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

A Computational Treatment of Anaphora and its Algorithmic Implementation

Jean-Philippe Bernardy · Stergios Chatzikyriakidis · Aleksandre Maskharashvili

Abstract In this paper, we propose a framework capable of dealing with anaphora and ellipsis which is both general and algorithmic. This generality is ensured by the compination of two general ideas. First, we use a dynamic semantics which reperent effects using a monad structure. Second we treat scopes flexibly, extending them as needed.

We additionally implement this framework as an algorithm which translates abstract syntax to logical formulas. We argue that this framework can provide a unified account of a large number of anaphoric phenomena. Specifically, we show its effectiveness in dealing with pronominal and VP-anaphora, reflexives, strict and lazy pronouns, lazy identity, bound variable anaphora, e-type pronouns, and cataphora. This means that in particular we can handle complex cases like Bach-Peters sentences, which require an account dealing simultaneously with several phenomena.

We use Haskell as a meta-language to present the theory, which also consitutes an implementation of all the phenomena discussed in the paper. To demonstrate coverage, we propose a test suite that can be used to evaluate computational approaches to anaphora.

1 Introduction

Anaphora is a vast topic that has been exploited and discussed from the point of view of different formal semantics systems, each focusing on different phenomena [Coo79; Eva80; Rob89; Rob90; GS91; PZ92]. A question which arises is to what extent these different aspect of anaphora can be considered as instances of a single phenomena. Furthermore, if they turn out to be related, can they be treated under a single unified appoach? Our overarching aim is to

CLASP, Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg
E-mail: firstname.lastname@gu.se

construct a system for a unified computational treatment of anaphora of all kinds.

In this paper, we propose an account of anaphora which captures simultaneously (at least) all of the following phenomena:

- Basic anaphoric cases
- Category-general anaphora (VP anaphora, sentential anaphora, etc.)
- Strict and sloppy anaphora
- Bound variable readings
- E-type pronouns
- Donkey anaphora
- Cataphora
- Bach-Peters sentences

In particular, we find that our treatment of E-type pronouns and Bach-Peters sentences is a novel one, and as such can be considered a contribution to formal semantics in their own right. Acknowledging this fact, we concentrate the discussion in this paper on the elements which build up to our accounts for these phenomena, and we direct the reader to the long version of this paper [BCM20] for an even broader discussion of supported phenomena. As a witness to the completeness of our treatment, we provide a faithful formalization in Haskell [Mar10]. This formalization doubles up as an implementation. To demonstrate coverage of our theory, we have additionally tested on a suite of 65 representative examples presented in the appendix.

The structure of the paper is as follows: in Section 2, we give the background theory to this paper and briefly introduce monads. In Section 3, we provide a unified monadic account for the range of phenomena listed above. In Section 4, we present the Haskell implementation of the system and evaluate it in Section 5. In Section 6, we discuss related work. Lastly, in Section 7, we conclude and propose directions for future research.

2 Background

2.1 Higher-order logic with sorts and Σ types

In this paper we interpret natural language phrases (with anaphoric elements) as logical formulas. Most of the particular details of the logic that we use are of limited importance to the dynamic semantics that we propose. Unless explicitly stated otherwise, our analysis transposes easily to any higher-order logic. Yet we find it convenient to use a few specific surface features from constructive type-theories, sometimes called Modern Type Theories (MTTs) in the literature.

First, we will specify the domain of quantification for all quantifiers. More specifically, we will write $\forall(x : Man).Mortal(x)$ instead of $\forall x.Man(x) \to Mortal(x)$ and thus Man should be understood as a type (not a predicate). In effect, we use type many-sortedness offered by MTTs and we take the CNs-as-Types view, notably used by Ranta [Ran94] and later by Boldini [Bol00].

There is substantial literature on whether CNs should be interpreted as types or predicates. Within the MTT tradition, most of the approaches have been using the CNs-as-Types. Later work by Luo [Luo12], Chatzikyriakidis [Cha14], and Chatzikyriakidis and Luo [CL17] have explicitly argued that the CNs-as-Types approach is to be preferred over the predicates one. The debate is still open on which approach is superior, but in this paper our choice of the CN-as-types approach is justified by its better fit with scope extension (see Section 3.4.4).

Second, if such domains need to embed predicates, we use Σ-types to embed them.

Third, as proposed by Chatzikyriakidis and Luo [CL13], we make use of the following subtyping rule:

$$\Sigma(x : A).B \subseteq A$$

This rule is not critical, but allows to conveniently omit unessential projections. This allows us to conveniently write $\forall x : (\Sigma(y : Man).Wise(y)).Old(x)$ instead of $\forall x : (\Sigma(y : Man).Wise(y)).Old(\pi_1(x))$.

Finally, we will use the record notation of Σ types. This is purely a convenience, because records are isomorphic to nested Σ types. Thus, for the above formula we can write $\forall x : ([y : Man; p : Wise(y)]).Old(x.y)$. Note in particular the use of $.y$ for projection of the field y.

2.2 Monads

Originating in Category Theory [BW99], monads have been shown to be very useful for the semantics of programming languages and have been particularly widely used in functional programming languages. Monads have made it into the formal semantics of natural languages, succeeding in providing solutions to problems where the notions of context and context dynamics play an important role. Monadic treatments have been proposed for quantifiers [BS04], anaphora [Ung11], conversational implicatures [GA12], interrogatives [Sha02] and linguistic opacity [GA14], among others.

Formally, a monad is a triple $(M, return, \star)$ where

- M is a type-function: it maps any type a to a type Ma. For our purposes, Ma can be understood as the type of *effectful* computations that yield a value of type a.
- *return* is a function of type $a \to Ma$ which turns a pure (effect-free) value to a computation without effect.
- The operator $\star$, pronounced "bind" is a function of type $Ma \to (a \to Mb) \to Mb$ which composes two computations. That is, given an effectful computation of type Ma and a function mapping a to an effectful computation of type Mb, it returns a computation which in turn returns the result of the second computation. The effect of the composition is the composition of effects.

Additionally, for the triple to be a monad, the bind operator must be associative, and *return* must be its right and left unit. Formally:

- $(m \star f) \star g = m \star (\lambda x \to f\, x \star g)$
- $(return\ x) \star f = f\, x$
- $f \star return = f$

2.3 Do-notation

An expression such as $M \star \lambda x.(N \star \lambda y.O)$ is commonly used to bind the result of M as x in the expressions N and O; and y as the result of N in the expression O. An alternative notation for the same term, called the do-notation and originating from the Haskell language, is the following:

$$\begin{array}{l}\textbf{do}\ x \leftarrow M \\ \quad\ y \leftarrow N \\ \quad\ O\end{array}$$

The do-notation has an advantage of emphasizing the order of composition of effects (top down), and that of omitting the groupings of sub-expressions — which does not matter according to the associativity law of monads.

3 Generalized effectful semantics for anaphora: how to account for various phenomena

We place ourselves in the tradition of Montague semantics, in the sense that we interpret syntax trees as logical formulas. Nevertheless, we depart from that tradition in two respects. First, the logical system that we target is not a plain higher-order logic, but one equipped with dependent types. Second, each constituent is not just interpreted as a logical formula, but as a pair of a formula and an effect, which captures the dynamic aspects of meaning. In fact, following Shan [Sha02], the effect is coupled with the Montagovian[1] interpretation using a monad structure. More precisely, as Unger [Ung11] does, we assert that the relevant effects include:

- a lookup in an environment
- the production of a new environment

This kind of effects can be realized by using the standard *side-effects* monad of Moggi [Mog91], also called more accurately the *state* monad. We recall its type-function (M above) below using Haskell notation:

$$\textbf{type}\ Dynamic\ a = Env \to (a \times Env)$$

[1] Or indeed any interpretation which uses a logical system, such as dependent types, TTR, System F, etc.

Thus, if an effect-free semantic interpretation is a, its effectful counterpart is a function from an assumed environment *Env* (or background, context) to a pair of the semantic content a and a new environment. As is standard, looking up the environment is done using the *get* function, while updating it is done using the *set* function.

$$
\begin{aligned}
&\mathit{get} :: \mathit{Dynamic}\ \mathit{Env} \\
&\mathit{get} = \lambda\rho \rightarrow (\rho, \rho) \\
&\mathit{set} :: \mathit{Env} \rightarrow \mathit{Dynamic}\ 1 \\
&\mathit{set}\ \rho = \lambda\rho_0 \rightarrow (\rho, ())
\end{aligned}
$$

In *set*, the old environment ρ_0 is ignored. Additionally *set* returns no useful value (only its dynamic effect matters), and thus we return a unit type 1 which contains no useful information.

We take the type associated with syntactic categories to be the usual semantic interpretations assumed in Montague-style semantics. For example, the type associated with a verb phrase is $\mathit{object} \rightarrow \mathit{Prop}$: a function from objects (individuals in Montague's system) to propositions (truth values for Montague). For example, the semantic category corresponding to VP is:

$$\mathit{Dynamic}\ (\mathit{Semantics}\ (\mathit{VP}))$$

or, after expansion

$$
\begin{aligned}
&\mathit{Dynamic}\ (\mathit{Semantics}\ (\mathit{VP})) \\
&= \mathit{Dynamic}\ (\mathit{object} \rightarrow \mathit{Prop}) \\
&= \mathit{Env} \rightarrow ((\mathit{object} \rightarrow \mathit{Prop}) \times \mathit{Env})
\end{aligned}
$$

The monadic structure associated with *Dynamic*s has several pleasant properties. The existence of the *return* function ensures that any semantics cast in a Montagovian-style framework can be also embedded in our dynamic semantics (simply by associating it with the empty effect):

$$\mathit{return}\ x = \lambda\rho.\ (x, \rho)$$

Furthermore, the bind operator ensures that any two effectful interpretations can be combined arbitrarily. Consequently any combination interpretations in Montague Semantics (MS) can be lifted to effectful MS.

$$m \star f = \lambda\rho \rightarrow \mathbf{let}\ (\rho', x) = m\ \rho\ \mathbf{in}\ f\ x\ \rho'$$

Monadic associativity ensures that we do not have to worry about the order of grouping effects.

3.1 Structure of the environment and basic anaphoric cases

The environment (*Env*) is a map[2] from atomic anaphoric expressions (such as pronouns) to semantic representations. For now, we restrict the domain of the environment to NP anaphora:

$$\textbf{type}\ \mathit{Env} = \mathit{NP} \to \mathit{Dynamic}\ ((\mathit{object} \to \mathit{Prop}) \to \mathit{Prop})$$

For example, if the discourse has introduced a male person whose denotation is $[\![\mathit{Bill}]\!]$, the corresponding environment ρ can look like this:

$$\begin{aligned}
&\rho : \mathit{Env}\\
&\rho\ \texttt{"he"} = [\![\mathit{Bill}]\!]\\
&\rho\ \texttt{"him"} = [\![\mathit{Bill}]\!]\\
&\rho\ \texttt{"his"} = [\![\mathit{Bill's}]\!]\\
&\ldots
\end{aligned}$$

Thus in most cases, the effect of a non-anaphoric noun phrase is to modify the environment, so that appropriate anaphoric expressions can be mapped to it. We say that we *push* the interpretation onto the environment. This is done by creating an environment which checks for a pronoun that can refer to the expression and returns the given interpretation exactly in that case, as follows:

$$\begin{aligned}
&\mathit{pushNP} :: \mathit{NP} \to \mathit{Dynamic}\ ((\mathit{object} \to \mathit{Prop}) \to \mathit{Prop}) \to \mathit{Env} \to \mathit{Env}\\
&\mathit{pushNP}\ \mathit{pron}\ \mathit{referent}\ \rho\ x = \textbf{if}\ x \equiv \mathit{pron}\ \textbf{then}\ \mathit{referent}\ \textbf{else}\ \rho\ x
\end{aligned}$$

Using *pushNP*, we can implement (non-anaphoric) NPs easily. For example, the semantics of the proper name "Mary" is as follows, where *mary* is understood to be the underlying representation of the individual:

$$\begin{aligned}
&[\![\ \mathit{Mary}]\!] = \textbf{do}\\
&\quad \rho \leftarrow \mathit{get}\\
&\quad \mathit{set}\ (\mathit{pushNP}\ \texttt{"she"}\ (\mathit{return}\ (\lambda P \to P\ \mathit{mary}))\ \rho)\\
&\quad \ldots\\
&\quad \mathit{return}\ (\lambda P \to P\ \mathit{mary})
\end{aligned}$$

(In the full implementation we update all pronouns which can (co-)refer to "Mary", such as "her", "hers", "herself", etc.) We proceed with showing how anaphoric expressions can be interpreted. For instance, the pronoun "her" is implemented as follows:

$$\begin{aligned}
&[\![\ \mathit{her}]\!] = \textbf{do}\\
&\quad \rho \leftarrow \mathit{get}\\
&\quad \mathit{np} \leftarrow \rho\ \texttt{"her"}\\
&\quad \mathit{np}
\end{aligned}$$

Consider now the following full example:

[2] This function may be partial if the discourse is incomplete or incoherent.

(1) John loves Mary. Bill likes her.

In (1), upon encountering the phrase "Mary", our system will modify the environment such that further lookup for "her" will return $(\lambda P \rightarrow P\ mary)$. More pedantically, any effect combined with the effect of $[\![Mary]\!]$ will be affected in this manner. While straightforward, the mechanism described above allows to account for a large number of anaphoric phenomena (sometimes requiring local extension). The rest of this section describes such account.

3.2 Refinement: Category-general anaphora

In the examples seen so far, anaphoric expressions have been referring to noun-phrases exclusively. Thus, the domain of the environment could be only NP anaphora. Yet, English features anaphoric expressions for many syntactic categories, including at least adjectival phrases (AP), common nouns (CN), two-place common-nouns (CN2), verb phrases (VP) and transitive verb phrases (VP2). [3]

In the table below, we provide a number of examples of anaphora for various syntactic categories. We put the anaphora in square brackets and the referent in curly braces.

(2) John owns an {old} car. He likes [such] cars. (AP)
(3) Mary owns an old {car}. John owns a red [one]. (CN)
(4) The population of France is greater than [that] of Germany. (CN2)
(5) Mary fell. John did too. (VP)
(6) Mary has {read} all the books that John [has]. (VP2)
(7) {John may arrive this evening}. If [so] I am very happy. (S)

From these examples, we see that the syntactic form for the anaphoric expression determines the category of the referent. Thus, it is easy to choose the part of the environment to look for that referent. We can account for this phenomenon with a two-fold move:

– extending the environment so that there is a map for every possible syntactic category. Ideally this would be done as follows:

$$\textbf{type}\ Env = (Cat : Category) \rightarrow (phrase : Cat) \rightarrow Dynamic\ (Semantics\ Cat)$$

But since such dependent types are not supported by Haskell, we instead identify all categorical phrases for the second argument, using dynamic checks to tell them apart:

$$\textbf{type}\ Env = (Cat : Category) \rightarrow Phrase \rightarrow Dynamic\ (Semantics\ Cat)$$

[3] The semantic interpretations of these categories are given in Section 4.1. In traditional terms, CN and CN2 anaphora correspond to NP and complex NP anaphora respectively. Similarly, VP2 anaphora is known as Antecedent Contained Deletion.

– updating the environment for the appropriate category when encountering the referent and looking up the appropriate category when encountering the anaphora.

Using this technique we can account for cases such as the following:

(8) Mary fell. John did too.

The interpretation of the relevant constituents are as follows. When interpreting "fell", we push it in the environment for VP's, and when interpreting "did too", we look up this information.

$$
\begin{aligned}
&[\![\, \mathit{fell} \,]\!] = \mathbf{do} \\
&\quad \mathit{modify}\ (\mathit{pushVP}\ (\mathit{return}\ \mathit{fell})) \\
&\quad \mathit{return}\ \mathit{fell} \\
&[\![\, \mathbf{do}\ \mathit{too} \,]\!] = \mathbf{do} \\
&\quad \rho \leftarrow \mathit{get} \\
&\quad \mathit{vp} \leftarrow \rho\ \mathit{VP} \\
&\quad \mathit{vp}
\end{aligned}
$$

As an aside, one should note that, sometimes, the interpretation pushed in the environment should not be exactly the same as its pure semantics. We see an instance of this phenomenon in (7), where the sentence to put in the environment is that *without* the modal operator: "John arrives this evening".

3.3 Strict and sloppy anaphora

Sometimes the referent of an anaphoric expression contains an anaphoric expression itself, as in the following example:

(9) John loves his wife. Bill does too.

The following two readings of (9) are possible:

1. love(wifeof(john),john) × love(wifeof(john),bill)
2. love(wifeof(john),john) × love(wifeof(bill),bill)

We explain these interpretations as follows. In the so called strict reading 1., the referent for "so does" is taken to be the interpretation of "his wife" in its context of occurrence (i.e. John's wife), while in the second one, the reference is taken to be the effectful, dynamic semantics of "his wife" (i.e. the meaning should be re-interpreted in the environment at the point of occurrence of "so does").

Our framework is equipped to account for either reading. The first reading can be explained with the following interpretation for "loves":

$$
\begin{aligned}
&[\![\, \mathit{love} \,]\!]\ \mathit{object} = \mathbf{do} \\
&\quad o' \leftarrow \mathit{object}\ \mathit{Other} \\
&\quad \mathbf{let}\ \mathit{vp} = (\lambda x \rightarrow \mathit{love}\ (x, o'))
\end{aligned}
$$

modify (*push* (*VP*, `"do too"`) (*return vp*))
return vp

The above says that one begins by evaluating the direct object of the verb to a pure interpretation, o', and push the semantic verb phrase $\lambda x \to love\ (x, o')$ onto the environment, without any effect, by using *return*. We can also explain the second reading, by adapting only the interpretation of "love", as follows:

$[\![\ love\ ]\!]\ object =$ **do**
modify (*push* (*VP*, `"do too"`) ($[\![love]\!]$ *object*))
$o' \leftarrow$ *object Other*
return ($\lambda x \to love\ (x, o')$)

First, the effect of "loves his wife" is to push $[\![loves]\!]\ [\![his\ wife]\!]$ onto the environment. Then, upon resolution of "so does", one will perform the following effects:

- fetch ([[loves]] [[his wife]]) from the environment
- then evaluate ([[loves]] [[his wife]]) in the environment ρ, where in particular $\rho(NP, \texttt{"}his\texttt{"}) = BILL$
- obtain $\lambda x \to love\ (x, wifeof\ ([\![his]\!]\ \rho))$
- and finally evaluate the above to $\lambda x \to love\ (x, wifeof\ (bill))$

The careful reader may worry that, because the interpretation $[\![love]\!]\ object$ contains a reference to itself, its evaluation may not terminate. First, let us note that $[\![love]\!]\ object$ is itself a function, which is not called *before* being pushed into the environment: its calling is delayed until it is looked up by $[\![so\ does]\!]$. Second, even when it is called a second time, it is re-pushed onto the environment, but this time it will not be looked up again, and evaluation can terminate. Hence, unless the utterance itself contains recursive references, there is no termination issue.

The exact same mechanism accounts for "lazy pronouns" [Gea62], in sentences such as follows:

(10) The man who gave {his paycheck} to his wife was wiser than the one who gave [it] to his mistress. [Kar69]
(11) Bill wears {his hat} every day. John wears [it] only on Sundays.

In the above cases, we see that the referent itself contains an anaphoric expression, which is thus subject to the second stage of interpretation. As in the previous example, in certain contexts the strict interpretation is preferred. (Say if it is known that Bill shares a hat with John.) We can account for it by pushing the evaluated semantics in the environment. The selection between strict and sloppy readings can depend on various factors (lexical, pragmatic, etc.), which are out of scope of this paper.

3.4 Anaphoric scope flexibility: donkey sentences and E-Type pronouns

In this section, we take a look at cases of anaphora that require some scope flexibility. We discuss simpler cases like (12) and proceed to more difficult

cases of e-type and donkey anaphora, showing that our effectful semantics combined with intuitions from constructive type-theories gives a natural way of capturing the scope flexibility associated with anaphora in a natural language. We finish the section by describing a systematic, algorithmic procedure for scope extension which subsumes all the cases presented in the section.

(12) Every student admits that [he] is tired

In (12), the pronoun "He" refers to the variable bound by "every", which, strictly speaking, is not found in the syntax.

We can account for referents bound by quantifiers by using a suitable interpretation for quantifiers. We detail below the case of the quantifiers "every" and "some". Other quantifiers can be dealt with in a similar manner.

To interpret the phrase "every CN", we perform the following steps:

1. allocate a fresh variable (say x)
2. evaluate the common noun CN, obtaining a type T, and possibly affecting the environment.
3. associate the appropriate pronouns to point to x (according to the gender and number of the common noun which the quantifier applies to). Such a pronoun is a donkey pronoun: it points to an object x which is valid semantically, but not manifest in the syntax.
4. return the logical predicate $\lambda p \to \forall (x : T).\ (p\ x)$.

Formally:

$$
\begin{aligned}
&\mathit{every} :: \mathit{CN} \to \mathit{NP}\\
&\mathit{every}\ \mathit{cn} = \mathbf{do}\\
&\quad x \leftarrow \mathit{getFresh}\\
&\quad t \leftarrow \mathit{cn}\\
&\quad \mathit{modify}\ (\mathit{pushNP}\ (\mathit{pronoun}\ \mathit{cn})\ (\lambda p \to p\ x))\\
&\quad \mathit{return}\ (\lambda p \to \forall (x : t).\ (p\ x))
\end{aligned}
$$

Note that there cannot be a reference to x within cn, because x is pushed onto the environment only after evaluating cn.

3.4.1 Scoping with universal quantifiers

We underline that, according to our algorithm, all the effects associated with the quantifier persist even after closing the logical scope of a universal quantifier. This phenomenon is known as *telescoping*, which was introduced by Roberts [Rob87] (see also [PZ92]). In particular, the referent is still accessible. This allows us to interpret sentences such as (13) below.

(13) Every boy climbed on a tree. He is afraid to fall.
(14) Every boy climbed on Mary's shoulders. He likes her.
(15) Every boy climbed on a tree and was afraid to fall.
(16) Bill is old. Every boy climbed on a tree. He is afraid to fall.

Our system interprets (13) as

$$(\forall(x : boy).\ \rightarrow \exists(y : tree).\ climbed\,(x, y)) \wedge afraidToFall\,(x)$$

The above formula is not well scoped: x is unbound in the right conjunct. Yet, we regard our interpretation as *as-satisfactory-as possible*, because:

- it reflects the infelicitous character of sentences such as (13) (which is perhaps more clear in (14)).
- because any variable is bound at most once (every bound variable is fresh), it is possible to perform minimal scope extension *separately from anaphora resolution* to repair the sentence and obtain a reading equivalent to (15):

$$\forall(x : boy).\ (\exists(y : tree).\ climbed\,(x, y)) \wedge afraidToFall\,(x)$$

An alternative algorithm, proposed by Unger [Ung11], is to remove donkey pronouns from the context when the sentence which introduce them is complete. While logically consistent, such an approach has weaknesses. First, it precludes interpreting (13) at all, and thus (by default) also precludes any implicit repair by scope extension. Second, and perhaps worse, it makes further pronouns oblivious to quantifiers. Indeed, with this interpretation, in (16), the pronoun "he" refers *unambiguously* to "Bill", while we believe that the preferred interpretation is that "He" attempts to refer to "Every boy" (embracing the awkwardness).

3.4.2 CN scoping

Another consequence of letting all effects persist is that the referents introduced in the argument of the quantifier (CN) can be freely referred to later. This makes possible to correctly interpret (17), and even (18) albeit with a complicated scope-repair pass.

(17) Every boy climbed on a tree. It fell.
(18) Every committee has a chairman. He is appointed by its members.

We note in passing that in (17), after scope-resolution the existential quantifier will be the outermost one.

3.4.3 Scoping with existential quantifiers

While we regard the scope extension of universal quantification somewhat infelicitous, no such problem generally occurs with existential quantification alone.

(19) A boy climbed on a tree. [He] is afraid to fall.

In (19), the pronoun "He" refers unambiguously to the boy introduced in the first sentence. Combined with Ranta's analysis [Ran94], our framework offers an explanation for lack of awkwardness by appealing to constructive logic. If one interprets existential quantification constructively, one obtains $s : \Sigma(x : Boy)\Sigma(y : Tree).climbed(x, y)$ for the first sentence. But, when the scope of Σ is closed, one can substitute $s.x$ (the first component of Σ) for x in the environment and interpret references without any need for scope extension.

We note in passing that if one uses a type-theory with records (be it an eponymous theory like the TTR of Cooper [Coo17] or any proof assistant which happens to provide records), one can conveniently substitute record meet for conjunction, thereby avoid any substitution in the environment.

$$[x : Boy, y : Tree] \wedge [x : Individual, y : Individual, p : climbed(x, y)] \\ = [x : Boy, y : Tree, p : climbed(x, y)]$$

(In Unger's account [Ung11], existential quantifiers are left implicit. Only in a subsequent phase, existential quantification is added at the outermost possible scope. While this scheme works in many cases, it is unclear that using the outermost scope is always the best solution. We prefer to extend scope on a per-quantifier basis for a more fined-grained semantics.)

3.4.4 Scoping with implication

(20) If a man is tired, he leaves.

Our system interprets the above sentence as

$$(\exists(x : man).tired(x)) \to leave(x)$$

where x as a free occurrence. As discussed above for simple existentials, constructive type-theory informs that scope extension poses no problem whatsoever. Thus our solution is to systematically lift quantifiers in the premise to scope over an implication. Such a lifting does not change the reading of the sentence when the variable does not occur in the conclusion. It should be noted however that the quantifier changes polarity in this case.[4] Thus we obtain:

$$\forall(x : man).tired(x) \to leave(x)$$

The following example sheds some light as for why we attach the domains to the quantifiers.

(21) A man leaves if he is tired.

[4] In dynamic semantics, the observation that $\exists x.\phi \to \psi \Leftrightarrow \forall x.(\phi \to \psi)$ is known as Egli's corollary [Egl79]; it is a consequence of declaring that $(\exists x.\phi) \wedge \psi$ and $\exists x.(\phi \wedge \psi)$ are equivalent *without requiring that x is not free in ψ* (when x is not free in ψ, this equivalence holds in usual logics).

If we use predicates for CN, then the direct interpretation of the above sentence is

$$tired(x) \rightarrow (\exists x.man(x) \wedge leave(x))$$

and in this case logic dictates that there should be no polarity shift when lifting the quantification, thus we would get

$$\exists x.tired(x) \rightarrow man(x) \wedge leave(x)$$

which appears nonsensical (being tired implies being a man). Now, if we extend the scope of the quantifier with a domain, we obtain instead the following much more sensible interpretation:

$$\exists(x : man).tired(x) \rightarrow leave(x)$$

3.4.5 Prototypical donkey sentence

Consider the following sentence, famously referred to as *the donkey sentence* in the literature:

(22) Every man that owns a donkey beats it.

The compositional interpretation of the donkey sentence is:

$$\forall(x : Man).(\exists(y : Donkey).own(x, y)) \rightarrow beat(x, y)$$

On the face of it, one must extend the scope of y so that y is accessible in $beat\,(x, y)$. Extending the scope yields the following formula [Ran94]:

$$\forall(x : Man).\forall(y : Donkey).own(x, y) \rightarrow beat(x, y)$$

3.4.6 E-type pronouns

Consider the following sentences:

(23) Most boys climbed on a tree.
(24) Most boys climbed on a tree. They were afraid to fall.
(25) Most boys climbed on a tree. Every boy that climbed on a tree was afraid to fall.
(26) Most boys climbed on a tree and were afraid to fall.

The generalized quantifier "most" can be interpreted as a weakened universal quantifier, hereafter written $MOST$, so that (23) would be interpreted as follows:

$$MOST(a : \text{boy}).\ \exists(b : \text{tree}).\ \text{climbed\_on}(a, b)$$

Consequently, it would be possible to interpret references to the quantified variable by using scope extension. If we do that, we would interpret (24) in the same way as (26):

$$MOST(a : \text{boy}).\ (\exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{afraid\_to\_fall}(a)$$

However, according to Evans [Eva80], the above interpretation is incorrect. Instead, the pronoun should refer exactly to the boys that climbed the tree, and the semantics should be equivalent to that of (25):

$$(MOST(a : \text{boy}).\ \exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \\ (\forall(c : \Sigma(d : \text{boy}).\ \exists(e : \text{tree}).\ \text{climbed\_on}(d, e)).\ \text{afraid\_to\_fall}(c))$$

Fortunately, there is a simple way to fix the problem. Namely, it suffices to re-introduce the variable with the appropriate quantification once the scope of $MOST$ is closed. Thereby, if scope extension is needed, it is the latter quantifier (not $MOST$) whose scope will be extended. Thus, the interpretation of example (23) becomes:

$$(MOST(a : \text{boy}).\ \exists(c : \text{tree}).\ \text{climbed\_on}(a, c)) \wedge \\ (\forall(b : \Sigma(a : \text{boy}).\ \exists(c : \text{tree}).\ \text{climbed\_on}(a, c)).\ \text{true})$$

Note in particular the use of a Σ-type to quantify over the appropriate set and the use of the constant true to avoid making any statement when no continuation of the the sentence is present.

An advantage of this interpretation is that it works well with any location of the generalized quantifier. For example, the following sentence is interpreted correctly.

(27) Mary owns a few donkeys. Bill beats them.

3.4.7 Scope extension algorithm

In sum, the fundamental idea is that we do not interpret linguistic strings directly to a well-behaved logic, but rather use an intermediate semantic representation where there is a biunique relation between variable names and variable binders. This representation allows us to identify the desired binder for anaphoric expression regardless of semantic context. Pragmatic conditions on pronoun accessibility are checked separately.

Having seen how this idea plays out empirically, we describe it in its full generality in the rest of this section. The goal is to make sure that every variable x is bound by a quantifier somewhere in the logical expression. Even though anaphora resolution can create a reference out of scope of the corresponding quantifier, the binder is guaranteed to exist somewhere in the formula. Hence, the algorithm proceeds as follows.

1. If there is no free variable the expression is well-scoped and scope extension is thus complete. Otherwise, identify a variable which occurs free somewhere in the formula and call it x. Locate its quantifier. Let us call this quantifier ∇x (where ∇ can stand for $\forall, \exists, etc.$). Note that because we allocate a fresh variable for each of the quantifiers in our dynamic semantics, the correspondence between variables and quantifiers is unambiguous even when scoping is incorrect.

context	polarity	felicity
not ·	negative	low
· → M	negative	high
M → ·	positive	high
· M	positive	high

Table 1 Possible polarity and felicity of a few contexts. The dot indicate the position of the quantifier to lift.

2. Identify the innermost expression containing ∇x. This expression has in general the form of an operator (which we write $\circ$ here) applied to one or two operands, such as: $(\nabla x.\ M) \circ N$.
3. Determine the polarity of lifting in this context. The polarity is a function of the operator and the position of the operand. (See table 1.)
4. Rewrite this expression by lifting the quantifier above the operator $\circ$. In our example the result would be $\nabla x.\ (M \circ N)$ if the operator is positive in its first operand, and $\Delta x.(M \circ N)$ if the operator is negative in its first operand and Δ is dual to ∇. ($\forall$ is dual to $\exists$ and *vice-versa.*)[5]
   Weigh down the likelihood of the interpretation according to the felicity factor (See table 1.).
5. If the free occurrence of x was contained in N, it has now become bound. If not, we have extended the scope of ∇x, and we can loop from step 2. Because the formula is finite and each iteration extends the scope by a strictly positive amount, we can be certain that this process terminates.
6. Loop from step 1. until no free variable occurrence remains.

Finally we recall that for scope extension to work, the domain of discourse must be lifted together with the quantifier (see section 3.4.4).

Our approach to pronouns and their binding shares similarities with Dynamic Predicate Logic (DPL) of Groenendijk and Stokhof [GS91] as we treat pronouns as variables that are dynamically bound by quantifiers. That is, we allow quantified noun phrases to extend their operational scope beyond the clausal and sentential boundaries to bind variables that occur outside of the clause where their occur.

In contrast to DPL, where only an existential quantifier is capable of binding variables outside of its structural (standard) scope, in our approach, in addition to the existential quantifier, the universal quantifier can bind free variables outside of its scope. The reason for doing so is to be able to account for discourses such as (18), where universally quantified variables are referred by pronouns outside of the sentence where they occur.

(28) John does not have a car. # It is fast.

(29) A wolf might enter. It would growl.
(30) A wolf might enter. # It will growl.

[5] The dualisation of MOST and FEW can be problematic, but fortunately they rarely need to be scope-extended. Indeed, we have seen that in our interpretation of e-type pronouns that the variable bound by them is re-bound to a universal.

(31) Either there is no bathroom here or it is hidden.

In addition, we do not block any discourse referent, which is the case for theories of dynamic semantics such as DPL: entities under the scope of negation are not accessible outside of the scope (e.g. (28)). There are several reasons for doing that. Firstly, when modal operators interact with negation, discourse referents introduced under the scope of negated formula could be still accessible, like it is in (29). Since the current approach does not deal with modals, to be still able to give account to cases like (29), we prefer to make everything accessible for further usage. Another reason is that dynamic theories, following DPL, treat disjunction as purely static. In particular, given $\phi \vee \psi$, discourse referents of ϕ are not accessible for ψ. However, there is still no consensus about the issue (a commonly used counterexample to this static treatment is (31)).[6] We understand that the dynamization of all connectives (and, or, implication) and the universal quantifier can lead to various problems for logic based dynamic semantic theories as well as may allow for infelicitous discourses (e.g. (28) and (30)). However, since still there is no clear enough criteria for distinguishing phenomena that must be treated as dynamic from the ones that treated as static, we prefer to allow incorrect readings rather than block valid interpretations in certain cases. We provide Table 1 in lieu of the constraints found in dynamic semantic theories as a current heuristic while reasoning about anaphora and their bindings.

3.5 Cataphora

So far, our system captures cases where the referent precedes the anaphoric element exclusively. In this section, we explain how the system is extended to handle cataphora as well. Consider the following cataphoric cases:

(32) [He] was tired. Bill left.
(33) If Bill finds it, he will wear his coat.

Can we make "he" refer to "Bill", even though the referent occurs after the pronoun?

This can be supported by using a backward-traveling state monad, whose $\star$ operator is defined as follows. Note how the environment is threaded between calls.

$$\begin{aligned} m \star f = \lambda\rho_0 \to \mathbf{let}\ & (\rho_2, x) = m\ \rho_1 \\ & (\rho_1, y) = f\ x\ \rho_0 \\ \mathbf{in}\ & (\rho_2, y) \end{aligned}$$

The above looks *a priori* problematic because of the circular dependency:

[6] We believe that pragmatic devices should be integrated more within dynamic semantics in order to provide a more complete characterization of accessibility constraints (see [She11] for discussion).

- x depends on ρ_1
- ρ_1 depends on x

Yet, by using a non-strict evaluation strategy, as Haskell implements, one can compute a result in all the cases where the cycle is broken *dynamically*. That, if either

- the value of x does not depend on ρ_1 (for example if it does not contain a cataphoric reference), or
- the value of f does not return an environment depending on x (the return value of m).

It should be underlined that both forward and backward-traveling states can be combined in a single monad, thus accounting for both anaphora and cataphora. The bind operation for such a monad is the following:

$$\begin{aligned} m \star f = \lambda(\rho_0, \sigma_0) \rightarrow \textbf{let } & (\rho_2, \sigma_1, x) = m\ \rho_1\ \sigma_0 \\ & (\rho_1, \sigma_2, y) = f\ x\ \rho_0\ \sigma_1 \\ \textbf{in } & (\rho_2, \sigma_2, y) \end{aligned}$$

Finally, contexts could be equipped with a measure of distance and heuristics (lexical and pragmatic factors), which would allow us to choose between the possible readings. We do not do this here, and instead all interpretations are simply considered to be equally likely.

3.6 Bach-Peters Sentences

Karttunen [Kar71] defines a class of sentences, often referred to as *Bach-Peters sentences* in the literature, characterised by internal circular anaphoric dependency. Karttunen illustrates the issue with the following example:

(34) The pilot who shot at it hit the Mig that chased him.

In 3.6, the pronoun "it" from the noun phrase "the pilot who shot at it" refers to the same entity that is described by the noun phrase "the Mig that chased him", where the pronoun "him" refers to the entity referred by the noun phrase "the pilot who shot at it", which in turn is built with the help of the pronoun "it". So, there is a cyclic dependency between these two noun phrases.

As we discuss at length in the rest of the section, the definite determiner ("the") is another critical element of this class of sentences. For this reason, we will be using a special-purpose quantifier in the object logic, written $The(x : A)B$ which selects a member of the type A and makes it available in B as x. This is reminiscent of Hilbert's epsilon operator. The epsilon operator is sometimes used in the literature [EVH+95] to deal with other phenomena discussed earlier. However, we prefer to use a quantifier which follows the form of $\forall$ and $\exists$ instead: this allows a more general kind of reasoning, and is more useful for scope extension.

In agreement with the algorithm for cataphora defined above, our framework produces the following formula for example .

$$The(a : (\Sigma(b : PILOT).shotAt(b, c)).(The(c : \Sigma(d : MIG).chased(d, a)).hit(a, c)))$$

We see here that a well-formed logical formula can be produced. However, in the above form, c is not bound at its first occurence. Unfortunately, one cannot apply the scope-extension algorithm exactly as described above in this case. Indeed, attempting to swap the order of the "The" quantifiers results in a formula where a is not bound.

It is sometimes believed that the main reason for the difficulty in analysis is the circularity of anaphoric expression. However, another key ingredient is the use of a definite determiner in the second position. To see this, One can instead use an indefinite article for "Mig" (the second occurence of definite), and thus bind it existentially, as in the following example:

(35) The pilot who shot at it hit a Mig that chased him.

Then, a possible interpretation is the following:

$$The(a : (\Sigma(b : PILOT).shotAt(b, c))).\exists(c : (\Sigma(d : MIG).chased(d, a))).hit(a, c)$$

In this case, we can transform the formula to the following variant:

$$The(a : (\Sigma(b : PILOT).shotAt(b, d))).\Sigma(d : MIG).chased(d, a) \wedge hit(a, d)$$

which could in turn be potentially scope-extended to:

$$\Sigma(d : MIG).The(a : (\Sigma(b : PILOT).shotAt(b, d))).chased(d, a) \wedge hit(a, d)$$

The above formula is a direct interpretation of the following sentence:

(36) The pilot shot at a Mig that chased him and hit it.

Notice that the sentence (36) is comprised of two statements connected by the conjunction *and*. There is no circularity in the anaphoric dependency any more. However, while the sentences 3.6 and (36) are similar, their truth conditions are different: (36) is false in the case the pilot did not shot at the Mig, whereas such a claim would not work in the case of 3.6, because the noun phrase already restricts the situations only to those where the pilot shot at the Mig. Therefore, the change of quantifier that we performed is not semantic-preserving.

Having seen that the existential representation must be rejected, we return to the original form

$$The(a : (\Sigma(b : PILOT).shotAt(b, c)).(Thec : (\Sigma(d : MIG).chased(d, a)).hit(a, c)))$$

There is in fact a possible scope-transformation which makes both the pilot (a) and the Mig (c) in scope for $hit(a, c)$. The key is to simultaneously choose both the pilot and the Mig as a pair, from the relevant type. We write the result using record notation, as follows:

$$The(p : ([a : PILOT; c : MIG].shotAt(a, c) \wedge chased(c, a))).hit(p.a, p.c)$$

For completeness, we can examine what happens when the "the" quantifier in the first position is substituted by a universal quantifier:

(37) Every pilot who shot at it hit the Mig that chased him.

The interpretation exhibits the same substitution:

$$\forall(a : (\Sigma(b : PILOT).shotAt(b, c)).(The(c : \Sigma(d : MIG).chased(d, a)).hit(a, c)))$$

Here, again, quantifiers cannot be swapped, but it is valid to transform the pair of quantifiers into a universal quantifier over a pair:

$$\forall(p : ([a : PILOT; c : MIG].shotAt(a, c) \wedge chased(c, a))).hit(p.a, p.c)$$

In sum, to handle Bach-Peters sentences we need the following elements:

- a specific ε-like quantifier ("The") must be used to represent definites;
- a specific scope-extension rule dealing with this quantifier: when the quantifer The is a candidate for scope extension over another quantifer $\nabla x : C$, and the domain of "The" mentions the variable of this domain, we transform the formula into a ∇-quantification over a pair (otherwise the usual transformation applies):

$$\nabla(x : A).The(y : B).C \longrightarrow \nabla(p : [x : A, y : B]).C$$

where we subsitute $p.x$ for x and $p.y$ for y in C.

4 Implementation

If we have already presented snippets of code in the above, they have been often simplified, for pedagogical purposes. In this section we will show and comment a portion of the actual implementation, so one can see how big the gap between theory and practice is. In this section we will assume familiarity with Haskell and its standard library.

4.1 Dynamic Semantics

First we show the definition of types which serve as the interpretation of syntactic categories.

```
type Dynamic a = State Env a
type S    = Dynamic Prop
type VP   = Dynamic (Object → Prop)
type VP2  = Dynamic (Object → Object → Prop)
type CN   = Dynamic (Type, Gender, Number)
type CN2  = Dynamic ((Object → Type), Gender, Number)
type NP   = Role → Dynamic ((Object → Prop) → Prop)
```

The above types are already informative: they limit the way the dynamic semantics can be influenced by generated propositions. For the sake of example, let us consider the case with VP. We see that to obtain the proposition one

must *first* evaluate the effect, and only then one can obtain a predicate on objects. Thus the content of the predicate cannot influence the effect associated with the VP. (However, in the scope extension phase there is a deep interaction between effects and propositions, via variable bindings.)

```
data Env = Env { vpEnv     :: VPEnv,
               , vp2Env    :: VP2,
               , apEnv     :: AP,
               , cn2Env    :: CN2,
               , objEnv    :: [(Descriptor, NP)],
               , cnEnv     :: NounEnv,
               , freshVars :: [String] }
```

The environment contains:

- For each category, the phrases which can be possibly referred to by anaphora.
- A list of fresh variables, to be used with binders. (To avoid variable capture.)

In *objEnv*, descriptors allow to determine whether a pronoun can refer to its associated NP or not.

```
data Descriptor = Descriptor { dGender :: Gender
                             , dNumber :: Number
                             , dRole   :: Role }
```

The code which is responsible for pushing and looking up objects in the environment follows.

```
pushNP :: Descriptor → NP → Env → Env
pushNP d o1 Env {..} = Env { objEnv = (d, o1) : objEnv, .. }
type ObjQuery = Descriptor → Bool
getNP :: ObjQuery → Env → NP
getNP _ Env { objEnv = [] } = \_role →
  return (λvp → vp (constant "assumedObj"))
getNP q Env { objEnv = ((d, o) : os), .. } =
  if q d then o
         else getNP' q Env { objEnv = os, .. }
```

5 Evaluation

To evaluate our framework, we have constructed an implementation, as described above. We have then fed it a set of examples and checked the output for correctness. These examples are meant to showcase the strengths of our framework, but also reveal some of its weak spots.

A question that might arise is: why did we not simply use the FraCaS suite, which provides a section on anaphora with 27 examples? The answer is multi-fold:

1. FraCaS assumes a fully-fledged inference engine. In this paper we only propose a method to generate formulas.
2. Additionally, FraCaS tests success via inference problems only. This methodology tends to require examples with more context than necessary, and thus distract from the point at hand (was the anaphora properly resolved?). It even sometimes requires to infer the background (116) (in this paragraph numbers refer to examples in the anaphora section of FraCaS (27 examples numbered from 114 to 141).)
3. Additionally, FraCaS does not test for several phenomena that we capture (definite articles, sentential anaphora, reference to recent occurrences, etc.)
4. Several examples (117, 123-126, 131, 132, 138, 140, 141) rely on pragmatic factors and world knowledge, which we do not support
5. Several examples have ambiguous readings (129, 130, 135, 136)

Regardless, some of the examples of our suite are either inspired by, or taken straight from FraCaS (122, 133). In fact, while this paper was under review, Bernardy and Chatzikyriakidis [BC19] have implemented a variant of the algorithms presented here together with an inference system. We refer the interested readers to their work for more details on the combined system performance.

The complete results are shown in the appendix. The first entries match the examples shown in the body of the paper. The remaining entries test additional less important features and/or combinations of features. For every example, we show:

1. An English sentence
2. A possible parsing of that sentence, rendered with the operators that we have defined in our library
3. The first possible interpretation of the above including anaphora resolution done by our implementation. If all anaphoric expressions can be resolved, then we show the corresponding result. If not, we show one with explicit missing referents.
4. The above interpretation, after applying our algorithm for scope extension.
5. Optionally, a brief commentary of the example.

We stress that the formulas that we propose are not meant as the definitive interpretation for any given example. They are meant simply to demonstrate by example what our system is capable of (and what it is not capable of). Yet, we believe that the set of examples itself can be used as a stepping stone to construct an exhaustive test-suite for anaphora-resolution systems.

6 Related work

Among other sources, this work builds upon previous work dealing with anaphora either using constructive logics [Ran94] or monads on top of more standard logics, usually some variant of Montague Semantics [Ung11; Cha15; Cha17]. For some authors, including Ranta [Ran94], the algorithmic aspects of anaphora

resolution are left unspecified and only the existence of the referent in the logical formula seem to matter to the authors. The work presented in this paper can be seen as making such systems more precise.

Ideas similar to those presented here can be found in a number of papers by de Groote and colleagues [Gro06; QGA16]. In their work, continuations are used to get dynamic effects similar to the ones presented in this paper. [7] It is important to note that continuations have the same expressive power as monads, and thus we could have equally built our system around continuations. The continuation/monad distinction being a matter of taste, what are the essential differences between this work and that of de Groote and colleagues? First, while we embrace out-of-scope variables and repair interpretations when possible, while de Groote et al. interpret scope rigidly in function of syntax. Their approach means that either some references will not be resolved, or that quantifiers scope can never be closed. This is why we propose the above account, at the expense of not rejecting some incorrect sentences. It would be illuminating to compare the predictions that our account makes compared to the ones proposed by de Groote and colleagues on a large representative corpus, but we are lacking the parsed data to do so. Second, we have a focus on an automated implementation (witnessed by the test suite). Third, de Groote et al. aim at supporting more linguistic phenomena, such as modal subordination. While de Groote's approach to dynamics of language allows one to interpret a discourse as a logical form in a purely compositional manner, the anaphoric antecedents of pronouns that are out of structural scope of quantifiers are left unspecified; only the context from which they should be selected is given. Yet another step is needed to select from the context the right antecedents for the pronouns. The current work, by contrast, does that: anaphoric pronouns are resolved to their antecedents, and it is a part of a compositional interpretation. Since the basic premises of de Groote's work are compatible with ours, we believe that integrating their ideas with our work is possible and can be beneficial. In particular, we would like to support modal subordination as described in [QGA16].

Another related approach to study natural language semantics, including certain kinds of anaphora, was proposed by Maršík and Amblard [MA16]. They instead of monads use effects and handlers, a theory developed by Plotkin and Pretnar [PP09]. Effects and handlers can be encoded by monads, but to combine them is easer than to combine monads in general [HPP06].

Lastly, the idea of using re-interpreting effects to account for sloppy readings has been proposed before by Charlow [Cha15; Cha17]. In the way we see it, our account takes the idea further by making systematic use of the idea. Namely, every phrase should push itself into the environment for further reference. Additionally, the account proposed here offers: more flexible scoping rules and a complete, tested implementation, as well as scope extension.

[7] While this paper was under review, Itegulov, Lebedeva, and Woltzenlogel Paleo [ILWP18] presented a demo to analyze anaphora within and across the sentences, including event ones, based on de Groote and colleagues' earlier work.

7 Discussion and conclusion

In this paper, we have proposed a framework for dynamic semantics which is comprised of the following two parts:

1. An anaphora-resolution mechanism which is based on context-dependent re-evaluation of referents
2. A scope-extension mechanism

Either of these components, taken in isolation, is of moderate complexity. Yet, together, they can account for a wide range of anaphoric phenomena. This latter fact, together with a clear implementation as well as a test suite to measure the effectiveness of the present account are the main merits of the presented work as we see it.

In the future, we would like to extend our account to more anaphoric phenomena that have been puzzling in the literature. One such phenomenon is modal subordination, in particular accounting for contrasts similar to the ones shown in the above listed examples, repeated as follows:

(38) A wolf might enter. It would growl.
(39) A wolf might enter. # It will growl.

Another issue that we would like to explore is the extension of the anaphora test suite presented in this paper. The idea would be to build a complete anaphora test suite, which could be used as a benchmark for various computational semantics systems for anaphora.

Acknowledgements We warmly thank anonymous reviewers of earlier drafts of this paper, whose suggestions have led to much improvement of the contents.

The research in this paper is supported by grant 2014-39 from the Swedish Research Council, which funds the Centre for Linguistic Theory and Studies in Probability (CLASP) in the Department of Philosophy, Linguistics, and Theory of Science at the University of Gothenburg.

References

[BC19] Jean-Philippe Bernardy and Stergios Chatzikyriakidis. "A Wide-Coverage Symbolic Natural Language Inference System". In: *Proceedings of the 22nd Nordic Conference on Computational Linguistics*. ACL, 2019.

[BCM20] Jean-Philippe Bernardy, Stergios Chatzikyriakidis, and Aleksandre Maskharashvili. *A Computational Treatment of Anaphora and its Algorithmic Implementation: Extended Version*. Available on the author's homepage: `https://jyp.github.io/pdf/phoroi.pdf`. 2020.

[Bol00] Pascal Boldini. "Formalizing Contexts in Intuitionistic Type Theory". In: *Fundamenta Informaticae* 4.2 (2000).

[BS04] Chris Barker and Chung chieh Shan. "Continuations in natural language". In: *Proceedings of the fourth ACM SIGPLAN workshop on continuations, Hayo Thielecke*. 2004, pp. 55–64.

[BW99] Michael Barr and Charles Wells. *Category Theory for Computing Science*. third. Prentice Hall, 1999. ISBN: 0133238091.

[Cha14] Stergios Chatzikyriakidis. "Adverbs in a modern type theory". In: *Proceedings of LACL2014. LNCS 8535*. Ed. by N. Asher and S. Soloviev. 2014, pp. 44–56.

[Cha15] Simon Charlow. "Monadic dynamic semantics for anaphora". In: *Ohio State Dynamic Semantics Workshop* (2015).

[Cha17] Simon Charlow. "A modular theory of pronouns and binding". In: *Logic and Engineering of Natural Language Semantics (LENLS) 14*. Springer, 2017.

[CL13] Stergios Chatzikyriakidis and Zhaohui Luo. "Adjectives in a modern type-theoretical setting". In: *Formal Grammar*. Springer. 2013, pp. 159–174.

[CL17] Stergios Chatzikyriakidis and Zhaohui Luo. "On the interpretation of common nouns: Types versus predicates". In: *Modern Perspectives in Type-Theoretical Semantics*. Springer, 2017, pp. 43–70.

[Coo17] Robin Cooper. "Adapting type theory with records for natural language semantics". In: *Modern Perspectives in Type-Theoretical Semantics*. Ed. by Stergios Chatzikyriakidis and Zhaohui Luo. Springer, 2017, pp. 71–94.

[Coo79] Robin Cooper. "The interpretation of pronouns". In: *Syntax and Semantics* 10 (1979). Ed. by F. Heny and H. Schnelle, pp. 535–561.

[Egl79] Urs Egli. "The Stoic Concept of Anaphora". In: *Semantics From Different Points of View*. Ed. by Rainer Bäuerle, Urs Egli, and Arnim von Stechow. Springer Verlag, 1979, pp. 266–283.

[Eva80] Gareth Evans. "Pronouns". In: *Linguistic inquiry* 11.2 (1980), pp. 337–362.

[EVH+95] Urs Egli, Klaus Von Heusinger, et al. "The epsilon operator and E-type pronouns". In: *Amsterdam Studies in the Theory and History of Linguistic Science Series 4* (1995).

[GA12] Gianluca Giorgolo and Ash Asudeh. "Monads for Conventional Implicatures." In: *Proceedings of Sinn und Bedeutung 16.* MIT-Working Papers in Linguistics. 2012.

[GA14] Gianluca Giorgolo and Ash Asudeh. "Monads as a solution for generalized opacity". In: *EACL 2014* (2014), p. 19.

[Gea62] Peter Thomas Geach. *Reference and Generality*. Cornell University Press, 1962.

[Gro06] Philippe de Groote. "Towards a Montagovian account of dynamics". In: *Semantics and Linguistic Theory*. Vol. 16. 2006, pp. 1–16.

[GS91] Jeroen Groenendijk and Martin Stokhof. "Dynamic Predicate Logic". In: *Linguistics and Philosophy* 14.1 (1991), pp. 39–100.

[HPP06] Martin Hyland, Gordon Plotkin, and John Power. "Combining Effects: Sum and Tensor". In: *Theor. Comput. Sci.* 357.1 (July 2006), pp. 70–99. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2006.03.013. URL: http://dx.doi.org/10.1016/j.tcs.2006.03.013.

[ILWP18] Daniyar Itegulov, Ekaterina Lebedeva, and Bruno Woltzenlogel Paleo. "Sensala: a Dynamic Semantics System for Natural Language Processing". In: *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*. Santa Fe, New Mexico: Association for Computational Linguistics, Aug. 2018, pp. 123–127. URL: https://www.aclweb.org/anthology/C18-2027.

[Kar69] Lauri Karttunen. "Discourse Referents". In: *Proceedings of the 1969 Conference on Computational Linguistics*. COLING '69. Stockholm, Sweden: Association for Computational Linguistics, 1969, pp. 1–38. DOI: 10.3115/990403.990490.

[Kar71] Lauri Karttunen. "Definite Descriptions with Crossing Coreference: A Study of the Bach-Peters Paradox". In: *Foundations of Language* 7.2 (1971), pp. 157–182. ISSN: 0015900X.

[Luo12] Z. Luo. "Common Nouns as Types". In: *Logical Aspects of Computational Linguistics (LACL'2012). LNCS 7351*. Ed. by D. Bechet and A. Dikovsky. 2012.

[MA16] Jirka Maršík and Maxime Amblard. "Introducing a Calculus of Effects and Handlers for Natural Language Semantics". In: *Formal Grammar*. Ed. by Annie Foret et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 257–272. ISBN: 978-3-662-53042-9.

[Mar10] Simon Marlow. "Haskell 2010 Language Report". http://haskell.org/definition/haskell2010.pdf. 2010.

[Mog91] Eugenio Moggi. "Notions of computation and monads". In: *Information and computation* 93.1 (1991), pp. 55–92.

[PP09] Gordon Plotkin and Matija Pretnar. "Handlers of Algebraic Effects". In: *Proceedings of the 18th European Symposium on Programming Languages and Systems: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*. ESOP '09. York, UK: Springer-Verlag, 2009, pp. 80–94. ISBN: 978-3-642-00589-3. DOI: 10.1007/978-3-642-00590-9_7.

[PZ92] Massimo Poesio and Alessandro Zucchi. "On Telescoping". In: *SALT II - Proceedings from the Conference on Semantics and Linguistic Theory (Columbus, Ohio, May 1-3)*. Ed. by Chris Barker and David Dowty. 1992, pp. 347–366.

[QGA16] Sai Qian, Philippe de Groote, and Maxime Amblard. "Modal Subordination in Type Theoretic Dynamic Logic". In: *LiLT (Linguistic Issues in Language Technology)* 14 (2016).

[Ran94] A. Ranta. *Type-Theoretical Grammar*. Oxford University Press, 1994.

[Rob87] Craige Roberts. *Modal Subordination, Anaphora, and Distributivity*. 1987. URL: https://www.asc.ohio-state.edu/roberts.21/dissertation.pdf.

[Rob89] Craige Roberts. "Modal Subordination and Pronominal Anaphora in Discourse". In: *Linguistics and Philosophy* 12.6 (1989), pp. 683–721. DOI: 10.1007/BF00632602.

[Rob90] Craige Roberts. *Modal Subordination, Anaphora, and Distributivity*. Garland Press, 1990. URL: https://www.asc.ohio-state.edu/roberts.21/dissertation.pdf.

[Sha02] Chung-chieh Shan. "Monads for natural language semantics". In: *CoRR* cs.CL/0205026 (2002). URL: http://arxiv.org/abs/cs.CL/0205026.

[She11] Lewis Karen Shelley. "Understanding dynamic discourse". Ph.D. Thesis. Rutgers, The State University of New Jersey, 2011.

[Ung11] Christina Unger. "Dynamic semantics as monadic computation". In: *JSAI International Symposium on Artificial Intelligence*. Springer. 2011, pp. 68–81.

Appendix: Test suite

In the test suite we make heavy use of operators, to keep the examples concise, and in fact, readable at all. The operators are infix notations for combinators presented above, or variants thereof. We list only their types here. The complete code (including all combinators and lexical items) is available as supplementary material.

$$(!) :: NP \to VP \to S$$
$$(?) :: VP2 \to NP \to VP$$
$$(¿) :: NP \to VP2 \to VP$$
$$(\#\#\#) :: S \to S \to S$$
$$(\#) :: AP \to CN \to CN$$
$$\_of :: CN2 \to NP \to CN$$
$$that :: CN \to VP \to CN$$
$$adVP :: VP \to AdVP \to VP$$

Sometimes the environment does not contain any appropriate referent for NPs. In such a case we output "assumedNP"; If the NP is of the form "the cn" and no object satisfying cn can be found, we output $\mathbf{U}(cn)$.

1. John loves Mary. Bill likes her.
   $(johnNP\,!\,(lovesVP'\,?\,maryNP))\;\#\#\#\;(billNP\,!\,(likeVP\,?\,herNP))$
   $\mathrm{love}(\mathrm{john},\mathrm{mary}) \wedge \mathrm{like}(\mathrm{bill},\mathrm{mary})$
2. John owns an {old} car. He loves [such] cars.
   $(johnNP\,!\,(own\,?\,aDet\,(oldAP\,\#\,carCN)))\;\#\#\#\;(heNP\,!\,(lovesVP'\,?\,(suchDet\,carsCN)))$
   $(\exists(a : [b : \mathrm{car};\ p : \mathrm{old}(b)]).\ \mathrm{own}(\mathrm{john}, a)) \wedge (\forall(d : [c : \mathrm{cars};\ p : \mathrm{old}(c)]).\ \mathrm{love}(\mathrm{john}, d))$
3. Mary owns an old car. John owns a red one.
   $(maryNP\,!\,(own\,?\,aDet\,(oldAP\,\#\,carCN)))\;\#\#\#\;(johnNP\,!\,(own\,?\,(aDet\,(redAP\,\#\,one))))$
   $(\exists(a : [b : \mathrm{car};\ p : \mathrm{old}(b)]).\ \mathrm{own}(\mathrm{mary}, a)) \wedge (\exists(c : [d : \mathrm{car};\ p : \mathrm{red}(d)]).\ \mathrm{own}(\mathrm{john}, c))$

4. The population of France is larger than that of Germany
   $the\,(populationCN2\backslash\_of\grave{}\ francePN)\,!\,(is\_larger\_thanV2\,?\,thatOf\,germanyPN)$
   larger($\mathbf{U}$(population(france)), $\mathbf{U}$(population(germany)))
5. Mary fell. John did too.
   $(maryNP\,!\,fellVP)\ \#\#\#\ (johnNP\,!\,doesTooVP)$
   fell(mary) $\wedge$ fell(john)
6. Mary has read all the books that john has.
   $maryNP\,!\,(readV2\,?\,(every\,(book\backslash that\grave{}\ (johnNP\,¿\,hasTooV2))))$
   $\forall(a : [b : \text{book};\ p : \text{read(john}, b)]).\ \text{read(mary}, a)$
7. John may arrive this evening. If [so] Bill will be happy.
   $johnNP\,!\,(may\,(arriveVP\backslash adVP\grave{}\ thisEvening))\ \#\#\#\ (so\ ==>\ (billNP\,!\,ishappy))$
   may(this_evening(arrive(john))) $\wedge$ (may(this_evening(arrive(john))) $\rightarrow$ happy(bill))
   Remark: Incorrect: the sentence needs to be re-interpreted with another mood.
8. Mary fell. John did too.
   $(maryNP\,!\,fellVP)\ \#\#\#\ (johnNP\,!\,doesTooVP)$
   fell(mary) $\wedge$ fell(john)
9. A donkey leaves. The donkey is tired.
   $(aDet\ donkey)\,!\,leavesVP\ \#\#\#\ (the\ donkey\,!\,isTiredVP)$
   $(\exists(a : \text{donkey}).\ \text{leaves}(a)) \wedge \text{tired}(a)$
   $\exists(a : \text{donkey}).\ \text{leaves}(a) \wedge \text{tired}(a)$
10. A donkey leaves. The mule is tired.
    $(aDet\ donkey)\,!\,leavesVP\ \#\#\#\ (the\ mule\,!\,isTiredVP)$
    $(\exists(a : \text{donkey}).\ \text{leaves}(a)) \wedge \text{tired}(\mathbf{U}(\text{mule}))$
11. A donkey and a mule walked in. The donkey was sad. It brayed.
    $((aDet\ donkey)\backslash andNP\grave{}\ (aDet\ mule)\,!\,walkedInVP)\ \#\#\#\ (the\ donkey\,!\,was\_sadVP)\ \#\#\#\ (itNP\,!\,brayedVP)$
    $(\exists(a : \text{donkey}).\ \text{walkedin}(a)) \wedge (\exists(b : \text{mule}).\ \text{walkedin}(b)) \wedge \text{was\_sad}(a) \wedge \text{brayed}(a)$
    $\exists(a : \text{donkey}).\ \text{walkedin}(a) \wedge (\exists(b : \text{mule}).\ \text{walkedin}(b)) \wedge \text{was\_sad}(a) \wedge \text{brayed}(a)$
12. A donkey and a mule walked in. The mule was sad. It brayed.
    $((aDet\ donkey)\backslash andNP\grave{}\ (aDet\ mule)\,!\,walkedInVP)\ \#\#\#\ (the\ mule\,!\,was\_sadVP)\ \#\#\#\ (itNP\,!\,brayedVP)$
    $(\exists(a : \text{donkey}).\ \text{walkedin}(a)) \wedge (\exists(b : \text{mule}).\ \text{walkedin}(b)) \wedge \text{was\_sad}(b) \wedge \text{brayed}(b)$
    $(\exists(a : \text{donkey}).\ \text{walkedin}(a)) \wedge (\exists(b : \text{mule}).\ \text{walkedin}(b) \wedge \text{was\_sad}(b) \wedge \text{brayed}(b))$
13. {John} slapped Bill. [He] hurt his hand.
    $(johnNP\,!\,(slappedV2\,?\,billNP))\ \#\#\#\ (heNP\,!\,(hurtV2\,?\,his\ handCN2))$
    slapped(john, bill) $\wedge$ hurt(bill, the(hand(bill)))
    Remark: Incorrect: we do not report all readings in this prototype. Choosing the correct reading depends on pragmatic and lexical factors.
14. John loves his wife. Bill does too.
    $johnNP\,!\,(lovesVP'\,?\,his\ wifeCN2)\ \#\#\#\ (billNP\,!\,doesTooVP)$
    love(john, the(wife(john))) $\wedge$ love(bill, the(wife(bill)))
    Remark: Sloppy reading
15. The man who gave his paycheck to his wife was wiser than the one who gave it to his mistress.
    $the\,(man\backslash that\grave{}\,((gaveV3\backslash appVP3\grave{}\ his\ paycheckCN2)?his\ wifeCN2))!(is\_wiser\_thanV2?(the\,(one\backslash that\grave{}\ ((gaveV3\backslash appVP3\grave{}\ itNP)\,?\,his\ mistressCN2))))$
    wiser($\mathbf{U}$((Σ(a : man). gave(the(paycheck(a)), the(wife(a)), a))), $\mathbf{U}$((Σ(b : man). gave(the(paycheck(b)), the(mistress(b)), b))))
16. Bill wears his hat every day. John wears it on Sundays.
    $(billNP\,!\,((wearV2\,?\,(his\ \ hatCN2))\backslash adVP\grave{}\ \ everyday))\ \#\#\#\ (johnNP\,!\,((wearV2\,?\,itNP)\backslash adVP\grave{}\ onSundays))$
    everyday(wear(bill, the(hat(bill)))) $\wedge$ on_sundays(wear(john, the(hat(john))))
17. John wears his hat on Sundays. Mary does too.
    $(johnNP\,!\,((wearV2\,?\,(his\ hatCN2))\backslash adVP\grave{}\ onSundays))\ \#\#\#\ (maryNP\,!\,doesTooVP)$
    on_sundays(wear(john, the(hat(john)))) $\wedge$ on_sundays(wear(mary, the(hat(mary))))
18. John wears his hat on Sundays. His colleagues do too.
    $(johnNP\,!\,((wearV2\,?\,(his\ hatCN2))\backslash adVP\grave{}\ onSundays))\ \#\#\#\ (his\ colleaguesCN2\,!\,doesTooVP)$
    on_sundays(wear(john, the(hat(john))))$\wedge$on_sundays(wear(the(colleagues(john)), the(hat(the(colleagues(john)))))))
19. Bill is handsome. John loves him.
    $(billNP\,!\,isHandsomeVP)\ \#\#\#\ (johnNP\,!\,(lovesVP'\,?\,himNP))$
    handsome(bill) $\wedge$ love(john, bill)
20. Bill is handsome. John loves himself.
    $(billNP\,!\,isHandsomeVP)\ \#\#\#\ (johnNP\,!\,(lovesVP'\,?\,himSelfNP))$
    handsome(bill) $\wedge$ love(john, john)
21. Bill's wife loves him.
    $((billNP\backslash poss\grave{}\ wifeCN2)\,!\,(lovesVP'\,?\,himNP))$
    love(the(wife(bill)), bill)
22. Bill's wife loves himself.
    $((billNP\backslash poss\grave{}\ wifeCN2)\,!\,(lovesVP'\,?\,himSelfNP))$

love(the(wife(bill)), assumednp)
Remark: "himself" cannot be resolved in this instance. This is expected given the usual interpretation.

23. This car is so much better than that one.
unsupported
unsupported
Remark: There is no exophoric context in the prototype
24. A man is standing over there.
unsupported
unsupported
Remark: There is no deictic context in the prototype
25. He is not the managing director. He is.
unsupported
unsupported
Remark: There is no exophoric context in the prototype
26. Every student admits that he is tired.
$(every\ studentCN!\ (admitVP\ (heNP!\ isTiredVP)))$
$\forall(a : \text{student}).\ \text{admit}(\text{tired}(a), a)$
27. Every boy climbed on a tree. He is afraid to fall.
$(every\ boyCN!\ (climbedOnV2?\ (aDet\ treeCN)))\ \#\#\#\ (heNP!\ isAfraid)$
$(\forall(a : \text{boy}).\ \exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{afraid\_to\_fall}(a)$
$\forall(a : \text{boy}).\ (\exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{afraid\_to\_fall}(a)$
28. Every boy climbed on Mary's shoulders. He likes her.
$(every\ boyCN!\ (climbedOnV2?\ (maryNP\ `poss`\ shouldersCN2)))\ \#\#\#\ (heNP!\ (likeVP?\ herNP))$
$(\forall(a : \text{boy}).\ \text{climbed\_on}(a, \text{the}(\text{shoulders}(\text{mary})))) \wedge \text{like}(a, \text{mary})$
$\forall(a : \text{boy}).\ \text{climbed\_on}(a, \text{the}(\text{shoulders}(\text{mary}))) \wedge \text{like}(a, \text{mary})$
29. Every boy climbed on a tree and was afraid to fall.
$(every\ boyCN!\ ((climbedOnV2?\ aDet\ treeCN)\ `andVP`\ isAfraid))$
$\forall(a : \text{boy}).\ (\exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{afraid\_to\_fall}(a)$
30. Bill is old. Every boy climbed on a tree. He is afraid to fall.
$(billNP!\ isOld)\ \#\#\#\ (every\ boyCN!\ (climbedOnV2?\ (aDet\ treeCN)))\ \#\#\#\ (heNP!\ isAfraid)$
$\text{old}(\text{bill}) \wedge (\forall(a : \text{boy}).\ \exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{afraid\_to\_fall}(a)$
$\text{old}(\text{bill}) \wedge (\forall(a : \text{boy}).\ (\exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{afraid\_to\_fall}(a))$
31. Every boy climbed on a tree. It fell.
$(every\ boyCN!\ (climbedOnV2?\ (aDet\ treeCN)))\ \#\#\#\ (itNP!\ fellVP)$
$(\forall(a : \text{boy}).\ \exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{fell}(b)$
$\exists(b : \text{tree}).\ (\forall(a : \text{boy}).\ \text{climbed\_on}(a, b)) \wedge \text{fell}(b)$
Remark: Scope extension can forces the order of quantifiers.
32. Every commitee has a chairman. He is appointed by its members.
$every\ commitee!\ (has?\ (aDet\ chairman))\ \#\#\#\ (heNP!\ (isAppointedBy?\ (its\ members)))$
$(\forall(a : \text{commitee}).\ \exists(b : \text{chairman}).\ \text{have}(a, b)) \wedge \text{appoint}(\text{the}(\text{members}(a)), b)$
$\forall(a : \text{commitee}).\ \exists(b : \text{chairman}).\ \text{have}(a, b) \wedge \text{appoint}(\text{the}(\text{members}(a)), b)$
33. A boy climbed on a tree. He is afraid to fall.
$(aDet\ boyCN!\ (climbedOnV2?\ (aDet\ treeCN)))\ \#\#\#\ (heNP!\ isAfraid)$
$(\exists(a : \text{boy}).\ (\exists(b : \text{tree}).\ \text{climbed\_on}(a, b))) \wedge \text{afraid\_to\_fall}(a)$
$\exists(a : \text{boy}).\ (\exists(b : \text{tree}).\ \text{climbed\_on}(a, b)) \wedge \text{afraid\_to\_fall}(a)$
34. If a man is tired, he leaves.
$((aDet\ man)!\ isTiredVP) ==> (heNP!\ leavesVP))$
$(\exists(a : \text{man}).\ \text{tired}(a)) \rightarrow \text{leaves}(a)$
$\forall(a : \text{man}).\ \text{tired}(a) \rightarrow \text{leaves}(a)$
35. A man leaves if is he tired.
$((aDet\ man)!\ leavesVP) <== (heNP!\ isTiredVP))$
$\text{tired}(a) \rightarrow (\exists(a : \text{man}).\ \text{leaves}(a))$
$\exists(a : \text{man}).\ \text{tired}(a) \rightarrow \text{leaves}(a)$
Remark: Scope extension does not dualize the quantifier when lifting from positive position.
36. Every man that owns a donkey beats it.
$(every\ (man\ `that`\ (own?\ (aDet\ donkey)))!\ (beatV2?\ itNP))$
$\forall(a : [b : \text{man};\ c : \text{donkey};\ p : \text{own}(b, c)]).\ \text{beat}(a, a.c)$
37. Most boys climbed on a tree.
$(most\ boysCN!\ (climbedOnV2?\ (aDet\ treeCN)))$
$(MOST(a : \text{boy}).\ \exists(c : \text{tree}).\ \text{climbed\_on}(a, c)) \wedge (\forall(b : [a : \text{boy};\ c : \text{tree};\ p : \text{climbed\_on}(a, c)]).\ \text{true})$
38. Most boys climbed on a tree. They were afraid to fall.
$(most\ boysCN!\ (climbedOnV2?\ (aDet\ treeCN)))\ \#\#\#\ (theyPlNP!\ isAfraid)$
$(MOST(a : \text{boy}).\ \exists(c : \text{tree}).\ \text{climbed\_on}(a, c)) \wedge (\forall(b : [a : \text{boy};\ c : \text{tree};\ p : \text{climbed\_on}(a, c)]).\ \text{true}) \wedge \text{afraid\_to\_fall}(b.a)$
$(MOST(a : \text{boy}).\ \exists(c : \text{tree}).\ \text{climbed\_on}(a, c)) \wedge (\forall(b : [a : \text{boy};\ c : \text{tree};\ p : \text{climbed\_on}(a, c)]).\ \text{true} \wedge \text{afraid\_to\_fall}(b.a))$

39. Most boys climbed on a tree. Every boy that climbed on a tree was afraid to fall.
(*most boysCN*!(*climbedOnV2*?(*aDet treeCN*)))###(*every* (*boyCN*\`*that*\`(*climbedOnV2*?(*aDet treeCN*)))! *isAfraid*)
($MOST(a : \text{boy}).\ \exists(c : \text{tree}).\ \text{climbed\_on}(a, c)) \wedge (\forall(b : [a : \text{boy};\ c : \text{tree};\ p : \text{climbed\_on}(a, c)]).\ \text{true}) \wedge (\forall(d : [e : \text{boy};\ f : \text{tree};\ p : \text{climbed\_on}(e, f)]).\ \text{afraid\_to\_fall}(d))$
40. Most boys climbed on a tree and were afraid to fall.
(*most boysCN*! ((*climbedOnV2* ? (*aDet treeCN*)) \`*andVP*\` *isAfraid*))
($MOST(a : \text{boy}).\ (\exists(c : \text{tree}).\ \text{climbed\_on}(a, c)) \wedge \text{afraid\_to\_fall}(a)) \wedge (\forall(b : [a : \text{boy};\ p : (\exists(c : \text{tree}).\ \text{climbed\_on}(a, c)) \wedge \text{afraid\_to\_fall}(a)]).\ \text{true})$
41. Mary owns a few donkeys. Bill beats them.
(*maryNP*! (*own* ? (*few donkeys*)) ### (*billNP*! (*beatV2* ? *theyPlNP*))
($FEW(a : \text{donkeys}).\ \text{own}(\text{mary}, a)) \wedge (\forall(b : [a : \text{donkeys};\ p : \text{own}(\text{mary}, a)]).\ \text{true}) \wedge \text{beat}(\text{bill}, b.a)$
($FEW(a : \text{donkeys}).\ \text{own}(\text{mary}, a)) \wedge (\forall(b : [a : \text{donkeys};\ p : \text{own}(\text{mary}, a)]).\ \text{true} \wedge \text{beat}(\text{bill}, b.a))$
42. John does not have a car. # It is fast.
negation (*johnNP*! (*has* ? *aDet carCN*)) ### (*itNP*! *isFastVP*)
$\neg(\exists(a : \text{car}).\ \text{have}(\text{john}, a)) \wedge \text{fast}(a)$
$\forall(a : \text{car}).\ \neg\text{have}(\text{john}, a) \wedge \text{fast}(a)$
Remark: The sentence is infelicitous, but our algorithm produces a (dubious) interpretation anyway
43. A wolf might enter. It would growl.
(*aDet wolf*! (*might enterVP*)) ### (*itNP*! *would growlVP*)
$(\exists(a : \text{wolf}).\ \text{enter}(a)) \wedge \text{growl}(a)$
$\exists(a : \text{wolf}).\ \text{enter}(a) \wedge \text{growl}(a)$
Remark: Modals not supported
44. A wolf might enter. # It will growl.
(*aDet wolf*! (*might enterVP*)) ### (*itNP*! *will growlVP*)
$(\exists(a : \text{wolf}).\ \text{enter}(a)) \wedge \text{growl}(a)$
$\exists(a : \text{wolf}).\ \text{enter}(a) \wedge \text{growl}(a)$
Remark: Modals not supported
45. Either there is no bathroom here or it is hidden.
negation (*thereIs bathroomCN*) \`*orS*\` (*itNP*! *isHiddenVP*)
$\neg(\exists(a : \text{bathroom}).\ \text{here}(a)) \vee \text{hidden}(a)$
$\forall(a : \text{bathroom}).\ \neg\text{here}(a) \vee \text{hidden}(a)$
46. He was tired. Bill left.
(*heNP*! *isTiredVP*) ### (*billNP*! *leavesVP*)
$\text{tired}(\text{bill}) \wedge \text{leaves}(\text{bill})$
47. If bill finds it, he will wear his coat.
(*billNP*! (*findV2* ? *itNP*)) ==> (*heNP*! (*wearV2* ? *his coatCN2*))
$\text{find}(\text{bill}, \text{the}(\text{coat}(\text{bill}))) \rightarrow \text{wear}(\text{bill}, \text{the}(\text{coat}(\text{bill})))$
48. The pilot who shot at it hit the Mig that chased him.
epsilon (*pilot* \`*that*\` (*shotAtV2* ? *itNP*)) ! (*hitV2* ? (*epsilon* (*mig* \`*that*\` (*chasedV2* ? *himNP*′))))
$The(a : [b : \text{pilot};\ p : \text{shotat}(b, c)]).\ (The(c : [d : \text{mig};\ p : \text{chased}(d, a)]).\ \text{hit}(a, c))$
49. The pilot who shot at it hit a Mig that chased him.
epsilon (*pilot* \`*that*\` (*shotAtV2* ? *itNP*)) ! (*hitV2* ? (*aDet* (*mig* \`*that*\` (*chasedV2* ? *himNP*′))))
$The(a : [b : \text{pilot};\ p : \text{shotat}(b, c)]).\ \exists(c : [d : \text{mig};\ p : \text{chased}(d, a)]).\ \text{hit}(a, c)$
$\exists(c : [d : \text{mig};\ p : \text{chased}(d, a)]).\ (The(a : [b : \text{pilot};\ p : \text{shotat}(b, c)]).\ \text{hit}(a, c))$
50. The pilot shot a Mig that chased him and hit it.
(*the pilot*) ! ((*shotAtV2* ? (*aDet* (*mig* \`*that*\` (*chasedV2* ? *himNP*′)))) \`*andVP*\` (*hitV2* ? *itNP*))
$(\exists(a : [b : \text{mig};\ p : \text{chased}(b, \mathbf{U}(\text{pilot}))]).\ \text{shotat}(\mathbf{U}(\text{pilot}), a)) \wedge \text{hit}(\mathbf{U}(\text{pilot}), a)$
$\exists(a : [b : \text{mig};\ p : \text{chased}(b, \mathbf{U}(\text{pilot}))]).\ \text{shotat}(\mathbf{U}(\text{pilot}), a) \wedge \text{hit}(\mathbf{U}(\text{pilot}), a)$
51. Everyone admits that they are tired. Mary does too
((*everyone*! (*admitVP* (*theySingNP*! *isTiredVP*))) ### (*maryNP*! *doesTooVP*))
$(\forall(a : \text{person}).\ \text{admit}(\text{tired}(a), a)) \wedge \text{admit}(\text{tired}(\text{mary}), \text{mary})$
52. A lawyer signed every report. So did an auditor.
(*aDet lawyerCN*! (*signV2* ? (*every reportCN*))) ### (*aDet auditorCN*! *doesTooVP*)
$(\exists(a : \text{lawyer}).\ (\forall(b : \text{report}).\ \text{sign}(a, b))) \wedge (\exists(c : \text{auditor}).\ (\forall(d : \text{report}).\ \text{sign}(c, d)))$
53. John loves his spouse. Mary does too.
johnNP! (*lovesVP*′ ? *his marriedCN2*) ### (*maryNP*! *doesTooVP*)
$\text{love}(\text{john}, \text{the}(\text{married}(\text{john}))) \wedge \text{love}(\text{mary}, \text{the}(\text{married}(\text{mary})))$
54. Few congressmen love bill. They are tired.
(*few congressmen*! (*lovesVP billNP*)) ### (*theyPlNP*! *isTiredVP*)
$(FEW(a : \text{congressmen}).\ \text{love}(\text{bill}, a)) \wedge (\forall(b : [a : \text{congressmen};\ p : \text{love}(\text{bill}, a)]).\ \text{true}) \wedge \text{tired}(b.a)$
$(FEW(a : \text{congressmen}).\ \text{love}(\text{bill}, a)) \wedge (\forall(b : [a : \text{congressmen};\ p : \text{love}(\text{bill}, a)]).\ \text{true} \wedge \text{tired}(b.a))$
55. Few congressmen love bill. He is tired.
((*few congressmen*! (*lovesVP billNP*)) ### (*heNP*! *isTiredVP*))
$(FEW(a : \text{congressmen}).\ \text{love}(\text{bill}, a)) \wedge (\forall(b : [a : \text{congressmen};\ p : \text{love}(\text{bill}, a)]).\ \text{true}) \wedge \text{tired}(\text{bill})$
Remark: The e-type pronoun is plural.

56. John is tired. Bill loves him.
$(johnNP\,!\,isTiredVP)\ \#\#\#\ (billNP\,!\,(lovesVP\ himNP))$
tired(john) ∧ love(john, bill)
Remark: (Bill loves John, not himself.)
57. John is tired. Bill loves himself.
$(johnNP\,!\,isTiredVP)\ \#\#\#\ (billNP\,!\,(lovesVP\ himSelfNP))$
tired(john) ∧ love(bill, bill)
58. Few men that love their wife beat them.
$(few\ (men\ `that`\ (lovesVP\ (their\ wifeCN2))))\,!\,(beatV2?\ themSingNP)$
$(FEW(a : [c : \text{man};\ p : \text{love}(\text{the}(\text{wife}(c)), c)]).\ \text{beat}(a, \text{the}(\text{wife}(a))) \wedge (\forall(b : [a : \Sigma(c : \text{man}).\ \text{love}(\text{the}(\text{wife}(a.c)), a.c);\ p : \text{beat}(a, \text{the}(\text{wife}(a)))]).\ \text{true})$
Remark: Donkey pronoun
59. A donkey is tired. It leaves.
$(((aDet\ donkey)\,!\,isTiredVP)\ \#\#\#\ (itNP\,!\,leavesVP))$
$(\exists(a : \text{donkey}).\ \text{tired}(a)) \wedge \text{leaves}(a)$
$\exists(a : \text{donkey}).\ \text{tired}(a) \wedge \text{leaves}(a)$
Remark: Existentially-bound referrent
60. If a man beats a donkey, it is tired.
$(aDet\ man\,!\,(beatV2?\ (aDet\ donkey)))\ ==>\ (itNP\,!\,isTiredVP)$
$(\exists(a : \text{man}).\ (\exists(b : \text{donkey}).\ \text{beat}(a, b))) \rightarrow \text{tired}(b)$
$\forall(b : \text{donkey}).\ (\exists(a : \text{man}).\ \text{beat}(a, b)) \rightarrow \text{tired}(b)$
Remark: Change of polarity when doing scope extension under implication
61. Bill owns a donkey. He beats it.
$((billNP\,!\,(own?\ (aDet\ donkey)))\ \#\#\#\ (heNP\,!\,(beatV2?\ itNP)))$
$(\exists(a : \text{donkey}).\ \text{own}(\text{bill}, a)) \wedge \text{beat}(\text{bill}, a)$
$\exists(a : \text{donkey}).\ \text{own}(\text{bill}, a) \wedge \text{beat}(\text{bill}, a)$
62. A man that owns a donkey beats it.
$(aDet\ (man\ `that`\ (own?\ (aDet\ donkey)))\,!\,(beatV2?\ itNP))$
$\exists(a : [b : \text{man};\ c : \text{donkey};\ p : \text{own}(b, c)]).\ \text{beat}(a, a.c)$
63. If a man owns a donkey, he beats it.
$(aDet\ man\,!\,(own?\ (aDet\ donkey)))\ ==>\ (heNP\,!\,(beatV2?\ itNP))$
$(\exists(a : \text{man}).\ (\exists(b : \text{donkey}).\ \text{own}(a, b))) \rightarrow \text{beat}(a, b)$
$\forall(a : \text{man}).\ (\forall(b : \text{donkey}).\ \text{own}(a, b) \rightarrow \text{beat}(a, b))$
64. every man beats every donkey that he owns.
$every\ man\,!\,(beatV2?\ (every\ (donkey\ `that`\ (heNP\ ¿\ own))))$
$\forall(a : \text{man}).\ (\forall(b : [c : \text{donkey};\ p : \text{own}(a, c)]).\ \text{beat}(a, b))$
65. John leaves. He is tired.
$(johnNP\,!\,leavesVP)\ \#\#\#\ (heNP\,!\,isTiredVP)$
leaves(john) ∧ tired(john)
66. Bill owns a donkey. John owns one too.
$billNP\,!\,(own?\ (aDet\ donkey))\ \#\#\#\ (johnNP\,!\,(own?\ oneToo))$
$(\exists(a : \text{donkey}).\ \text{own}(\text{bill}, a)) \wedge (\exists(b : \text{donkey}).\ \text{own}(\text{john}, b))$
Remark: One-anaphora
67. John leaves. Mary does too.
$((johnNP\,!\,leavesVP)\ \#\#\#\ (maryNP\,!\,doesTooVP))$
leaves(john) ∧ leaves(mary)
68. John loves his wife. Bill does too.
$(johnNP\,!\,(lovesVP\ (his\ wifeCN2)))\ \#\#\#\ (billNP\,!\,doesTooVP))$
$\text{love}(\text{the}(\text{wife}(\text{john})), \text{john}) \wedge \text{love}(\text{the}(\text{wife}(\text{john})), \text{bill})$
Remark: Strict reading
69. If she loves him, Mary beats her husband.
$(sheNP\,!\,lovesVP\ himNP)\ ==>\ (maryNP\,!\,(beatV2?\ herHusbandNP))$
$\text{love}(\text{the}(\text{married}(\text{mary})), \text{mary}) \rightarrow \text{beat}(\text{mary}, \text{the}(\text{married}(\text{mary})))$
Remark: TODO: improve PC
70. Every woman that loves their husband beat him.
$every\ (woman\ `that`\ (lovesVP'?\ herHusbandNP))\,!\,(beatV2?\ himNP)$
$\forall(a : [b : \text{woman};\ p : \text{love}(b, \text{the}(\text{married}(b)))]).\ \text{beat}(a, \text{the}(\text{married}(a)))$
Remark: TODO: improve PC