma M : A & M \text{ has type } A
\end{array}
$$

In context validity judgments $\vdash_\Sigma \Gamma\ \mathrm{ctx}$, we presuppose $\vdash \Sigma\ \mathrm{sig}$; likewise, in typing judgments $\Gamma \vdash_\Sigma M : A$, we presuppose $\vdash_\Sigma \Gamma\ \mathrm{ctx}$. The rules for the signature and context validity judgments are as expected:

$$
\frac{}{\vdash \cdot\ \mathrm{sig}}
$$

$$
\frac{\vdash \Sigma\ \mathrm{ctx} \qquad \cdot \vdash_\Sigma A : \mathsf{Set} \qquad x \notin \Sigma}{\vdash \Sigma, x : A\ \mathrm{sig}}
$$

$$\frac{}{\vdash_\Sigma \cdot \ \mathrm{ctx}}$$

$$\frac{\vdash_\Sigma \Gamma \ \mathrm{ctx} \qquad \Gamma \vdash_\Sigma A : \mathsf{Set} \qquad x \notin \Gamma \cup \Sigma}{\vdash_\Sigma \Gamma, x : A \ \mathrm{ctx}}$$

Constants and hypotheses may be projected from signatures and contexts respectively:

$$\frac{}{\Gamma \vdash_{\Sigma, x:A, \Sigma'} x : A}\ \mathrm{const}$$

$$\frac{}{\Gamma, x : A, \Gamma' \vdash_\Sigma x : A}\ \mathrm{hyp}$$

The inference rules for the familiar terms are the usual ones:

$$\frac{}{\Gamma \vdash_\Sigma \mathsf{Set} : \mathsf{Set}}\ \mathsf{Set} : \mathsf{Set}$$

$$\frac{\Gamma \vdash_\Sigma A : \mathsf{Set} \qquad \Gamma, x : A \vdash_\Sigma B : \mathsf{Set}}{\Gamma \vdash_\Sigma (x : A) \to B : \mathsf{Set}} \to\mathrm{F}$$

$$\frac{\Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x.\, M : (x : A) \to B} \to\mathrm{I}$$

$$\frac{\Gamma \vdash_\Sigma M : (x : A) \to B \qquad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma MN : [N/x]B} \to\mathrm{E}$$

$$\frac{\Gamma \vdash_\Sigma A : \mathsf{Set} \qquad \Gamma, x : A \vdash_\Sigma B : \mathsf{Set}}{\Gamma \vdash_\Sigma (x : A) \times B : \mathsf{Set}} \times\mathrm{F}$$

$$\frac{\Gamma \vdash_\Sigma M : A \qquad \Gamma \vdash_\Sigma N : [M/x]B}{\Gamma \vdash_\Sigma \langle M, N \rangle : (x : A) \times B} \times\mathrm{I}$$

$$\frac{\Gamma \vdash_\Sigma P : (x : A) \times B}{\Gamma \vdash_\Sigma \mathsf{fst}(P) : A} \times\mathrm{E}_1$$

$$\frac{\Gamma \vdash_\Sigma P : (x : A) \times B}{\Gamma \vdash_\Sigma \mathsf{snd}(P) : [\mathsf{fst}(P)/x]B} \times\mathrm{E}_2$$

The only inference rule which is new deals with presuppositions:

$$\frac{\Gamma \vdash_\Sigma M : A \qquad \Gamma \vdash_\Sigma [M/x]N : B \qquad x \notin FV(B)}{\Gamma \vdash_\Sigma \mathsf{require}\ x : A\ \mathsf{in}\ N : B}\ \mathit{require}$$

A *require* term is essentially just a *let*, as found in many programming languages, with the assigned value provided by the type checker. This rule is a bit bizarre, of course, because the proof term M does not appear in the resulting proof term $\mathsf{require}\ x : A\ \mathsf{in}\ N$, but from a type-theoretic perspective this is acceptable. In realizability-based type theories such as Computational Type Theory (Allen, Bickford, Constable et al. 2005), information is routinely forgotten from the premises of inference rules; this allows the formulation of a polymorphic type theory containing very rich, extensional types including subsets and unions. On the other hand, this means that the terms of the theory may no longer be checked against their types, since the typing judgment is synthetic and not analytic (i.e. the evidence for its validity is not contained in the judgment itself). As such, it becomes necessary to consider terms as the *computational content* of a proposition's proof, which is justified by a derivation in the metalanguage.

It should also be noted that *require* will in general be a non-deterministic rule, unlike the other inference rules. This is because there could be many solutions for the presupposition.

We can now provide a true semantics for pronouns and definite determiners:

$$\begin{aligned}
[\![\text{he}]\!] &= \mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ x \\
[\![\text{it}]\!] &= \mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ x \\
[\![\text{the}]\!] &= \lambda P.\, \mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ (\mathsf{require}\ p : P\,x\ \mathsf{in}\ x)
\end{aligned}$$

Now let us reconsider our examples with the new semantics:

$[\![$A man walked in. He sat down.$]\!]$
$= (p : (x : \mathsf{E}) \times \mathit{Man}\,x \times \mathit{WalkedIn}\,x) \times \mathit{SatDown}(\mathsf{require}\ y : \mathsf{E}\ \mathsf{in}\ y)$

$[\![$A man walked in. The man (then) sat down.$]\!]$
$= (p : (x : \mathsf{E}) \times \mathit{Man}\,x \times \mathit{WalkedIn}\,x)$
$\quad \times \mathit{SatDown}\,(\mathsf{require}\ y : \mathsf{E}\ \mathsf{in}\ (\mathsf{require}\ q : \mathit{Man}\,y\ \mathsf{in}\ y))$

$[\![$If a farmer owns a donkey, he beats it.$]\!]$
$= (p : (x : \mathsf{E}) \times \mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Owns}\,x\,y)$
$\quad \to \mathit{Beats}\ (\mathsf{require}\ z : \mathsf{E}\ \mathsf{in}\ z)\ (\mathsf{require}\ w : \mathsf{E}\ \mathsf{in}\ w)$

$[\![$Every farmer who owns a donkey beats it.$]\!]$
$= (p : (x : \mathsf{E}) \times \mathit{Farmer}\,x \times (y : \mathsf{E}) \times \mathit{Donkey}\,y \times \mathit{Owns}\,x\,y)$
$\quad \to \mathit{Beats}\ p_1\ (\mathsf{require}\ w : \mathsf{E}\ \mathsf{in}\ w)$

All that remains is to define how to extract a final proof term without the use of require terms.

3.2 Extraction

To get the final propositions, we will define a meta-operation $\textsc{extr}(\mathcal{D})$ which transforms a derivation $\mathcal{D} :: \Gamma \vdash M : A$ into an extracted term M' which is

like M but without $\mathsf{require}$ terms, and instead with solutions in place of their bound variables. We define the operation inductively over the structure of the derivation as follows:

$$\textsc{extr}\left(\frac{}{\Gamma \vdash_\Sigma x : A}\,\text{const}\right) \rightsquigarrow x$$

$$\textsc{extr}\left(\frac{}{\Gamma \vdash_\Sigma x : A}\,\text{hyp}\right) \rightsquigarrow x$$

$$\textsc{extr}\left(\frac{}{\Gamma \vdash_\Sigma \mathsf{Set} : \mathsf{Set}}\,\mathsf{Set} : \mathsf{Set}\right) \rightsquigarrow \mathsf{Set}$$

$$\textsc{extr}\left(\frac{\mathcal{D} \qquad \mathcal{E}}{\Gamma \vdash_\Sigma A : \mathsf{Set} \qquad \Gamma, x : A \vdash_\Sigma B : \mathsf{Set}}{\Gamma \vdash_\Sigma (x : A) \to B : \mathsf{Set}}\,{\to}\mathrm{F}\right) \rightsquigarrow (x : \textsc{extr}(\mathcal{D})) \to \textsc{extr}(\mathcal{E})$$

$$\textsc{extr}\left(\frac{\mathcal{D}}{\Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x.\, B : (x : A) \to B}\,{\to}\mathrm{I}\right) \rightsquigarrow \lambda x.\, \textsc{extr}(\mathcal{D})$$

$$\textsc{extr}\left(\frac{\mathcal{D} \qquad \mathcal{E}}{\Gamma \vdash_\Sigma M : (x : A) \to B \qquad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma M\, N : [N/x]B}\,{\to}\mathrm{E}\right) \rightsquigarrow \textsc{extr}(\mathcal{D})\ \textsc{extr}(\mathcal{E})$$

$$\textsc{extr}\left(\frac{\mathcal{D} \qquad \mathcal{E}}{\Gamma \vdash_\Sigma A : \mathsf{Set} \qquad \Gamma, x : A \vdash_\Sigma B : \mathsf{Set}}{\Gamma \vdash_\Sigma (x : A) \times B : \mathsf{Set}}\,{\times}\mathrm{F}\right) \rightsquigarrow (x : \textsc{extr}(\mathcal{D})) \times \textsc{extr}(\mathcal{E})$$

$$\textsc{extr}\left(\frac{\mathcal{D} \qquad \mathcal{E}}{\Gamma \vdash_\Sigma M : A \qquad \Gamma \vdash_\Sigma N : [M/x]B}{\Gamma \vdash_\Sigma \langle M, N\rangle : (x : A) \times B}\,{\times}\mathrm{I}\right) \rightsquigarrow \langle \textsc{extr}(\mathcal{D}), \textsc{extr}(\mathcal{E})\rangle$$

$$\textsc{extr}\left(\frac{\mathcal{D}}{\Gamma \vdash_\Sigma P : (x : A) \times B}{\Gamma \vdash_\Sigma \mathsf{fst}(P) : A}\,{\times}\mathrm{E}_1\right) \rightsquigarrow \mathsf{fst}(\textsc{extr}(\mathcal{D}))$$

$$\textsc{extr}\left(\frac{\mathcal{D}}{\Gamma \vdash_\Sigma P : (x : A) \times B}{\Gamma \vdash_\Sigma \mathsf{snd}(P) : [\mathsf{fst}(P)/x]B}\,{\times}\mathrm{E}_2\right) \rightsquigarrow \mathsf{snd}(\textsc{extr}(\mathcal{D}))$$

$$\textsc{extr}\left(\frac{\mathcal{D} \qquad \mathcal{E}}{\Gamma \vdash_\Sigma M : A \qquad \Gamma \vdash_\Sigma [M/x]N : B}{\Gamma \vdash_\Sigma \mathsf{require}\ x : A\ \mathsf{in}\ M : B}\,\text{require}\right) \rightsquigarrow \textsc{extr}(\mathcal{E})$$

The most crucial rule is the last one — the preceding ones simply define extraction by induction on the structure of terms other than $\mathsf{require}$ terms. For a $\mathsf{require}$ term, however, we extract substituting the proof of the presupposed content for the variable in the body of the $\mathsf{require}$ term.

It is evident that the extraction process preserves type.

Theorem 1 *Given a derivation $\mathcal{D} :: \Gamma \vdash_\Sigma M : A$, there exists another derivation $\mathcal{D}' :: \Gamma \vdash_\Sigma \textsc{extr}(\mathcal{D}) : A$.*

Proof By straightforward induction on the structure of $\mathcal{D}$.

An example of extraction in action is necessary, so consider again the sentence "A man walked in. He sat down." Pre-extraction, the meaning of this will be:

$$(p : (x : \mathsf{E}) \times \mathit{Man}\, x \times \mathit{WalkedIn}\, x) \times \mathit{SatDown}(\mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ x)$$

Now let $\Sigma = \mathit{Man} : \mathsf{E} \to \mathsf{Set}, \mathit{WalkedIn} : \mathsf{E} \to \mathsf{Set}, \mathit{SatDown} : \mathsf{E} \to \mathsf{Set}$. After constructing a derivation that the above type is a Set under the signature Σ, we can extract the associated term. The left conjunct extracts to itself, so we will not look at that, but the extraction for the right conjunct is more interesting. The derivation for the right conjunct, letting $\Gamma = p : (x : \mathsf{E}) \times \mathit{Man}\, x \times \mathit{WalkedIn}\, x$, is:

$$\dfrac{\dfrac{}{\Gamma \vdash_\Sigma \mathit{SatDown} : \mathsf{E} \to \mathsf{Set}}\,\text{const} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\Sigma p : (x : \mathsf{E}) \times \mathit{Man}\, x \times \mathit{WalkedIn}\, x}\,\text{hyp}}{\Gamma \vdash_\Sigma \mathsf{fst}(p) : \mathsf{E}}\,\times\!\mathrm{E}_1 \qquad \mathcal{D} \\ \Gamma \vdash_\Sigma \mathsf{fst}(p) : \mathsf{E}}{\Gamma \vdash_\Sigma \mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ x : \mathsf{E}}\,\text{require}}{\Gamma \vdash_\Sigma \mathit{SatDown}(\mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ x) : \mathsf{Set}}\,{\to}\mathrm{E}$$

Inductively, we get:

$$\textsc{extr}\left(\dfrac{}{\Gamma \vdash_\Sigma \mathit{SatDown} : \mathsf{E} \to \mathsf{Set}}\,\text{const}\right) \rightsquigarrow \mathit{SatDown}$$

$$\textsc{extr}\left(\begin{array}{c}\mathcal{D} \\ \Gamma \vdash_\Sigma \mathsf{fst}(p) : \mathsf{E}\end{array}\right) \rightsquigarrow \mathsf{fst}(p)$$

For the $\mathsf{require}$ term's extraction, we substitute $\mathsf{fst}(p)$ in for x in x to get the following:

$$\textsc{extr}\left(\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\Sigma p : (x : \mathsf{E}) \times \mathit{Man}\, x \times \mathit{WalkedIn}\, x}\,\text{hyp}}{\Gamma \vdash_\Sigma \mathsf{fst}(p) : \mathsf{E}}\,\times\!\mathrm{E}_1 \qquad \mathcal{D} \\ \Gamma \vdash_\Sigma \mathsf{fst}(p) : \mathsf{E}}{\Gamma \vdash_\Sigma \mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ x : \mathsf{E}}\,\text{require}\right) \rightsquigarrow \mathsf{fst}(p)$$

And finally the extraction of whole subderivation yields $\mathit{SatDown}(\mathsf{fst}(p))$, and so the complete derivation yields

$$(p : (x : \mathsf{E}) \times \mathit{Man}\, x \times \mathit{WalkedIn}\, x) \times \mathit{SatDown}(\mathsf{fst}(p))$$

which is the meaning we had wanted.

A similar proof for "A man walked in. The man (then) sat down." can be given, with an extra non-trivial branch for $\mathit{Man}(\mathsf{fst}(p))$. Focusing just on the subproof for *the man*, we have the typing derivation

$$\dfrac{\begin{array}{c}\mathcal{E} \\ \Gamma \vdash_\Sigma \mathsf{fst}(p) : \mathsf{E}\end{array} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\Sigma p : (x : \mathsf{E}) \times \mathit{Man}\, x \times \mathit{WalkedIn}\, x}\,\text{hyp}}{\Gamma \vdash_\Sigma \mathsf{snd}(p) : \mathit{Man}(\mathsf{fst}(p)) \times \mathit{WalkedIn}(\mathsf{fst}(p))}\,\times\!\mathrm{E}_2}{\Gamma \vdash_\Sigma \mathsf{fst}(\mathsf{snd}(p)) : \mathit{Man}(\mathsf{fst}(p))}\,\times\!\mathrm{E}_1 \qquad \begin{array}{c}\mathcal{E} \\ \Gamma \vdash_\Sigma \mathsf{fst}(p) : \mathsf{E}\end{array}}{\Gamma \vdash_\Sigma \mathsf{require}\ q : \mathit{Man}(\mathsf{fst}(p))\ \mathsf{in}\ \mathsf{fst}(p) : \mathsf{E}}\,\text{require}}{\Gamma \vdash_\Sigma \mathsf{require}\ x : \mathsf{E}\ \mathsf{in}\ (\mathsf{require}\ q : \mathit{Man}\, x\ \mathsf{in}\ x)) : \mathsf{E}}\,\text{require}$$

This similarly extracts to $\mathsf{fst}(p)$ just as the subproof for *he* did before.

Extraction for "If a farmer owns a donkey, he beats it." and "Every farmer who owns a donkey beats it." unfolds in a similar fashion, with the extraction of the antecedent $(x : \mathsf{E}) \times \mathit{Farmer}\, x \times (y : \mathsf{E}) \times \mathit{Donkey}\, y \times \mathit{Owns}\, x\, y$ being trivial. The consequent *Beats* (require $z : \mathsf{E}$ in z) (require $w : \mathsf{E}$ in w) breaks down into three subproofs, one for the predicate *Beats* which extracts trivially, and the two require subproofs which extract like the previous pronominal examples. The only difference now being that the context has more options for the proofs.

Keen eyes will notice, however, that there should be four solutions, because both require terms require something of type E — the words *he* and *it* have no gender distinction in the semantics. This is left as an unspecified part of the framework, as there are a number of options for resolving gender constraints. Two options that are immediately obvious are 1) make E itself a primitive function $\mathsf{E} : \mathit{Gender} \to \mathsf{Set}$ and then specify a gender appropriately, or 2) add another require term so that, for example, $[\![\text{he}]\!] = \text{require } x : \mathsf{E} \text{ in } (\text{require } p : \mathit{Masc}\, x \text{ in } x)$ and provide appropriate axioms (possibly simply by deferring to other cognitive systems for judging gender). The former solution is akin to how certain versions of HPSG treat gender as a property of indices not of syntactic elements.

4 Discussion

In the previous sections, we have described an approach to pronominal and presuppositional pragmatics based on dependent types, as an alternative to DRT and Dynamic Semantics. The main difference from a standard dependently typed λ calculus is the addition of require terms, an extraction process to eliminate require terms. Like DRT, the dependent approach relies on an intensional, "syntactic" semantics, in the form of proof terms. For proponents of a more direct system, which can produce denotations without an intermediate intensional system, that sort of semantics is undesirable, but research exists into type theoretical approaches to metavariables using modal type theories which may be able to give a clean denotational interpretation to require terms, such as Nanevski, Pfenning & Pientka (2008).

A modal extension can also provide an interesting solution to another well-known problem in pragmatics. Consider the sentence "John will pull the rabbit out of the hat" when said of a scene that has 3 rabbits, 3 hats, but only a single rabbit in a hat. This sentence seems to be pragmatically acceptable and unambiguous, despite there being neither a unique rabbit nor a unique hat. In the framework given above, there should be 9 possible ways of resolving the presuppositions, leading to pragmatic ambiguity. A simple modality (approximately a possibility modality), however, can make sense of this: if the assertion of such a sentence presupposes that the sentence can be true via a modality (i.e. to assert P is to presuppose $\diamond P$), then there is only one way to solve the rabbit and hat presuppositions which will also make it possible to resolve the possibility presupposition — pick the rabbit that is in a hat,

and the hat that the rabbit is in — yielding a unique, unambiguous meaning. Whether this belongs in the semantics-pragmatics or in some higher system (such as a Gricean pragmatics) is debatable, but that such a simple solution is readily forthcoming at all speaks to the power of the above framework.

References

1. Allen, S.F., Bickford, M., Constable, R.L., Eaton, R., Kreitz, C., Lorigo, L., Moran, & E. (2005). Innovations in computational type theory using Nuprl. *Journal of Applied Logic*, 4(4).
2. Devriese, D., & Piessens, F. (2011). On the bright side of type classes: instance arguments in Agda. *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*.
3. Harper, R., Honsell, F., & Plotkin, G. (1993). A framework for defining logics. *Journal of the ACM*, 40(1).
4. Hurkens, A. (1995). A Simplification of Girard's Paradox. *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications*.
5. Marlow, S. (2010). Haskell 2010 Language Report.
6. Martin-Löf, P. (1984). *Intuitionistic Type Theory*. Bibliopolis, Napoli.
7. Nanevski, A., Pfenning, F., & Pientka, B. (2008). Contextual modal type theory. *Transactions on Computational Logic*, 9(3).