Coreference without Discourse Referents

A non-representational DRT-like discourse semantics

Gianluca Giorgolo, Christina Unger

UiL-OTS, Universiteit Utrecht

Abstract

We propose a non-representational, compositional treatment of anaphora. By non-representational we mean working directly with denotations without relying on specific features of a representational language, such as discourse referents. The approach is based on an adapted version of the montagovian theory of generalized quantifiers. Furthermore we use a language abstraction borrowed from programming language semantics to enforce a specific order of evaluation for the interpretations. This allows us to reproduce the sequential dynamics of anaphora. We present a complete formalization of our theory that can be easily implemented in a functional computational paradigm. In the end we discuss limitations of our approach and present preliminary solutions.

1 Introduction

This paper presents a compositional treatment of anaphora. Other similar proposals can be found in the literature: Groenendijk and Stokhof (1991) modify the interpretation of quantifiers in first-order logic by letting them have an unlimited scope, Muskens (1996) emulates DRT in type theory by adding sorts for discourse referents (thus maintaining a representational component), while de Groote (2006) employs a continuation-passing like approach where each sentence takes the continuation of the discourse as an argument, thus allowing a re-use of bound variables. The main goal of our work is to work directly with the denotations of linguistic constituents rather than relying on specific features of some representational language, as it is the case for more traditional approach to anaphora like DRT (Kamp and Reyle 1993) and file change semantics (Heim 1983).

In addition to this theoretical concern, we would also like to maintain some intuitive insights from DRT that are somehow obfuscated by the complexity of the formal machinery in proposals like the already cited de Groote (2006). And from a more applied perspective we would like to be able to go almost mechanically from a Montague-style lexicon to a "dynamic" lexicon.

Another fundamental issue that our proposal takes into account is the need to deal with time related phenomena. Anaphora, for example, enforces a particular sequence of linguistic events: first the antecedent noun phrase is introduced and then an anaphoric pronoun picks up the reference. A purely compositional account is however unable to capture these restrictions, but, as suggested by Shan (2002), we can simulate such capability through *monads*, a category-theory derived triple of mappings mainly used in programming language semantics to account for side-effecting computations (Moggi 1989, Wadler 1995).[1] This is similar to the ap-

[1] At the same time monads will allow us to express more concisely the complex mathematical objects

proach of Barker and Shan (2008) who employ continuation to enforce a specific order of evaluation.

The paper is structured as follows: in section 2 we present at an intuitive level the model and how we can treat the main aspects of anaphora in a purely compositional way. Section 3 elaborates on the ideas presented in section 2 by giving a formal implementation of the system. As the reader will notice, our formalization is strongly influenced by functional programming. Finally, in section 4 we illustrate how the system works by presenting a sample lexicon and working out some simple examples. We conclude with some problematic cases on which we are still working.

2 Compositional anaphora

2.1 Formal and notational preliminaries

As customary, we will assume that the denotations of natural language expressions can be identified with elements of set theoretic objects joined together in a set, referred to as a *frame*. We define a frame $\mathfrak{F}$ inductively as follows:[2]

1. $t \in \mathfrak{F}$, with $t = \{0, 1\}$.
2. $e \in \mathfrak{F}$, where e is a non-empty arbitrary set, the set of entities.
3. If $\sigma \in \mathfrak{F}$ and $\pi \in \mathfrak{F}$ then $(\sigma \to \pi) \in \mathfrak{F}$, where $(\sigma \to \pi)$ denotes the set of all functions with domain σ and codomain π

In the following, when we name elements of $\mathfrak{F}$, we will drop outermost parentheses and assume right associativity for functional abstraction ($\to$). We will also call these elements *types*, and state that an object has a certain type if it is a member of the set named by that type. When the exact type is not relevant we will use lower case Greek letters to indicate a type meta-variable.

Sometimes we will refer also to objects outside the frame but we will also refer to them using types. In particular we will mention an arbitrary singleton set and call it $Unit$.[3]

Often we will also need to state that a particular mathematical object is a member of a certain set (or type). To do this we will follow the type theoretical convention of writing $\ell : \tau$, meaning that the object ℓ is a member of the set (or has type) τ.

Finally, we will represent the mathematical objects of our frames by using λ-terms. The syntax we use is fairly standard: constants of various types (the exact type is normally left implicit but it is clear from the context) are typeset in boldface (e.g. $\mathbf{c}$), variables are typeset in italics (e.g. x), we write the application of a term t to a term u as $t\ u$, abstraction is represented as usual by a prefixed λ followed by

required by our approach.

[2] In our definition we do not distinguish types and domains.

[3] The nature of the single inhabitant of this type is completely irrelevant, as this is the value produced by computations which are interesting only for their side-effects.

the name of the variable that is abstracted in the rest of the term, we write $\langle t, u \rangle$ for the ordered pair formed by terms t and u, and $\pi_1\ p$ and $\pi_2\ p$ for the first and second projection of the pair p respectively.

2.2 Doing without discourse referents

In a theory like DRT, referents are purely linguistic[4] devices, that require an additional mechanism, an *assignment function*, to be interpretable. Our goal is to work directly with denotational objects and remove the need for a representation. Our solution is based on denotational objects corresponding to *generalized quantifiers*. The idea is to consider reference as a process that creates a function (the generalized quantifier) that given a particular set of conditions (equivalent to the conditions expressed in body part of boxes in DRT) produces a filter on the set of entities. In other words, noun phrases that introduce a reference contribute to discourse meaning by providing a promise of an entity-filter that can be fulfilled only when the predication connected to the noun phrase is complete.

Type theoretically this amounts to considering two kind of objects: on one hand *filters*, i.e. objects of type $(e \to t) \to t$, and on the other hand *conditions*, i.e. objects of type $e \to t$. The pair formed by a filter and a condition is the fundamental mathematical gadget of our approach, and we will call it a *reference pair*. It captures the idea that a discourse basically highlights specific regions of its domain of interpretation (in the simplest case sets of entities) and then incrementally and monotonically provides information regarding these areas of interest. The final interpretation of the discourse (type theoretically its truth value) is obtained by applying the filters to the corresponding conditions and conjoining the resulting truth values.

For example, the discourse "Someone shouts." can be roughly modeled by assuming that the noun phrase someone introduces a filter $\lambda P.\exists x.P\ x$, which is then coupled with then condition $\lambda x.\mathbf{shout}\, x$, contributed by the predicate shouts. We represent this situation as follows:

$$\boxed{\Phi = \lambda P.\exists x.P\ x \quad \Gamma = \lambda x.\mathbf{shout}\ x} \quad \longleftarrow \tag{1}$$

Now suppose that the discourse is continued with the sentence "She bleeds.": our account requires the condition to be updated with the new predication[5] introduced by the verb-phrase bleeds. The resulting pair can be represented as follows:

$$\boxed{\Phi = \lambda P.\exists x.P\ x \quad \Gamma = \lambda x.\mathbf{shout}\ x \wedge \mathbf{bleed}\ x} \quad \longleftarrow \tag{2}$$

The details concerning the compositional process that results in this pair are explained in section 2.4, but the idea behind it is that the new condition is merged, through intersection, with the already present condition.

[4]The language we are referring to is the box language or any formal substitute, not natural language.
[5]Of course assuming that the pronoun she refers back to the "someone" mentioned before.

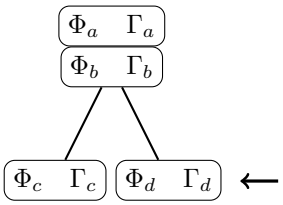

Figure 1: Sample reference tree

2.3 Coreferential availability

One of the strength of a theory like DRT is the ability to predict the possibility for discourse referents to be "picked up" by an anaphoric noun phrase. DRT encodes this information in terms of *accessibility*, a notion related to the structural properties of the terms used to represent a discourse. To maintain this predictive capability we also need to keep track of discourse structure.

We want to structure the introduced references so that *availability for coreference* and *saliency* are easily computable. We thus organize the reference pairs in a tree of stacks of such pairs. We will call such a tree a *reference tree*. Together with this structure we keep a pointer to the node we are currently operating on. The tree structure allows us to reproduce the DRT notion of accessibility in terms of *node dominance*: from the current position we have access to reference pairs that are either in the same node (somewhere in the stack) or in a node that *dominates* the current one. The information in any other node is not accessible. The use of a stack in the each node is a crude way to encode saliency: pairs higher in the stack are more recent and thus more salient for coreference resolution.[6]

For example, if the reference tree built so far is the one represented in figure 1 and the current position is signaled by the arrow, at this specific point we have access to the information encoded by the Φ_d, Γ_d pair, by the two pairs in the dominating node, Φ_a, Γ_a and Φ_b, Γ_b, but not to the pair Φ_c, Γ_c.

Let us consider a more concrete example. First we assume that indefinite noun phrases introduce a reference pair in the current node, while universally quantified noun phrases first create a new child node and then introduce a reference pair in the newly created position. Upon exiting the scope of the universal quantifier the pointer is moved up again to the topmost position. In this way the reference introduced by the universal quantifier will not be available for future coreference. Thus, if we are given the discourse "A droid entered. A woman arrived. Each man laughed.", we can represent the denotation we will get after processing the

[6]Our model is actually completely independent of any coreference resolution strategy, so it could be coupled with a more advanced algorithm for saliency computation. We are using this simple strategy just for illustrative purposes.

last sentence as follows:

$$\begin{array}{c} \boxed{\Phi = \lambda P.\exists x.P\ x \quad \Gamma = \lambda x.\textbf{woman}\ x \wedge \textbf{arrive}\ x} \quad \longleftarrow \\ \boxed{\Phi = \lambda P.\exists x.P\ x \quad \Gamma = \lambda x.\textbf{droid}\ x \wedge \textbf{enter}\ x} \\ \vert \\ \boxed{\Phi = \lambda P.\forall x.P\ x \quad \Gamma = \lambda x.\textbf{man}\ x \rightarrow \textbf{laugh}\ x} \end{array} \tag{3}$$

The reference pairs introduced by the two sentences "A droid entered." and "A woman arrived." are still available for further expansion, we could for example extend the condition on the "woman" filter by uttering something like "She sat down.". On the other hand, the reference pair introduced by "Everyone laughed." is inaccessible, disallowing any attempt to modify it. This is in line with the intuition that a discourse continuation like "He found her funny." should be ruled out.[7]

2.4 Building denotations

We turn now to the process of building the denotational structures introduced in the previous section. The strategy we present combines the principle of compositionality with a notion of information threading.

The main idea is that sentences are processed one at the time in a completely compositional fashion, and, similarly to a standard montagovian setting, they produce a value. However, while the value produced by a sentence is considered to be a truth value in standard Montague semantics, in our case a sentence will produce a value that corresponds to a *condition*, a function of type $e \rightarrow t$, roughly the denotation of the verb phrase. This condition is then integrated in the condition of the current most salient reference pair through set intersection.[8] The modified reference tree is then passed as a context for the subsequent sentence. The integration and the state passing is performed by an operator referred to as *merge*.

Noun phrases corresponding to generalized quantifiers contribute to discourse meaning by pushing new reference pairs on the stack of the current node, while definite noun phrases and pronouns alter the position of the pointer in the current reference tree.

The interpretation of the whole discourse is obtained by taking the conjunction of the application of each filter to its condition. For example, if the discourse we want to interpret results in a reference tree like the one in figure 1, then the interpretation of the discourse corresponds to the formula:

$$\Phi_a\ \Gamma_a \wedge \Phi_b\ \Gamma_b \wedge \Phi_c\ \Gamma_c \wedge \Phi_d\ \Gamma_d \tag{4}$$

This formula is equivalent to the predicate logic translation of the discourse's representation structure that DRT would build.

[7] Of course assuming that we can pick up the "droid" reference only with a neuter pronoun it.

[8] Objects of type $e \rightarrow t$ can in fact be regarded as sets, as they corresponds to the *characteristic function* of a set: for every object of type e they tell us if the element is in the set (returning 1) or not (returning 0).

3 Formalization

In this section we formally describe how to implement the ideas outlined in section 2. All our definitions have a functional programming flavour, and in fact they can be implemented quite literally in a functional language like Haskell or ML. We think that expressing the model directly in a well understood computational paradigm is a major advantage as it guarantees a correct implementation.

3.1 Adding a level of abstraction: monads

The main reason for introducing monads is the fact that they allow the simulation of the notion of sequencing without the need to leave the known ground of pure compositionality. Thus, we don't need to extend our models to account for an operator corresponding to the well known `;` of imperative programming languages (see for example (Muskens 1996)).

Actually monads can be used to simulate many more "side effects" than sequencing, and in fact we will use them also to simulate state changing computations. In the previous discussion we often referred to threading a modified reference tree from one sentence to another, and monads allow us to model this process in an elegant way.

A monad is defined as a triple $\langle M, \eta, \star \rangle$ where:

- M is a name for the set representing the monadic computations, most often some type of function. M is usually used in a parametrized fashion: $M\ \alpha$ indicates that we are looking at the specific sets of monadic computations that yield an object of type α.
- η (pronounced "unit") is a function from any set α to the set $M\ \alpha$, we can think of it as the function that lifts any value into a monadic computation.
- $\star$ (pronounced "bind") is a function of type $M\,\alpha \rightarrow (\alpha \rightarrow M\,\beta) \rightarrow M\,\beta$ (a sort of "monadic apply"), it extracts a value from a monadic computation, plugs it into function that uses the value to create a new monadic computation and returns the new monad.

The specific monad we are going to use is commonly referred to as the *State* monad. It can be used to simulate computations that produce a value by reading and possibly modifying an environment, and simultaneously to introduce a notion of *sequencing*. For the State monad, M will correspond to the a set of functions from the set of objects we decide to consider as environments to a pair composed of a value and a (possibly new) environment. Assuming that we call the set of environments env, we can concisely refer to set corresponding to this monad with the type $env \rightarrow \langle \alpha, env \rangle$, where α is the result type of the computations we are interested in[9]. In our case, of course, the environment will be a tree of reference pairs.

[9]So, strictly speaking, M is not a set but the set of all sets corresponding to the computations with a specified result type.

In the case of the State monad, η couples a value with an unmodified environment:

$$\begin{aligned} &\eta : a \rightarrow M\ a \\ &\eta\ a = \lambda s.\langle a, s\rangle \end{aligned} \tag{5}$$

The function $\star$ simulates the sequencing of operations:

$$\begin{aligned} &\star : M\ a \rightarrow (a \rightarrow M\ b) \rightarrow M\ b \\ &v \star k = \lambda s.k\ (\pi_1\ (v\ s))\ (\pi_2\ (v\ s)) \end{aligned} \tag{6}$$

In words, $\star$ passes the current computational environment to the first monadic computation and then uses the result of this computation plus the resulting environment to start the second computation.

For practical reasons it is useful to introduce a third function $\triangleright : Ma \rightarrow Mb \rightarrow M\ b$ defined on the base of $\star$ as follows: $k \triangleright v = k \star (\lambda x.v)$, where x must not occur free in the term v. This function will be used to thread operations that only affect the environment without producing any meaningful value.

3.2 Implementation

To implement the theoretical notions presented in section 2 we need to define the following functions:

- $\circ$ (pronounced "apply") is the monadic version of functional application. It represents the compositional step of standard Montague-semantics lifted at the level of stateful computations. It can be defined as follows:

  $$\begin{aligned} &\circ : (M\ (\alpha \rightarrow \beta)) \rightarrow (M\ \alpha) \rightarrow (M\ \beta) \\ &g \circ v = g \star (\lambda f.v \star (\lambda x.\eta\ (f\ x))) \end{aligned} \tag{7}$$

  The idea is the following: first the computation g is performed, and its value is named f, subsequently the computation v is carried out, within the context produced by the previous computation, and produces a value that we call x. Finally the result of applying the function f to its argument x is plugged into a computation that keeps the environment returned by the last computation.

- $\oplus$ (pronounced "merge") represents the merge operation introduced in section 2.4. Its definition is based on a couple of helper functions that are specific for the environment we assume for our State monad:

  - $addCondition : (e \rightarrow t) \rightarrow M\ Unit$, which intersects the passed condition with the condition in the currently most salient reference pair.
  - $backToTop : M\ Unit$, a function that moves the pointer from the current reference pair to the most salient pair in the root of the tree.

Assuming these two functions we can define $\oplus$ as follows:

$$\oplus : M\ (e \to t) \to M\ (e \to t) \to M\ (e \to t) \tag{8}$$
$$c \oplus k = c \star (\lambda v.addCondition\ v \triangleright (backToTop \triangleright k))$$

In words, $\oplus$ extracts the condition generated by the first sentence (remember that sentences have the type of predicates in our theory), adds it as a condition for the current most salient reference pair, moves the pointer to the topmost node and then goes on by calculating the computation that represents the contribution of the second sentence to the discourse.

Finally we introduce some useful functions that will be used in the example lexicon introduced in the following section:

- $addChild : M\ Unit$, which attaches to the tree a new node as a child of the current node and sets the newly created node as the current node.
- $createRef : ((e \to t) \to t) \to M\ Unit$, which creates a new reference pair in the current node, by using the filter passed as an argument and the constantly true function $\lambda x.1 : e \to t$ as a default condition. This function is the *identity element* of set-intersection on $\wp(e)$[10], so it can be safely used as a place holder until the real condition is computed.
- $moveToMostSalientRef : M\ Unit$, which moves the pointer to the most salient reference pair.
- $getMostSalientRef : M\ \langle((e \to t) \to t), (e \to t)\rangle$[11], a function that returns the currently most salient reference pair.

4 A toy lexicon and examples

The construction of a lexicon is a straightforward procedure: we start with a standard montagovian lexicon and lift to the monadic level all the lexical items that do not have any referential power by means of the η function. All the items that have some referential power (e.g. generalized quantifiers, pronouns, etc.) are instead paired with a denotation reflecting the principles illustrated above.

For example, in the case of an intransitive verb like run, whose denotation is a subset of the domain of entities that we can represent as **run** $: e \to t$, we can obtain the denotation for our dynamic semantics by applying η to **run**. We obtain an object of type $M\ (e \to t)$, representable as $\lambda e.\langle \mathbf{run}, e\rangle$.

In the case of generalized quantifiers, determiners and pronouns, we need a different approach. All these items play some role in anaphoric reference, and their dynamic denotation must reflect this. To better understand how we can formally capture the intuitions presented above let us work out the denotation of a quantified

[10] As stated above, we can identify a set with its characteristic function and of course $\lambda x.1$ represents the complete domain e. It is trivial to see that for every $A \subseteq e$ we have $A \cap e = e \cap A = A$.

[11] Here we abuse notation by lifting the syntax for pairs from terms to types.

WORD	TYPE/DENOTATION
someone	$M\ ((e \to t) \to e \to t)$
	$createRef(\lambda P.\exists x.P\ x) \triangleright \eta(\lambda p.\lambda x.p\ x)$
Yoda	$M\ ((e \to t) \to e \to t)$
	$createRef(\lambda P.P\ \mathbf{yoda}) \triangleright \eta(\lambda p.\lambda x.p\ x)$
some	$M\ ((e \to t) \to (e \to t) \to e \to t)$
	$createRef(\lambda P.\exists x.P\ x) \triangleright \eta(\lambda p.\lambda q.\lambda x.p\ x \wedge q\ x)$
every	$M\ ((e \to t) \to (e \to t) \to e \to t)$
	$addChild \triangleright createRef(\lambda P.\forall x.P\ x) \triangleright \eta(\lambda p.\lambda q.\lambda x.p\ x \to q\ x)$

Table 1: Generalized quantifiers and determiners

noun phrase like "everyone". First of all we have to decide the type we want to assign to it. We can start from the classical montagovian type $(e \to t) \to t$ and adapt it to the dynamic context. The new denotation should still be an object that takes a predicate of type $e \to t$ as an argument but instead of returning a truth value, the type of sentence denotations in Montague semantics, it should return a condition as described above. The denotation must be wrapped in a monadic layer, so the type of the denotation of a generalized quantifier should be something of type $M((e \to t) \to (e \to t))$. The specific mathematical object that represents the denotation of "everyone" should have the following effects on the environment:

- it should add a new level in the reference tree,
- create a new filter-condition pair whose filter is a function that checks that the corresponding condition is universally satisfied,

and then produce as a result the condition generated by the predicate of the sentence. With the tools presented so far we can formally represent such an object as follows:

$$addChild \triangleright createRef(\lambda P.\forall x.P\ x) \triangleright \eta(\lambda p.\lambda x.p\ x). \tag{9}$$

We give some more examples of generalized quantifier and determiner denotations in table 1.

In the case of the non referential use of quantified noun phrases, as it is the case for *de dicto* readings, we can simply keep the montagovian denotation and lift it to the monadic level by the use of η. The effect we obtain is basically to *trap* the quantifier inside a condition and make it inaccessible for further modification.

LEXICAL ENTRY	TYPE	DENOTATION
someone	$M\ ((e \to t) \to t)$	$\eta(\lambda P.\exists x.P\ x)$
every	$M\ ((e \to t) \to (e \to t) \to t)$	$\eta(\lambda P.\lambda Q.\forall x.P\ x \to Q\ x)$

The denotation of anaphoric pronouns in our dynamic semantics is a function that has the only effect of moving the pointer to the currently most salient reference

pair and then let the condition defined by the predicate it scopes over to be added to this reference pair. For example, the denotation of a pronoun like she will be a function of type $M\ ((e \to t) \to e \to t)$ that we can represent as follows:

$$moveToMostSalientRef \triangleright \eta(\lambda p.\lambda x.p\ x) \tag{10}$$

As it was the case for quantified noun phrases, pronouns do not produce any "usable" value but rather direct the values produced by the predicates to the correct reference pair.

We conclude this section illustrating the implementation of our system with a simple example. Suppose we want to analyze the discourse "Yoda woke-up. He was tied-up.". We will simplify the treatment by ignoring issues like tense and aspect and consider the past participle tied an adjective (together with the particle up). The lexical entries for the words of this discourse can be derived from the discussion above, with the exception of the verb to be, which we will consider to be the identity function on objects of type $e \to t$ (of course lifted to the monadic level). The computation of the meaning of the discourse starts in the context of a reference tree with a single node, containing an empty stack of reference pairs. The computation of the denotation of the first sentence equals to the evaluation of the following term:

$$(createRef(\lambda P.P\ \textbf{yoda}) \triangleright \eta(\lambda p.\lambda x.p\ x)) \circ \eta(\textbf{wake-up}) : M\ (e \to t) \tag{11}$$

Evaluating this term produces the value **wake-up** but also modifies the environment producing the following reference tree:

$$\boxed{\Phi = \lambda P.P\ \textbf{yoda} \quad \Gamma = \lambda x.1} \quad \longleftarrow \tag{12}$$

The result of the computation is then merged with the result of computing the denotation of the second sentence:

$$(\textbf{wake-up}) \oplus ((moveToMostSalientRef \triangleright \eta(\lambda p.\lambda x.p\ x)) \circ (\eta(\lambda p.\lambda x.p\ x) \circ \eta(\textbf{tied-up}))) \tag{13}$$

The $\oplus$ operation will first intersect the condition **wake-up** with the reference pair currently pointed at, producing the environment:

$$\boxed{\Phi = \lambda P.P\ \textbf{yoda} \quad \Gamma = \lambda x.1 \wedge \textbf{wake-up}\ x} \quad \longleftarrow \tag{14}$$

This is the environment used to compute the denotation of the term to the right of $\oplus$ in 13. Evaluating the denotation of the pronoun he moves the pointer to the most salient referent (we have only one referent, so the environment is left unchanged), so the evaluation of term 13 ends with the environment represented in 14 and the value **tied-up**. To finish he computation of the discourse semantics we can merge the discourse with the empty condition $\eta(\lambda x.1) : M\ (e \to t)$, similarly to what we

would do in a continuation passing regime by passing the identity function at the end. The resulting reference tree is the following:

$$\boxed{\Phi = \lambda P.P\ \textbf{yoda} \quad \Gamma = \lambda x.1 \wedge \textbf{wake-up}\ x \wedge \textbf{tied-up}\ x} \quad \longleftarrow \tag{15}$$

which can be flattened into the formula:

$$\textbf{wake-up yoda} \wedge \textbf{tied-up yoda} \tag{16}$$

4.1 Open issues

We have not given a treatment of pronouns in object position yet. If we keep the simplistic approach presented in the previous section, we can assume that pronoun in object position retrieves from the environment the reference pair it refers to, and then fills the object position of the transitive predicate with it. The denotation of such a pronoun is an object of type $M\ ((e \to e \to t) \to e \to t)$ and we can represent it with the following term:

$$getMostSalientRef \star (\lambda\langle t, c\rangle.\eta(\lambda r.\lambda y.t(c \cap (\lambda x.r(x)(y))))) \tag{17}$$

To exemplify the resulting semantics consider the simple discourse "A jedi entered. Everyone watched her.". We skip all the details of the computation and present the final environment:

$$\begin{array}{c} \boxed{\Phi = \lambda P.\exists x.P\ x \quad \Gamma = \lambda x.\textbf{jedi}\ x \wedge \textbf{enter}\ x} \quad \longleftarrow \\ | \\ \boxed{\Phi = \lambda P.\forall x.P\ x \quad \Gamma = \lambda x.\exists y.\textbf{jedi}\ y \wedge \textbf{enter}\ y \wedge \textbf{watch}\ y\ x} \end{array} \tag{18}$$

However, this approach is problematic. The first problem shows up in the case of a discourse like "A jedi entered. Someone helped him. He was wounded.": in this case our treatment would produce a condition for the reference pair introduced by someone that enforces only the fact that some entity helped an entity that is a jedi and not a wounded jedi.

The second problem is that the formula that we get once we flatten the reference tree does not describe the model we intuitively associate with the discourse. In fact the formula we obtain describe a situation roughly equivalent to the discourse "A jedi entered. Everyone watched a jedi.". We do not encounter the same problem if the pronoun is under the scope of an existential quantifier, as the following holds

$$(\exists x.P\ x \wedge (\exists y.R\ x\ y)) \leftrightarrow (\exists x.P\ x \wedge (\exists y.\exists z.P\ z \wedge R\ z\ y))) \tag{19}$$

This observation suggests that we could change the denotation for universal quantifiers. Instead of interpreting as reference introducers, we can keep the classical montagovian interpretation for them. Note that this does not change their accessibility, because even if we interpret them as introducing a referent in the discourse,

this referent cannot be picked up by subsequent pronouns. For example, we can associate everyone with the following denotation:

$$\eta(\lambda P.\forall x.P\ x) : M\ ((e \to t) \to t) \tag{20}$$

However, composing this interpretation with the one of watched her would result in an object of type t, while we assume sentences to be of type $e \to t$. To solve this problem and to get the correct interpretation of this sentence, we need to change the denotation for the anaphoric pronoun as well: the pronoun should take as arguments the transitive verb and the generalized quantifier and generate a condition that can then be intersected with the one of the reference pair it points to. More formally, we can assign to her the following denotation:

$$\begin{aligned} &moveToMostSalientRef \triangleright \eta(\lambda P.\lambda Q.\lambda x.Q\ (P\ x)) : \\ &\quad M\ ((e \to e \to t) \to ((e \to t) \to t) \to e \to t) \end{aligned} \tag{21}$$

With this semantics the discourse "A jedi entered. Everyone watched her. She bled." produces the following reference tree:

$$\boxed{\Phi = \lambda P.\exists x.P\ x \quad \Gamma = \lambda x.\textbf{jedi}\ x \wedge \textbf{enter}\ x \wedge \forall y.\textbf{watch}\ x\ y \wedge \textbf{bled}\ x} \tag{22}$$

This reference tree in turn produces the desired truth conditions. Nevertheless it forces us to make an unnatural distinction between the denotation of pronouns in object position coupled with universal quantifiers and those under the scope of existential quantifiers. In addition we need to postulate a phonologically silent operator that lifts the type of a sentence such as Everyone smiled to the type of conditions and adds it to the currently most salient reference pair.

The solutions presented here are only partial and require further research.

5 Conclusion

We presented a completely compositional model for discourse semantics capable of capturing the main aspects of anaphoric coreference. Our approach extends the montagovian approach to phenomena that are traditionally accounted for with *ad hoc* mechanisms. We grounded our approach on the notion of generalized quantifiers showing how the decoupling of the quantificational component (the *filter* in our terms) and the restrictive component (the *condition*) of their usual denotation can be used to represent updatable reference. To capture the sequential nature of discourse we borrowed the notion of *monads* from programming language semantics. This allowed us to enforce a specific order of evaluation needed to capture the *dynamics* between *introduction* and *backward-reference* in discourse semantics. At the same time, monads allowed for a neat emulation of evolving state, a notion that intuitively captures the ongoing changes in the creation of a discourse model.

Finally, we pointed out some limitations of our approach and proposed partial solutions. At the moment we are working on alternative approaches to pronoun semantics.

References

Barker, Chris and Chung-chieh Shan (2008), Donkey anaphora is in-scope binding, *Semantics and Pragmatics* **1** (1), pp. 1–46.

de Groote, Philippe (2006), Towards a montagovian account of dynamics, *Proceedings of Semantics and Linguistic Theory XVI*.

Groenendijk, Jeroen and Martin Stokhof (1991), Dynamic predicate logic, *Linguistics and Philosophy* **14** (1), pp. 39–100.

Heim, Irene (1983), File change semantics and the familiarity theory of definiteness, *in* Bäuerle, Rainer, Christoph Schwarze, and Arnim von Stechow, editors, *Meaning, use, and interpretation of language*, de Gruyter.

Kamp, Hans and Uwe Reyle (1993), *From Discourse to Logic*, Kluwer, Dordrecht.

Moggi, E. (1989), Computational lambda-calculus and monads, *Proceedings of the Fourth Annual Symposium on Logic in computer science*, IEEE Press, Piscataway, NJ, USA, pp. 14–23.

Muskens, Reinhard (1996), Combining Montague Semantics and Discourse Representation, *Linguistics and Philosophy* **19**, pp. 143–186.

Shan, Chung-Chieh (2002), Monads for natural language semantics, *CoRR*.

Wadler, Philip (1995), Monads for functional programming, *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques-Tutorial Text*, Springer-Verlag, London, UK, pp. 24–52.