The title of the paper

Philippe de Groote   Ekaterina Lebedeva

Abstract

Xxx xxx

1 Introduction

2 Mathematical preliminaries

[3]

In this section, based on [1], [2] and [6], the basic definitions and theorems of type-free (3) and simply-typed (4) lambda calculus are presented.

3 Type-free Lambda Calculus

Definition 3.1. [λ-terms] The set of **λ-terms** Λ constructed from an enumerable set of variables $V = \{v, v_1, v_2, \dots\}$ is defined inductively as follows:

$$\begin{array}{rcll} x \in V & \Longrightarrow & x \in \Lambda & \\ M, N \in \Lambda & \Longrightarrow & (MN) \in \Lambda & (\textbf{application}) \\ x \in V, M \in \Lambda & \Longrightarrow & (\lambda x.M) \in \Lambda & (\textbf{abstraction}) \end{array}$$

In the term $(\lambda x.M)$, called abstraction, the variable x is the **argument** of the function and M is the **body** of the function.

Example 3.2. The following are λ-terms:

$$\begin{array}{l} x \\ (x_1 x_2) \\ (\lambda x.(x_1 x_2)) \\ (\lambda x_1.(x_1 x_2)) \\ ((\lambda x.(x_1 x_2)) x_3) \end{array}$$

□

Remark 3.3. [Parenthesis conventions]

- application is left-associative

$$MN_1N_2 \dots N_n =_s (\dots((MN_1)N_2)\dots N_n)$$

- a sequence of λ-abstractions $\lambda x_1.(\lambda x_2.(\dots(\lambda x_n.M)))$ is abbreviated as $\lambda x_1 x_2 \dots x_n.M$

$$\lambda x_1 x_2 \dots x_n.M =_s \lambda x_1.(\lambda x_2.(\dots(\lambda x_n.M)))$$

- parentheses surrounding the body of an abstraction can be dropped

$$\lambda x_1 x_2 \dots x_n.(M) =_s \lambda x_1 x_2 \dots x_n.M$$

- outermost parentheses can be dropped

$$(M) =_s M$$

Note that according to the conventions on parentheses, term $\lambda x.MN$ is a more concise way of writing $\lambda x.(MN)$ and is not equivalent to $(\lambda x.M)N$.

Example 3.4. According to Remark 3.3, the λ-terms in Example 3.2 can be written as follows:

$$\begin{array}{l} x \\ x_1x_2 \\ \lambda x.x_1x_2 \\ \lambda x_1.x_1x_2 \\ (\lambda x.x_1x_2)x_3 \end{array}$$

□

Definition 3.5. [Free and bound variables] A variable x is **free** in a λ-term M if x is not in the scope of λx. If x is in the scope of λx, it is **bound**.

Example 3.6. In the term $(\lambda x.x_1x_2)$, the variables x_1 and x_2 are free. In the term $(\lambda x_1.x_1x_2)$, the variable x_1 is bound and the variable x_2 is free. In the term $x(\lambda x.x)$, the variable occurs free in the subterm x and bound in the subterm $\lambda x.x$. □

Definition 3.7. [Closed λ-terms]

1. The set of **free variables** of M, $FV(M)$ is defined inductively as follows:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\lambda x.M) &= FV(M) - \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \end{aligned}$$

2. M is **closed** or a **combinator**, if $FV(M) = \emptyset$

If an equation $M = N$ is provable in the lambda calculus, the provability is denoted by $\lambda \vdash M = N$ or sometimes just by $M = N$.

Definition 3.8. [Axioms and rules] For all $M, N, L, Z \in \Lambda$ the following axioms and rules hold:

$$\frac{}{(\lambda x.M)N = M[x := N]}\ \beta\text{-conversion}$$

$$\frac{}{M = M}$$

$$\frac{M = N}{N = M}$$

$$\frac{M = N \qquad N = L}{M = L}$$

$$\frac{M = N}{MZ = NZ}$$

$$\frac{M = N}{ZM = ZN}$$

$$\frac{M = N}{\lambda x.M = \lambda x.N}\ \text{rule } \xi$$

Importantly, substitution $[x := N]$ in M, denoted $M[x := N]$, is only applicable to the free occurrences of x in M. For example,

$$(xy(\lambda x.x))[x := N] = Ny(\lambda x.x)$$

Definition 3.9. [Substitution] The result of **substitution** of N for the free occurences of x in M, i.e. $M[x := N]$, is defined inductively on the structure of M as follows:

$$\begin{aligned} x[x := N] &\doteq N \\ y[x := N] &\doteq y \ \text{ provided } x \neq_s y \\ (\lambda y.M_1)[x := N] &\doteq \lambda y.(M_1[x := N]) \\ (M_1 M_2)[x := N] &\doteq (M_1[x := N])(M_2[x := N]) \\ (\lambda x.M_1)[x := N] &\doteq \lambda x.M_1 \end{aligned}$$

Lemma 3.10 (Substitution lemma)**.** *If $x \neq_s y$ and $x \notin FV(L)$, then*

$$M[x := N][y := L] =_s M[y := L][x := N[y := L]]$$

Proof. The proof is by induction on the structure of M. □

When performing a substitution $M[x := N]$, it is necessary to rename those bound variables in M that are free in N. Otherwise, the substitution may lead to a false result. For example, without renaming the bound variable x in $\lambda x.xy$, the substitution $(\lambda x.xy)[y := x]$ leads to the term $\lambda x.xx$ that acts differently from the desired term, because the free variable x became bound. However, changing x to z, for example, before making the substitution, leads to the desired term $\lambda z.zx$.

Definition 3.11. [α-conversion] A **change of bound variable** x in M, or an **α-conversion** in M, is the replacement of an occurrence of $\lambda x.N$ in M by $\lambda y.(N[x := y])$, where y does not occur in N.

Definition 3.12. [α-congruency] M and N are **α-congruent**, denoted $M =_\alpha N$, if one can result from the other by a finite series of changes of bound variables.

Example 3.13.

$$\begin{aligned} &\lambda x.xy =_\alpha \lambda z.zy \neq_\alpha \lambda x.xx \\ &\lambda xy.yx(\lambda x.x) =_\alpha \lambda xy.yx(\lambda z.z) =_\alpha \lambda zy.yz(\lambda x.x) \\ &\lambda xy.yx =_\alpha \lambda zy.yz =_\alpha \lambda zx.xz =_\alpha \lambda yx.xy \end{aligned}$$

□

It is natural to identify the terms that are α-congruent, as they represent the same processes. Moreover, the same processes can be represented by different terms. For example, $\lambda x.Mx$ and M both lead to MN when applied to N. Hence, the following rule can be introduced:

Definition 3.14. [Extensionality] **Extensionality** is the following derivation rule, provided $x \notin FV(MN)$:

$$\frac{Mx = Nx}{M = N}$$

The extensionality rule allows to prove $\lambda x.Mx = M$. Alternatively, $\lambda x.Mx = M$ can be considered to be an axiom:

Definition 3.15. [η-conversion] Let $x \notin FV(M)$. Then

$$\frac{}{\lambda x.Mx = M}\ \eta\text{-conversion}$$

Definition 3.16. [β-normal form]

1. M is a **β-normal form**, if M has no subterm of the form $(\lambda x.L)K$
2. M **has a β-normal form**, if there exists an N such that N is a β-normal form and $N = M$.

When M is a β-normal form, it is often said that M is in normal form.

Example 3.17.

1. $\lambda x.x$ is in normal form.
2. $(\lambda x.x)y$ has a normal form, namely y.
3. $(\lambda xy.y)z$ has a normal form, namely $(\lambda y.y)$.
4. $(\lambda xy.x)z$ has a normal form, namely z.
5. $(\lambda x.xx)(\lambda y.yy)$ has no normal form.

□

A notion of reduction on Λ is a binary relation on Λ. The classical notion of reduction β is defined as follows:

Definition 3.18. $\beta = \{((\lambda x.M)N, M[x := N])|M, N \in \Lambda\}$

Definition 3.19. [β-redex, β-contractum] A **β-redex** is a term M such that $(M, N) \in \beta$ for some term N. In this case N is called **β-contractum** of M.

Definition 3.20. The notion of reduction β induces the following binary relations:

$\rightarrow_\beta$ **one-step β-reduction**

$\rightarrow^*_\beta$ **β-reduction**

$=_\beta$ **β-equality** (also called **β-convertibility**)

These relations are inductively defined as follows:

$$\begin{array}{rcl}
(M,N) \in \beta & \Longrightarrow & M \rightarrow_\beta N \\
M \rightarrow_\beta N & \Longrightarrow & ZM \rightarrow_\beta ZN \\
M \rightarrow_\beta N & \Longrightarrow & MZ \rightarrow_\beta NZ \\
M \rightarrow_\beta N & \Longrightarrow & \lambda x.M \rightarrow_\beta \lambda x.N \\
\\
M \rightarrow_\beta N & \Longrightarrow & M \rightarrow^*_\beta N \\
M \rightarrow^*_\beta M & & \\
M \rightarrow^*_\beta N, N \rightarrow^*_\beta Z & \Longrightarrow & M \rightarrow^*_\beta Z \\
\\
M \rightarrow^*_\beta N & \Longrightarrow & M =_\beta N \\
M =_\beta N & \Longrightarrow & N =_\beta M \\
M =_\beta N, N =_\beta Z & \Longrightarrow & M =_\beta Z
\end{array}$$

The notions of β-redex and β-equality allow to give an alternative, more formal, definition of β-normal form:

Definition 3.21. [β-normal form]

1. A term N is called a β-normal form, if N does not contain (as subterm) any β-redex.

2. A term N is a β-normal form of M, if N is a β-normal form and $M =_\beta N$.

The notion of β-reduction is very important, because it characterizes provability in λ and it is Church-Rosser:

Proposition 3.22. *$M =_\beta N$ iff $\lambda \vdash M = N$*

Proof. See [1, p.59]. □

Theorem 3.23 (Church-Rosser theorem for $\rightarrow^*_\beta$)**.** *If $L \rightarrow^*_\beta M$ and $L \rightarrow^*_\beta N$, then there exists a term Z such that $M \rightarrow^*_\beta Z$ and $N \rightarrow^*_\beta Z$.*

Proof. See [1, p.62] or [6, p.289]. □

$$\begin{tikzcd}
& L \arrow[dl] \arrow[dr] & \\
M \arrow[dr, dashed] & & N \arrow[dl, dashed] \\
& Z &
\end{tikzcd}$$

Figure 1: Diamond property.

The Church-Rosser theorem says that for two β-convertible terms, there is a term to which they both β-reduce, as illustrated in Figure 1. The property described in the theorem, that if a term can be reduced to two different terms, then these two terms can be further reduced to one term, is called the **diamond property** or **confluence**. The theorem states that β-reduction is confluent.

4 Simply-typed Lambda Calculus

Lambda terms can be assigned expressions, called "types", to denote their intended input and output sets. There exists two typing paradigms: à la Curry, sometimes called **implicit**, and à la Church, sometimes called **explicit**. This section first recalls the basics of Curry-style approach and then briefly compares it with the Church-style.

Definition 4.1. [Simple types] Given a set A of **atomic types**, the set of **types** T is inductively defined as follows:

$$\begin{array}{rcll} \alpha \in A & \Longrightarrow & \alpha \in T & \\ \alpha, \beta \in A & \Longrightarrow & (\alpha \to \beta) \in T & (\textbf{function types}) \end{array}$$

An atomic type is intended to denote some particular set. A function type $(\alpha \to \beta)$ is intended to denote some set of functions from α to β, i.e. the functions that take as the argument a member of the set denoted by α and return as an output a member of the set denoted by β.

Remark 4.2. [Parenthesis convention] A complex functional type $(\alpha_1 \to (\alpha_2 \to \cdots \to (\alpha_{n-1} \to \alpha_n) \dots))$ is abbreviated as $\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n$ (i.e. parentheses are associated to the right):

$$(\alpha_1 \to (\alpha_2 \to \cdots \to (\alpha_{n-1} \to \alpha_n) \dots)) =_s \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n$$

Definition 4.3. [$\lambda{\to}$-Curry]

1. A **statement** is of the form $M : \sigma$ with $M \in \Lambda$ and $\sigma \in T$. The type σ is the **predicate** and the term M is the **subject** of the statement.

2. A **declaration** is a statement with a variable as a subject.

3. A **basis** is a set of declarations with distinct variables as subjects.

Definition 4.4. [Derivation rules in $\lambda{\to}$-Curry] A statement $M : \sigma$ is **derivable** from a basis Γ, denoted $\Gamma \vdash_{\lambda\to\text{-Curry}} M : \sigma$, $\Gamma \vdash_{\lambda\to} M : \sigma$ or simply $\Gamma \vdash M : \sigma$, if $\Gamma \vdash M : \sigma$ can be produced by the following rules:

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ \text{axiom}$$

$$\frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}\ \text{app}$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}\ \text{abs}$$

Lemma 4.5 (Substitution lemma for $\lambda{\to}$-Curry)**.**

1. *If* $\Gamma \vdash M : \sigma$*, then* $\Gamma[\alpha := \tau] \vdash M : \sigma[\alpha := \tau]$.

2. *Suppose* $\Gamma, x : \sigma \vdash M : \tau$ *and* $\Gamma \vdash N : \sigma$*. Then* $\Gamma \vdash M[x := N] : \tau$.

Proof. 1. The proof is by induction on the derivation of $M : \sigma$.

2. The proof is by induction on the generation of $\Gamma, x : \sigma \vdash M : \tau$

□

The following theorem states that the set of terms having a certain type is closed under reduction:

Theorem 4.6 (Subject reduction theorem for $\lambda{\to}$-Curry)**.** *Suppose* $M \to_\beta^* N$*. Then*

$$\Gamma \vdash M : \sigma \implies \Gamma \vdash N : \sigma$$

Proof. See [2, p.41]. □

While in Curry's approach each term is assigned a type after the term has been built, in Church's approach, the type of a term is integrated in the term itself. For example, the term $\lambda x.x$ can be assigned a type according to the Curry and Church styles respectively as follows:

$$\vdash_{Curry} \lambda x.x : (\sigma \to \sigma)$$
$$\vdash_{Church} \lambda x^{\sigma}.x : (\sigma \to \sigma)$$

The term $\lambda x^{\sigma}.x$ itself is annotated in a Church system by σ. This means that $\lambda x^{\sigma}.x$ takes the argument x from the particular set denoted by σ. In contrast, a Curry system allows each term to have a polymorphic type. For example, the term $\lambda x.x : (\sigma \to \sigma)$ denotes the operation of doing nothing regardless how σ is instantiated: it can stand, for example, for integers or for booleans.

Definition 4.7. [T-annotated λ-terms] Let V be a set of variables, T be a set of types. The set Λ_T of **T-annotated λ-terms** is defined as follows:

$$\begin{array}{rcl} x \in V & \Longrightarrow & x \in \Lambda_T \\ M, N \in \Lambda_T & \Longrightarrow & MN \in \Lambda_T \\ x \in V, M \in \Lambda_T, \sigma \in T & \Longrightarrow & \lambda x^{\sigma}.M \in \Lambda_T \end{array}$$

The typed lambda calculus à la Church is defined similarly to the typed lambda calculus à la Curry: an important difference is in the derivation rule corresponding to the abstraction: the abstracted variable is explicitly annotated with a type in the Church-style system. The explicit annotation of types in Church-style system makes it possible to decide whether a term has a certain type. This is an undecidable question for some Curry systems. On the other hand, a Curry-style system has more power and more flexibility than a Church-style system. For example, the easiest way to answer the question whether an untyped term M has any typed analogues is to re-state the question in Curry's notation. Furthermore, Curry-style systems can be generalized in ways Church-style systems cannot.

Terms à la Church can be easily mapped into terms à la Curry. This is done simply by "erasing" all type annotations within the term à la Church:

Definition 4.8. $|\cdot| : \Lambda_T \to \Lambda$ is defined as follows:

$$\begin{array}{rcl} |x| & \doteq & x \\ |MN| & \doteq & |M||N| \\ |\lambda x^{\sigma}.M| & \doteq & \lambda x.|M| \end{array}$$

The following proposition states that terms in the Church version project to terms in the Curry version of $\lambda\!\to$; and that terms in the Curry style can be "lifted" to terms in the Church style:

Proposition 4.9.

1. *Let $M \in \Lambda_T$. Then*

$$\Gamma \vdash_{Church} M : \sigma \quad \Longrightarrow \quad \Gamma \vdash_{Curry} |M| : \sigma$$

2. *Let $N \in \Lambda$. Then*

$$\begin{aligned}\Gamma \vdash_{Curry} N : \sigma \quad \Longrightarrow \quad & \textit{exists } M \in \Lambda_T \\ & \textit{such that } \Gamma \vdash_{Church} M : \sigma \textit{ and } |M| =_\alpha N\end{aligned}$$

Proof. Both (1) and (2) are proved by induction on the given derivation. □

See [2] and [6] for profound introductions to the two typing styles and their detailed comparisons.

5 A continuation-based account of dynamics

Philippe de Groote [4] showed that it is possible to handle dynamic phenomena of natural language by standard tools of mathematical logic, such as simply-typed lambda calculus, and, therefore, stay within Montague's program. This is accomplished in de Groote's framework, called below G_0, by providing Montague semantics with a notion of context in a systematic and precise way.

The meaning of a sentence is a function of the context. It can be expressed in lambda calculus by defining the term standing for the interpretation of a sentence as an abstraction over a variable standing for the context.

Definition 5.1. [Context, Environment] A **context** or **environment** is a term of type γ that stores the essential information from what has already been processed in the computation of the meaning of the whole discourse.

In order to make the framework flexible, the context type γ is a parameter, which can define any complex type. Therefore, there is no restriction on the representation of context. One can define it as a simple structure focusing on a particular phenomenon and elaborate it as more complex phenomena are considered.

A sentence can have a potential to change (or update) the context. The updated context has to be passed as an argument to the meaning of the subsequent sentence. In order to do so compositionally, de Groote used the notion of continuation: the meaning of a sentence not only is a function of a context, but also is a function of a continuation with respect to the computation of the meaning of the whole discourse. Within the body of the term standing for the meaning of a sentence, the continuation is given the possibly updated context and returns a proposition. Therefore, the continuation has type $(\gamma \to o)$.

Definition 5.2. [Continuation] A **continuation** is a term of type $(\gamma \to o)$ that denotes what is still to be processed in the computation of the meaning of the whole discourse.

Thus, a sentence is dynamically interpreted as a function that takes a context e of type γ and a continuation ϕ of type $(\gamma \to o)$ and returns a proposition:

$$[\![s]\!] = \underbrace{\gamma}_{\text{context}} \to \underbrace{(\gamma \to o)}_{\text{continuation}} \to \underbrace{o}_{\text{proposition}}$$

Type $(\gamma \to (\gamma \to o) \to o)$ is, therefore, defined to be the type of a dynamic proposition.

Example 5.3. The meaning of the sentence (1) is the λ-term (1):

(1) *John loves Mary.*

$$\underbrace{\lambda\ \overbrace{\underbrace{e^{\gamma}}_{\text{context}}\ \underbrace{\phi^{\gamma\to o}}_{\text{continuation}}\ .\ \overbrace{\overbrace{\overbrace{\mathbf{love}^{\iota\to\iota\to o}\ \mathbf{j}^{\iota}}^{\iota\to o}\ \mathbf{m}^{\iota}}^{o} \wedge \overbrace{\phi e^{*}}^{o}}^{o}}^{\gamma\to(\gamma\to o)\to o}}_{\text{dynamic proposition}} \tag{1}$$

where e^* is the context obtained by updating e. □

Note the presence of the conjunct ϕe^* in (1) that conveys that an updated context is passed as an argument to the continuation of a proposition, and is, therefore, accessible in the rest of the computation. This kind of conjunct is a subterm of every proposition in G_0.

In G_0 each object is interpreted as a variable (i.e. as a term of type ι). The framework is presented on the phenomena of cross-sentential and donkey anaphora and the type of context γ is defined as a list of individuals for the sake of simplicity:

$$\gamma \doteq \texttt{list of } [\ \iota\ ] \tag{2}$$

Thus, the context stores only interpretations of objects that previously occurred in the discourse. When a new object is interpreted as an individual x, the current context e is updated with x, resulting in $(x :: e)$, where $::$ is a list constructor of type $(\iota \to \gamma \to \gamma)$ (i.e. it is a function that takes an individual and a context and returns an (updated) context).[1]

Therefore, returning to Example 5.3, e^* is $(\mathbf{m} :: \mathbf{j} :: e)$. Hence, the interpretation of Sentence (1) is as follows:

$$\lambda e\phi.\mathbf{love\ j\ m} \wedge \phi(\mathbf{m} :: \mathbf{j} :: e) \tag{3}$$

Term (3) has to be computed compositionally from lexical meanings $[\![John]\!]$, $[\![Mary]\!]$ and $[\![loves]\!]$. Particularly, it has to be the result of normalizing $[\![loves]\!][\![Mary]\!][\![John]\!]$.

A noun phrase in Montague semantics is a term taking a property as an argument and returning a proposition. In framework G_0 there should be two additional arguments for a term to return a proposition. Therefore, everywhere where a term of type o occurs in Montague's interpretation, there has to be a term of type $(\gamma \to (\gamma \to o) \to o)$ in de Groote's interpretation, as can be easily seen comparing (4a) and (4b), where Ω is an abbreviation for $(\gamma \to (\gamma \to o) \to o)$. Thus, a noun phrase is interpreted as a function of three arguments (a property, a context and a continuation) that returns a proposition, as can be more easily seen in (4c), where no abbreviation is used:

$$[\![np]\!] =_{Montague} \underbrace{(\iota \to o)}_{\substack{\text{static}\\ \text{property}}} \to \underbrace{o}_{\substack{\text{static}\\ \text{proposition}}} \tag{4a}$$

$$[\![np]\!] =_{de\ Groote} \underbrace{(\iota \to \Omega)}_{\substack{\text{dynamic}\\ \text{property}}} \to \underbrace{\Omega}_{\substack{\text{dynamic}\\ \text{proposition}}} \tag{4b}$$

$$[\![np]\!] =_{de\ Groote} \underbrace{(\iota \to \gamma \to (\gamma \to o) \to o)}_{\substack{\text{dynamic}\\ \text{property}}} \to \underbrace{\underbrace{\gamma}_{\text{context}} \to \underbrace{(\gamma \to o)}_{\text{continuation}} \to \underbrace{o}_{\text{proposition}}}_{\substack{\text{dynamic}\\ \text{proposition}}} \tag{4c}$$

[1] Operation $::$ is right associative. For example, $(x :: y :: e)$ is equivalent to $(x :: (y :: e))$.

The interpretation of *Mary*, for example, is as follows:

$$[\![Mary]\!] = \lambda \overbrace{\underbrace{\mathbf{P}^{\iota\to\gamma\to(\gamma\to o)\to o}}_{\text{dynamic property}}. \underbrace{\lambda \underbrace{e^{\gamma}}_{\text{context}} \underbrace{\phi^{\gamma\to o}}_{\text{continuation}} . \overbrace{\overbrace{\overbrace{\mathbf{Pm}^{\iota}}^{\gamma\to(\gamma\to o)\to o} \ e}^{(\gamma\to o)\to o} \ \overbrace{(\lambda e'^{\gamma}. \overbrace{\phi(\overbrace{\mathbf{m} :: e'}^{\gamma})}^{o})}^{\gamma\to o}}^{o}}_{\text{dynamic proposition}}}^{(\iota\to\gamma\to(\gamma\to o)\to o)\to\gamma\to(\gamma\to o)\to o} \tag{5}$$

The interpretation of *John* is analogous:

$$[\![John]\!] = \lambda\mathbf{P}.\lambda e\phi.\mathbf{Pj}e(\lambda e'.\phi(\mathbf{j} :: e')) \tag{6}$$

A transitive verb is interpreted in Montague semantics as a term taking two type-raised individuals and returning a proposition. Since in de Groote's framework there has to be an abstraction over a context and a continuation to get a proposition, everywhere where a term of type o occurs in Montague's interpretation, there has to be a term of type $(\gamma \to (\gamma \to o) \to o)$ in de Groote's interpretation. This can be seen comparing types in (7):

$$[\![tv]\!] =_{Montague} ((\underbrace{\iota \to o)}_{\text{property}} \to \underbrace{o}_{\text{proposition}}) \to ((\underbrace{\iota \to o)}_{\text{property}} \to \underbrace{o}_{\text{proposition}}) \to \underbrace{o}_{\text{proposition}} \tag{7a}$$

$$[\![tv]\!] =_{de\ Groote} ((\underbrace{\iota \to \Omega)}_{\text{property}} \to \underbrace{\Omega}_{\text{proposition}}) \to ((\underbrace{\iota \to \Omega)}_{\text{property}} \to \underbrace{\Omega}_{\text{proposition}}) \to \underbrace{\Omega}_{\text{proposition}} \tag{7b}$$

Then the interpretation of *loves* is as follows:

$$\llbracket loves \rrbracket = \overbrace{\lambda \mathbf{Y}^{(\iota\to\Omega)\to\Omega}\mathbf{X}^{(\iota\to\Omega)\to\Omega}.\, \overbrace{\mathbf{X}(\overbrace{\lambda x.\, \overbrace{\mathbf{Y}(\overbrace{\lambda y.\overbrace{(\lambda e'^{\gamma}\phi^{\gamma\to o}.\,\overbrace{\overbrace{\overbrace{\mathbf{love}^{\iota\to\iota\to o}x^{\iota}}^{\iota\to o}y^{\iota}}^{o}\wedge\overbrace{\phi e'}^{o}}^{o})}^{\Omega}}^{\iota\to\Omega})}^{\Omega}}^{\iota\to\Omega})}^{\Omega}}^{((\iota\to\Omega)\to\Omega)\to((\iota\to\Omega)\to\Omega)\to\Omega} \tag{8}$$

Example 5.4. [**D**, *John loves Mary*] Now, given lexical interpretations (8), (5) and (6) of *loves*, *Mary* and *John* respectively, the meaning (3) of Sentence (1) can be computed compositionally according to the parse tree in Figure 2:

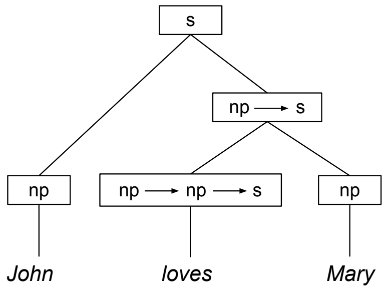

Figure 2: Syntactic parse tree of sentence *John loves Mary.*

$$
\begin{aligned}
\mathbf{D} &= [\![loves]\!][\![Mary]\!][\![John]\!] \\
&= (\lambda \mathbf{Y}\mathbf{X}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.(\lambda e'\phi.\mathbf{love}xy \wedge \phi e'))))[\![Mary]\!][\![John]\!] \\
&\to_\beta (\lambda \mathbf{X}.\mathbf{X}(\lambda x.[\![Mary]\!](\lambda y.(\lambda e'\phi.\mathbf{love}xy \wedge \phi e'))))[\![John]\!] \\
&\to_\beta [\![John]\!](\lambda x.[\![Mary]\!](\lambda y.(\lambda e'\phi.\mathbf{love}xy \wedge \phi e'))) \\
&= [\![John]\!](\lambda x.(\lambda \mathbf{P}.\lambda e\phi.\mathbf{P}\mathbf{m}e(\lambda e.\phi(\mathbf{m} :: e')))(\lambda y.(\lambda e'\phi.\mathbf{love}xy \wedge \phi e'))) \\
&\to_\beta [\![John]\!](\lambda x.\lambda e\phi.(\lambda y.(\lambda e'\phi.\mathbf{love}xy \wedge \phi e'))\mathbf{m}e(\lambda e'.\phi(\mathbf{m} :: e'))) \\
&\to_\beta [\![John]\!](\lambda x.\lambda e\phi.(\lambda e'\phi.\mathbf{love}x\mathbf{m} \wedge \phi e')e(\lambda e'.\phi(\mathbf{m} :: e'))) \\
&\to_\beta^* [\![John]\!](\lambda x.\lambda e\phi.\mathbf{love}x\mathbf{m} \wedge (\lambda e'.\phi(\mathbf{m} :: e')e)) \\
&\to_\beta [\![John]\!](\lambda x.\lambda e\phi.\mathbf{love}x\mathbf{m} \wedge \phi(\mathbf{m} :: e)) \\
&= (\lambda \mathbf{P}.\lambda e\phi.\mathbf{P}\mathbf{j}e(\lambda e'.\phi(\mathbf{j} :: e')))(\lambda x.\lambda e\phi.\mathbf{love}x\mathbf{m} \wedge \phi(\mathbf{m} :: e)) \\
&\to_\beta \lambda e\phi.(\lambda x.\lambda e\phi.\mathbf{love}x\mathbf{m} \wedge \phi(\mathbf{m} :: e))\mathbf{j}e(\lambda e'.\phi(\mathbf{j} :: e')) \\
&\to_\beta^* \lambda e\phi.\mathbf{lovejm} \wedge (\lambda e'.\phi(\mathbf{j} :: e'))(\mathbf{m} :: e) \\
&\to_\beta \lambda e\phi.\mathbf{lovejm} \wedge \phi(\mathbf{j} :: \mathbf{m} :: e)
\end{aligned} \tag{9}
$$

□

To cope with anaphora, the context has to be accessed. This is accomplished by a special function sel of type $(\gamma \to \iota)$ that takes a context and returns an individual. The function sel is assumed to implement an anaphora resolution algorithm and to work as an oracle always retrieving the correct antecedent. This allows to interpret pronouns as shown, for example, for *he* below:

$$[\![he]\!] = \overbrace{\lambda \mathbf{P}^{\iota\to\gamma\to(\gamma\to o)\to o}.\lambda e^{\gamma}\phi^{\gamma\to o}.\ \overbrace{\overbrace{\overbrace{\mathbf{P}(\overbrace{\mathsf{sel}_{he}e}^{\iota})}^{\gamma\to(\gamma\to o)\to o}\ e}^{(\gamma\to o)\to o}\ \phi}^{o}}^{(\iota\to\gamma\to(\gamma\to o)\to o)\to\gamma\to(\gamma\to o)\to o} \tag{10}$$

Example 5.5. [**S**, *He smiles at her*] The meaning of the sentence (2) computed in accordance with the parse-tree shown in Figure 3 is as follows:

(2) *He smiles at her.*

$$
\begin{aligned}
\mathbf{S} &= \llbracket smiles\_at \rrbracket \llbracket her \rrbracket \llbracket he \rrbracket \\
&= (\lambda \mathbf{Y}\mathbf{X}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.(\lambda e'\phi.\mathbf{smile}xy \wedge \phi e'))))\llbracket her \rrbracket \llbracket he \rrbracket \\
&\to_\beta (\lambda \mathbf{X}.\mathbf{X}(\lambda x.\llbracket her \rrbracket(\lambda y.(\lambda e'\phi.\mathbf{smile}xy \wedge \phi e'))))\llbracket he \rrbracket \\
&\to_\beta \llbracket he \rrbracket(\lambda x.\llbracket her \rrbracket(\lambda y.(\lambda e'\phi.\mathbf{smile}xy \wedge \phi e'))) \\
&= \llbracket he \rrbracket(\lambda x.(\lambda \mathbf{P}.\lambda e\phi.\mathbf{P}(\mathsf{sel}_{her}e)e\phi)(\lambda y.(\lambda e'\phi.\mathbf{smile}xy \wedge \phi e'))) \\
&\to_\beta \llbracket he \rrbracket(\lambda x.(\lambda e\phi.(\lambda y.(\lambda e'\phi.\mathbf{smile}xy \wedge \phi e'))(\mathsf{sel}_{her}e)e\phi)) \\
&\to_\beta \llbracket he \rrbracket(\lambda x.(\lambda e\phi.(\lambda e'\phi.\mathbf{smile}x(\mathsf{sel}_{her}e) \wedge \phi e')e\phi)) \\
&\to_\beta^* \llbracket he \rrbracket(\lambda x.(\lambda e\phi.\mathbf{smile}x(\mathsf{sel}_{her}e) \wedge \phi e)) \\
&= (\lambda \mathbf{P}.\lambda e\phi.\mathbf{P}(\mathsf{sel}_{he}e)e\phi)(\lambda x.(\lambda e\phi.\mathbf{smile}x(\mathsf{sel}_{her}e) \wedge \phi e)) \\
&\to_\beta \lambda e\phi.(\lambda x.(\lambda e\phi.\mathbf{smile}x(\mathsf{sel}_{her}e) \wedge \phi e))(\mathsf{sel}_{he}e)e\phi \\
&\to_\beta \lambda e\phi.(\lambda e\phi.\mathbf{smile}(\mathsf{sel}_{he}e)(\mathsf{sel}_{her}e) \wedge \phi e)e\phi \\
&\to_\beta^* \lambda e\phi.\mathbf{smile}(\mathsf{sel}_{he}e)(\mathsf{sel}_{her}e) \wedge \phi e
\end{aligned}
\tag{11}
$$

$\square$

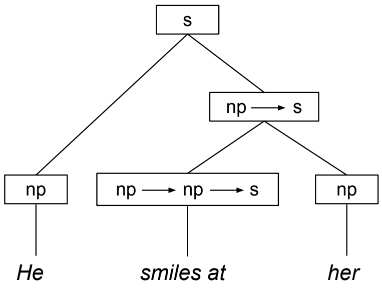

Figure 3: Syntactic parse tree of sentence *He smiles at her.*

As Example 5.5 shows, Sentence (2) is meaningful in de Groote's approach in the sense that it has an interpretation (11). However, the function sel can return individuals for *he* and *her* only when the sentence is evaluated over some context containing the corresponding antecedents. This can be done when the sentence is uttered in a discourse and the representation of this discourse is already computed. When the meaning of the discourse is updated with the meaning of the sentence, the pronominal anaphora is resolved.

Discourses in [4] are, like sentences, interpreted as terms of type $(\gamma \to (\gamma \to o) \to o)$. The update of a discourse interpreted as $\mathbf{D}$ with a sentence interpreted as $\mathbf{S}$ results in interpretation $\mathsf{upd}\ \mathbf{D}\ \mathbf{S}$ of a new discourse. This interpretation is defined by the following equation:

$$\mathsf{upd}\ \mathbf{D}\ \mathbf{S} \doteq \overbrace{\lambda e^{\gamma}\phi^{\gamma\to o}.\,\overbrace{\overbrace{\mathbf{D}^{\gamma\to(\gamma\to o)\to o}e}^{(\gamma\to o)\to o}\overbrace{(\lambda e'^{\gamma}.\,\overbrace{\overbrace{\mathbf{S}^{\gamma\to(\gamma\to o)\to o}e'}^{(\gamma\to o)\to o}\,\phi}^{o})}^{\gamma\to o}}^{o}}^{\gamma\to(\gamma\to o)\to o} \tag{12}$$

Example 5.6. [$\mathsf{upd}\ \mathbf{D}\ \mathbf{S}$] Now interpretations (9) and (11) can be composed through equation (12), regarding (1) as the discourse updated with the sentence (2). This leads to the interpretation of the piece of discourse (3):

(3) *John loves Mary. He smiles at her.*

$$\begin{aligned}
& \lambda e\phi.(\overbrace{\lambda e\phi.\mathbf{lovejm} \wedge \phi(\mathbf{j} :: \mathbf{m} :: e)}^{\mathbf{D}})e(\lambda e'.(\overbrace{\lambda e\phi.\mathbf{smile}(\mathsf{sel}_{he}e)(\mathsf{sel}_{her}e) \wedge \phi e}^{\mathbf{S}})e'\phi) \\
\to_{\beta}^{*}\ & \lambda e\phi.(\lambda e\phi.\mathbf{lovejm} \wedge \phi(\mathbf{j} :: \mathbf{m} :: e))e(\lambda e'.\mathbf{smile}(\mathsf{sel}_{he}e')(\mathsf{sel}_{her}e') \wedge \phi e') \\
\to_{\beta}^{*}\ & \lambda e\phi.\mathbf{lovejm} \wedge (\lambda e'.\mathbf{smile}(\mathsf{sel}_{he}e')(\mathsf{sel}_{her}e') \wedge \phi e')(\mathbf{j} :: \mathbf{m} :: e) \\
\to_{\beta}\ & \lambda e\phi.\mathbf{lovejm} \wedge \mathbf{smile}(\mathsf{sel}_{he}(\mathbf{j} :: \mathbf{m} :: e))(\mathsf{sel}_{her}(\mathbf{j} :: \mathbf{m} :: e)) \wedge \phi(\mathbf{j} :: \mathbf{m} :: e)
\end{aligned} \tag{13}$$

□

Interpretation (13) of the discourse consisting of the utterance of (3) is computed in a compositional manner. Note that the context of the interpretation of the first sentence is passed to sel operators of the interpretation of the second sentence. Assuming that an anaphora resolution mechanism is implemented in sel, the following semantic representation of (3) is obtained:

$$\lambda e\phi.\mathbf{love\ j\ m} \wedge \mathbf{smile\ j\ m} \wedge \phi(\mathbf{j} :: \mathbf{m} :: e) \tag{14}$$

The context ($\mathbf{j} :: \mathbf{m} :: e$) in (13) (and hence in (14)) is accessible for future computation. This means that the individuals $\mathbf{j}$ and $\mathbf{m}$ can serve as ancestors for anaphoric pronouns in the following sentences. However, this is not always the case. For example, assuming accessibility constraint requirements of DRT, the individuals introduced by quantifiers in Sentence (4) should not be accessible for anaphoric triggers outside of the sentence. However, they clearly should be accessible for anaphoric pronouns within the sentence.

(4) *Every farmer who owns a donkey beats it.*

Lexical item	Syntactic category	Continuation-based interpretation in G_0
farmer	n	$\lambda xe\phi.\mathbf{f}x \wedge \phi e$
donkey	n	$\lambda xe\phi.\mathbf{d}x \wedge \phi e$
owns	np → np → s	$\lambda\mathbf{YX}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.(\lambda e'\phi.\mathbf{o}xy \wedge \phi e')))$
beats	np → np → s	$\lambda\mathbf{YX}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.(\lambda e'\phi.\mathbf{b}xy \wedge \phi e')))$
a	n → np	$\lambda\mathbf{PQ}.\lambda e\phi.\exists(\lambda x.\mathbf{P}xe(\lambda e'.\mathbf{Q}x(x :: e')\phi))$
every	n → np	$\lambda\mathbf{PQ}.\lambda e\phi.(\forall x.\neg(\mathbf{P}xe(\lambda e'.\neg(\mathbf{Q}x(x :: e')(\lambda e'''.\top))))) \wedge \phi e$
who	(np → s) → n → n	$\lambda\mathbf{RQ}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.\mathbf{R}(\lambda\mathbf{P}.\mathbf{P}x)e'\phi)$
it	np	$\lambda\mathbf{P}.\lambda e\phi.\mathbf{P}(\mathsf{sel}_{it}e)e\phi$

Table 1: Continuation-based interpretations of lexical items of the sentence *Every farmer who owns a donkey beats it* in framework G_0.

This accessibility constraint can also be implemented in de Groote's approach. For example, lexical items of (4) can be assigned meanings shown in Table 1 that lead to the desirable interpretation of the sentence, as demonstrated below. Since the lexical interpretations are dynamic, the resulting dynamic meaning of the donkey sentence does not suffer the drawbacks of the static meaning.

Example 5.7. [*Every farmer who owns a donkey beats it*] The meaning of the noun phrase *a donkey* is computed by reducing the term $[\![a]\!][\![donkey]\!]$:

$$\begin{aligned}
[\![a]\!][\![donkey]\!] &= (\lambda\mathbf{PQ}.\lambda e\phi.\exists(\lambda y.\mathbf{P}ye(\lambda e'.\mathbf{Q}y(y :: e')\phi)))[\![donkey]\!] \\
&\to_\beta \lambda\mathbf{Q}.\lambda e\phi.\exists(\lambda y.[\![donkey]\!]ye(\lambda e'.\mathbf{Q}y(y :: e')\phi)) \\
&= \lambda\mathbf{Q}.\lambda e\phi.\exists(\lambda y.(\lambda xe\phi.\mathbf{d}x \wedge \phi e)ye(\lambda e'.\mathbf{Q}y(y :: e')\phi)) \\
&\to_\beta \lambda\mathbf{Q}.\lambda e\phi.\exists(\lambda y.(\lambda e\phi.\mathbf{d}y \wedge \phi e)e(\lambda e'.\mathbf{Q}y(y :: e')\phi)) \\
&\to_\beta^* \lambda\mathbf{Q}.\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge (\lambda e'.\mathbf{Q}y(y :: e')\phi)e) \\
&\to_\beta \lambda\mathbf{Q}.\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{Q}y(y :: e)\phi) \qquad (15)
\end{aligned}$$

Note that in Equation (15) the environment passed as an argument to $\mathbf{Q}$ contains the variable y introduced by the existential quantifier. This means that this variable is available to the formula $\mathbf{Q}$. Note also that the continuation ϕ of the term (15) is within the scope of the existential quantifier.

The meaning of the verb phrase *owns a donkey* is computed by β-reducing the term $\llbracket \mathit{owns} \rrbracket(\llbracket a \rrbracket \llbracket \mathit{donkey} \rrbracket)$:

$$\begin{aligned}
& \llbracket \mathit{owns} \rrbracket(\llbracket a \rrbracket \llbracket \mathit{donkey} \rrbracket) \\
= {} & (\lambda \mathbf{YX}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.(\lambda e'\phi.\mathbf{o}xy \wedge \phi e'))))(\llbracket a \rrbracket \llbracket \mathit{donkey} \rrbracket) \\
\rightarrow_\beta {} & \lambda \mathbf{X}.\mathbf{X}(\lambda x.(\llbracket a \rrbracket \llbracket \mathit{donkey} \rrbracket)(\lambda \mathbf{y}.(\lambda e'\phi.\mathbf{o}xy \wedge \phi e'))) \\
= {} & \lambda \mathbf{X}.\mathbf{X}(\lambda x.(\lambda \mathbf{Q}.\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{Q}y(y :: e)\phi))(\lambda y.(\lambda e'\phi.\mathbf{o}xy \wedge \phi e'))) \\
\rightarrow_\beta {} & \lambda \mathbf{X}.\mathbf{X}(\lambda x.(\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge (\lambda y.(\lambda e'\phi.\mathbf{o}xy \wedge \phi e'))y(y :: e)\phi))) \\
\rightarrow_\beta^* {} & \lambda \mathbf{X}.\mathbf{X}(\lambda x.(\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e))))
\end{aligned}$$

The dynamic meaning of the relative clause *who owns a donkey* is computed as follows:

$$\begin{aligned}
& \llbracket \mathit{who} \rrbracket(\llbracket \mathit{owns} \rrbracket(\llbracket a \rrbracket \llbracket \mathit{donkey} \rrbracket)) \\
= {} & (\lambda \mathbf{RQ}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.\mathbf{R}(\lambda \mathbf{P}.\mathbf{P}x)e'\phi))(\llbracket \mathit{owns} \rrbracket(\llbracket a \rrbracket \llbracket \mathit{donkey} \rrbracket)) \\
\rightarrow_\beta {} & \lambda \mathbf{Q}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.(\llbracket \mathit{owns} \rrbracket(\llbracket a \rrbracket \; \llbracket \mathit{donkey} \rrbracket))(\lambda \mathbf{P}.\mathbf{P}x)e'\phi) \\
= {} & \lambda \mathbf{Q}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.(\lambda \mathbf{X}.\mathbf{X}(\lambda x.(\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e)))))(\lambda \mathbf{P}.\mathbf{P}x)e'\phi) \\
\rightarrow_\beta {} & \lambda \mathbf{Q}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.(\lambda \mathbf{P}.\mathbf{P}x)(\lambda x.(\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e))))e'\phi) \\
\rightarrow_\beta {} & \lambda \mathbf{Q}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.(\lambda x.(\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e))))xe'\phi) \\
\rightarrow_\beta {} & \lambda \mathbf{Q}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.(\lambda e\phi.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e)))e'\phi) \\
\rightarrow_\beta^* {} & \lambda \mathbf{Q}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e'))) \qquad (16)
\end{aligned}$$

The meaning of *farmer who owns a donkey* is computed by applying the λ-term (16) to the lexical interpretation of *farmer*:

$$\begin{aligned}
& (\llbracket \mathit{who} \rrbracket(\llbracket \mathit{owns} \rrbracket(\llbracket a \rrbracket \llbracket \mathit{donkey} \rrbracket)))\llbracket farmer \rrbracket \\
= {} & (\lambda \mathbf{Q}x.\lambda e\phi.\mathbf{Q}xe(\lambda e'.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e'))))\llbracket farmer \rrbracket \\
\rightarrow_\beta {} & \lambda x.\lambda e\phi.\llbracket farmer \rrbracket xe(\lambda e'.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e'))) \\
= {} & \lambda x.\lambda e\phi.(\lambda xe\phi.\mathbf{f}x \wedge \phi e)xe(\lambda e'.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e'))) \\
\rightarrow_\beta {} & \lambda x.\lambda e\phi.(\lambda e\phi.\mathbf{f}x \wedge \phi e)e(\lambda e'.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e'))) \\
\rightarrow_\beta^* {} & \lambda x.\lambda e\phi.\mathbf{f}x \wedge (\lambda e'.\exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e')))e \\
\rightarrow_\beta {} & \lambda x.\lambda e\phi.\mathbf{f}x \wedge \exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e))
\end{aligned}$$

The dynamic meaning of the noun phrase *every farmer who owns a donkey* is computed by applying the meaning of *every* to the meaning of *farmer who* *owns a donkey.*

$$\begin{aligned}
&\llbracket every\rrbracket((\llbracket who\rrbracket(\llbracket owns\rrbracket(\llbracket a\rrbracket\llbracket donkey\rrbracket)))\llbracket farmer\rrbracket)\\
=\ &(\lambda\mathbf{P}\mathbf{Q}.\lambda e\phi.(\forall x.\neg(\mathbf{P}xe(\lambda e'.\neg(\mathbf{Q}x(x :: e')(\lambda e'''.\top))))) \wedge \phi e)\\
&((\llbracket who\rrbracket(\llbracket owns\rrbracket(\llbracket a\rrbracket\ \llbracket donkey\rrbracket)))\llbracket farmer\rrbracket)\\
\to_\beta\ &\lambda\mathbf{Q}.\lambda e\phi.(\forall x.\neg(((\llbracket who\rrbracket(\llbracket owns\rrbracket(\llbracket a\rrbracket\ \llbracket donkey\rrbracket)))\llbracket farmer\rrbracket)\\
&\qquad\qquad\qquad xe(\lambda e'.\neg(\mathbf{Q}x(x :: e')(\lambda e'''.\top))))) \wedge \phi e\\
=\ &\lambda\mathbf{Q}.\lambda e\phi.(\forall x.\neg((\lambda x.\lambda e\phi.\mathbf{f}x \wedge \exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e)))\\
&\qquad\qquad\qquad xe(\lambda e'.\neg(\mathbf{Q}x(x :: e')(\lambda e'''.\top))))) \wedge \phi e\\
\to_\beta\ &\lambda\mathbf{Q}.\lambda e\phi.(\forall x.\neg((\lambda e\phi.\mathbf{f}x \wedge \exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \phi(y :: e)))\\
&\qquad\qquad\qquad e(\lambda e'.\neg(\mathbf{Q}x(x :: e')(\lambda e'''.\top))))) \wedge \phi e\\
\to_\beta^*\ &\lambda\mathbf{Q}.\lambda e\phi.(\forall x.\neg(\mathbf{f}x \wedge \exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy\wedge\\
&\qquad\qquad\qquad\qquad\qquad(\lambda e'.\neg(\mathbf{Q}x(x :: e')(\lambda e'''.\top)))(y :: e)))) \wedge \phi e\\
\to_\beta\ &\lambda\mathbf{Q}.\lambda e\phi.(\forall x.\neg(\mathbf{f}x \wedge \exists(\lambda y.\mathbf{d}y \wedge \mathbf{o}xy \wedge \neg(\mathbf{Q}x(x :: y :: e)(\lambda e'''.\top))))) \wedge \phi e
\end{aligned}\tag{17}$$

Note that in Equation (17) the environment containing all the individuals with their properties collected during the computation is locally passed to the formula $\mathbf{Q}$. The continuation ϕ receives only the environment e that is passed to the term as an argument; therefore, all individuals collected during the computation of the meaning of *every farmer who owns a donkey* are not available outside the logical formula interpreting this phrase.

The meaning of the verb phrase *beats it* is computed as follows:

$$\begin{aligned}
\llbracket beats\rrbracket\llbracket it\rrbracket &= (\lambda\mathbf{Y}\mathbf{X}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.(\lambda e'\phi.\mathbf{b}xy \wedge \phi e'))))\llbracket it\rrbracket\\
&\to_\beta \lambda\mathbf{X}.\mathbf{X}(\lambda x.\llbracket it\rrbracket(\lambda y.(\lambda e'\phi.\mathbf{b}xy \wedge \phi e')))\\
&= \lambda\mathbf{X}.\mathbf{X}(\lambda x.(\lambda\mathbf{P}.\lambda e\phi.\mathbf{P}(\mathsf{sel}_{it}e)e\phi)(\lambda y.(\lambda e'\phi.\mathbf{b}xy \wedge \phi e')))\\
&\to_\beta \lambda\mathbf{X}.\mathbf{X}(\lambda x.\lambda e\phi.(\lambda y.(\lambda e'\phi.\mathbf{b}xy \wedge \phi e'))(\mathsf{sel}_{it}e)e\phi)\\
&\to_\beta \lambda\mathbf{X}.\mathbf{X}(\lambda x.\lambda e\phi.(\lambda e'\phi.\mathbf{b}x(\mathsf{sel}_{it}e) \wedge \phi e')e\phi)\\
&\to_\beta^* \lambda\mathbf{X}.\mathbf{X}(\lambda x.\lambda e\phi.\mathbf{b}x(\mathsf{sel}_{it}e) \wedge \phi e)
\end{aligned}\tag{18}$$

Finally, the dynamic meaning of the sentence is computed by applying the term (18) to the term (17):

$$\begin{aligned}
&\llbracket beats\rrbracket\llbracket it\rrbracket(\llbracket every\rrbracket((\llbracket who\rrbracket(\llbracket owns\rrbracket(\llbracket a\rrbracket\llbracket donkey\rrbracket)))\llbracket farmer\rrbracket))\\
&= (\lambda\mathbf{X}.\mathbf{X}(\lambda x.\lambda e\phi.\mathbf{b}x(\mathsf{sel}_{it}e)\wedge\phi e))\\
&\quad(\llbracket every\rrbracket((\llbracket who\rrbracket(\llbracket owns\rrbracket(\llbracket a\rrbracket\llbracket donkey\rrbracket)))\llbracket farmer\rrbracket))\\
&\to_\beta (\llbracket every\rrbracket((\llbracket who\rrbracket(\llbracket owns\rrbracket(\llbracket a\rrbracket\llbracket donkey\rrbracket)))\llbracket farmer\rrbracket))(\lambda x.\lambda e\phi.\mathbf{b}x(\mathsf{sel}_{it}e)\wedge\phi e)\\
&= (\lambda\mathbf{Q}.\lambda e\phi.(\forall x.\neg(\mathbf{f}x\wedge\exists(\lambda y.\mathbf{d}y\wedge\mathbf{o}xy\wedge\neg(\mathbf{Q}x(x::y::e)(\lambda e'''.\top)))))\wedge\phi e)\\
&\quad(\lambda x.\lambda e\phi.\mathbf{b}x(\mathsf{sel}_{it}e)\wedge\phi e)\\
&\to_\beta \lambda e\phi.(\forall x.\neg(\mathbf{f}x\wedge\exists(\lambda y.\mathbf{d}y\wedge\mathbf{o}xy\wedge\\
&\qquad\qquad\neg((\lambda x.\lambda e\phi.\mathbf{b}x(\mathsf{sel}_{it}e)\wedge\phi e)x(x::y::e)(\lambda e'''.\top)))))\wedge\phi e\\
&\to_\beta \lambda e\phi.(\forall x.\neg(\mathbf{f}x\wedge\exists(\lambda y.\mathbf{d}y\wedge\mathbf{o}xy\wedge\\
&\qquad\qquad\neg((\lambda e\phi.\mathbf{b}x(\mathsf{sel}_{it}e)\wedge\phi e)(x::y::e)(\lambda e'''.\top)))))\wedge\phi e\\
&\to^*_\beta \lambda e\phi.(\forall x.\neg(\mathbf{f}x\wedge\exists(\lambda y.\mathbf{d}y\wedge\mathbf{o}xy\wedge\\
&\qquad\qquad\neg(\mathbf{b}x(\mathsf{sel}_{it}(x::y::e))\wedge(\lambda e'''.\top)(x::y::e)))))\wedge\phi e\\
&\to_\beta \lambda e\phi.(\forall x.\neg(\mathbf{f}x\wedge\exists(\lambda y.\mathbf{d}y\wedge\mathbf{o}xy\wedge\neg(\mathbf{b}x(\mathsf{sel}_{it}(x::y::e))\wedge\top))))\wedge\phi e
\end{aligned}\tag{19}$$

The resulting dynamic interpretation (19) of the donkey sentence is logically equivalent to (20):

$$\lambda e\phi.\forall(\lambda x.\mathbf{f}x\to\forall(\lambda y.(\mathbf{d}y\wedge\mathbf{o}xy)\to\mathbf{b}x(\mathsf{sel}_{it}(y::x::e)))))\wedge\phi e\tag{20}$$

Note that, in accordance with DRT's accessibility constraint, the individuals bound by quantifiers are not accessible outside the sentence. □

First of all, the second argument of $\mathbf{b}$, standing for the anaphoric pronoun, is not a free dummy variable, but a term $(\mathsf{sel}_{it}(y::x::e))$. This term consists of the selection function sel that takes as argument a context containing the available individuals "collected" during the computation. Thus, in contrast to the static case, the second argument of $\mathbf{b}$ is self-sufficient: the function sel, which implements an anaphora resolution algorithm, selects a required individual from the context. In the current case, the selection function returns the individual y, leading to the final dynamic meaning (21) of Sentence (4):

$$\lambda e\phi.\forall(\lambda x.\mathbf{f}x\to\forall(\lambda y.(\mathbf{d}y\wedge\mathbf{o}xy)\to\mathbf{b}xy))\wedge\phi e\tag{21}$$

Moreover, in the dynamic interpretation, unlike in the static one, the formula $\mathbf{b}xy$ is within the scope of the quantifier binding the variable y, exactly as desired. Furthermore, the quantifier binding y has been changed during the computation from existential to universal, which is also impossible in the static approach. These improvements are the consequences of employing a continuation-passing technique.

The list below summarizes the advantages that de Groote's approach brought to compositional semantics:

- The approach is independent of the intermediate language used to express meanings of the expressions. This allows to use mathematical and logical theories developed outside computational linguistics.[2] Therefore, natural language phenomena can be explained in terms of well-established and well-understood theories.

- Variables do not have any special status and are variables in the usual mathematical sense. Therefore, the notions of free and bound variables are standard.

- There is no imperative dynamic notions as assignment functions, therefore destructive assignment problem does not hold. Meanings assigned to expressions are closed λ-terms.

- There is no need for rules that artificially extend the scope of quantifiers.

- Context and content are regarded separately, but they do interact during the computation of the meaning of discourse.

- The approach does not depend on any specific structure given to the context. In contrast, context is defined as a term of type parameter γ and, therefore, its structure can be altered when necessary.

- The approach is truly compositional: the meaning of a complex expression is computed by β-reducing the composition of the meanings of its lexical items.

6 The underlying dynamic logic

Although compositional dynamic framework G_0, introduced in [4] and reviewed in the previous section, have shown itself to be promising by successfully handling donkey anaphora, its interpretations look complex and the computation of the meaning can be difficult to understand. In his later talks,

[2] An extension of first logic language with λ is used here because it is convenient and intuitive.

de Groote proposed an improvement of framework G_0, called here framework G, that represents his semantics in a more concise and systematic way. To do so, de Groote defined a continuation-based dynamic logic and this section presents this logic.

6.1 Formal Details

Terms and types are given by Definitions 6.1 and 6.3:

Definition 6.1. [λ-terms] The set of **λ-terms** Λ is constructed from an enumerable set of variables $V = \{v, v_1, v_2, \dots\}$, logical constants $\wedge, \exists, \neg$, two special constants :: and sel, an enumerable set of predicate symbols $R = \{R_1, R_2, \dots\}$ and an enumerable set of constants $K = \{c, c_1, c_2, \dots\}$ using application and (function) abstraction:

$$\begin{array}{rcl}
x \in V & \Longrightarrow & x \in \Lambda \\
c \in K & \Longrightarrow & c \in \Lambda \\
M, N \in \Lambda & \Longrightarrow & (MN) \in \Lambda \\
x \in V, M \in \Lambda & \Longrightarrow & (\lambda x.M) \in \Lambda \\
M \in \Lambda & \Longrightarrow & (\exists M) \in \Lambda \\
M, N \in \Lambda & \Longrightarrow & (M \wedge N) \in \Lambda \\
M \in \Lambda & \Longrightarrow & (\neg M) \in \Lambda \\
M, x \in \Lambda & \Longrightarrow & (M :: x) \in \Lambda \\
M \in \Lambda & \Longrightarrow & (\mathsf{sel}(M)) \in \Lambda
\end{array}$$

Definition 6.2. [Free variables] The set of **free variables** of t, $FV(t)$, is defined inductively as follows:

$$\begin{aligned}
FV(x) &= \{x\} \\
FV(c) &= \{\} \\
FV(MN) &= FV(M) \cup FV(N) \\
FV(\lambda x.M) &= FV(M) - \{x\}
\end{aligned}$$

Definition 6.3. [Types] The set T of types is defined inductively as follows:

$$\begin{array}{rll}
\text{Atomic types:} & \iota \in \mathrm{T} & \text{(static individuals)} \\
 & o \in \mathrm{T} & \text{(static propositions)} \\
 & \gamma \in \mathrm{T} & \text{(context)} \\
 & & \\
\text{Complex types:} & \alpha, \beta \in \mathrm{T} \Longrightarrow & (\alpha \to \beta) \in \mathrm{T}
\end{array}$$

Typing rules define typing relations between terms and types:

Definition 6.4. [Typing rules] A statement $t : \delta$, meaning t has type δ, is **derivable** from the basis Δ, i.e. $\Delta \vdash t : \delta$, if $\Delta \vdash t : \delta$ can be produced using the following rules:

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ \text{axiom}$$

$$\frac{}{\Gamma \vdash \top : o}\ \text{axiom}$$

$$\frac{}{\Gamma, M : o, N : o \vdash M \wedge N : o}$$

$$\frac{}{\Gamma, M : \iota \to o \vdash \exists M : o}$$

$$\frac{}{\Gamma, M : o \vdash \neg M : o}$$

$$\frac{}{\Gamma, c : \gamma \vdash \mathsf{sel}\ c : \iota}$$

$$\frac{}{\Gamma, i : \iota, c : \gamma \vdash (i :: c) : \gamma}$$

$$\frac{}{\Gamma \vdash c_{iv} : \iota \to o}\ \text{axiom}$$

$$\frac{}{\Gamma \vdash c_{tv} : \iota \to \iota \to o}\ \text{axiom}$$

$$\frac{}{\Gamma \vdash c_{n} : \iota \to o}\ \text{axiom}$$

$$\frac{}{\Gamma \vdash c_{np} : (\iota \to o) \to o}\ \text{axiom}$$

$$\frac{}{\Gamma \vdash c_{rp} : (((\iota \to o) \to o) \to o) \to (\iota \to o) \to (\iota \to o)}\ \text{axiom}$$

$$\frac{\Gamma \vdash v : \alpha \to \beta \qquad \Gamma \vdash u : \alpha}{\Gamma \vdash vu : \beta}\ \text{app}$$

$$\frac{\Gamma, x : \alpha \vdash v : \beta}{\Gamma \vdash \lambda x.v : \alpha \to \beta}\ \text{abs}$$

where c_{tv}, c_{iv}, c_n, c_{np} and c_{rp} are constants standing for transitive and intransitive verbs, nouns, noun phrases and relative pronouns respectively. Typing rules for other syntactic categories can be added analogously.

The first axiom and the two rules (application and abstraction) are standard typing relations in simply-typed lambda calculus. The second axiom determines the type of the logical $\top$ symbol. The constant sel has type $(\gamma \to \iota)$ and the constant $::$ has type $(\iota \to \gamma \to \gamma)$.

Each static type can be dynamized in the following way:

Definition 6.5. [Dynamization of types] Let ι and o be atomic types, γ be a type parameter, α and β be arbitrary types. Then the types are **dynamized** in the following way:

$$\overline{\iota} \doteq \iota \tag{22a}$$

$$\overline{o} \doteq \gamma \to (\gamma \to o) \to o \tag{22b}$$

$$\overline{\alpha \to \beta} \doteq \overline{\alpha} \to \overline{\beta} \tag{22c}$$

Note that the type o of a static proposition is transformed to type $(\gamma \to (\gamma \to o) \to o)$. Type $(\gamma \to (\gamma \to o) \to o)$ is thus the type of a dynamic proposition. Therefore, dynamic propositions are functions from γ (type of context) and $(\gamma \to o)$ (type of continuation) to o (type of static proposition). Static and dynamic individuals are defined to have the same type.

For each logical constant ($\neg$, $\wedge$ and $\exists$), its dynamic counterpart is specified by the following definition:

Definition 6.6. [Dynamic logical constants] Let $\mathbf{A}$ and $\mathbf{B}$ be terms of type $(\gamma \to (\gamma \to o) \to o)$, $\mathbf{P}$ be the term of type $(\iota \to \gamma \to (\gamma \to o) \to o)$, e and e' be terms of type γ, ϕ be a term of type $(\gamma \to o)$, x be the term of type ι. **Dynamic negation, conjunction and existential quantification** are defined respectively as follows:

$$\neg\!\shortmid \mathbf{A} \doteq \lambda e\phi.\neg(\mathbf{A}e(\lambda e'.\top)) \wedge \phi e \tag{23a}$$

$$\mathbf{A} \mathbin{/\!\!\!\wedge} \mathbf{B} \doteq \lambda e\phi.\mathbf{A}e(\lambda e'.\mathbf{B}e'\phi) \tag{23b}$$

$$\exists\!\shortmid(\lambda x.\mathbf{P}[x]) \doteq \lambda e\phi.\exists(\lambda x.\mathbf{P}[x]\ (x :: e)\ \phi) \tag{23c}$$

Dynamic negation of a dynamic proposition $\mathbf{A}$, $\neg\!\shortmid \mathbf{A}$, is an abbreviation used for the term shown in (23a). Within the body of this term, the continuation and context of $\mathbf{A}$ are "erased" (by giving the term $(\lambda e'.\top)$ as the second argument of $\mathbf{A}$)[3], the resulting static proposition is negated, and the conjunct ϕe is added. Note that both ϕ and e are variables bound by λ. Therefore, the context of $\mathbf{A}$ is not available to the continuation of the resulting term $\neg\!\shortmid \mathbf{A}$. Dynamic existentially quantified term $\exists\!\shortmid(\lambda x.\mathbf{P}[x])$ is an

[3] This can be more clearly seen from Corollary 6.11.

abbreviation for the term shown in (23c). It has a λ-abstraction over variables e and ϕ and an existentially quantified variable x. Its body contains $\mathbf{P}[x]$, which is given e updated with x, $(x :: e)$, and ϕ as arguments. Note that $\neg\!\shortmid$ and $\exists\!\shortmid$ are defined respectively via $\neg$ and $\exists$. Dynamic conjunction is defined as a composition of two dynamic terms. The logical conjunction of static propositions is provided by the fact that each dynamic proposition has a conjunct ϕe in its body, as defined in (24a) of the next definition.

Given dynamic logical connectives, all static terms can be dynamized:

Definition 6.7. [Dynamization of terms] Let P be a term of type $(\iota_1 \to \cdots \to \iota_n \to o)$, A and B be terms of type o, $t_1, \ldots, t_n$ and x be terms of type ι. Then propositions, negated propositions, conjunctions of two propositions and existentially quantified propositions are dynamized in the following way:

$$\overline{\mathrm{P}t_1 \ldots t_n} \doteq \lambda e\phi.\mathrm{P}t_1 \ldots t_n \land \phi e \tag{24a}$$

$$\overline{\neg \mathrm{A}} \doteq \neg\!\shortmid \overline{\mathrm{A}} \tag{24b}$$

$$\overline{\mathrm{A} \land \mathrm{B}} \doteq \overline{\mathrm{A}} \mathbin{/\!\!\!\land} \overline{\mathrm{B}} \tag{24c}$$

$$\overline{\exists(\lambda x.\mathrm{P}[x])} \doteq \exists\!\shortmid(\lambda x.\overline{\mathrm{P}[x]}) \tag{24d}$$

Equation (24a) defines the dynamization of a proposition $\mathrm{P}t_1 \ldots t_n$ of type o by adding a λ-abstraction with two arguments e and ϕ (of types γ and $(\gamma \to o)$ respectively) and a conjunct ϕe. Therefore, the resulting dynamic term is of type $(\gamma \to (\gamma \to o) \to o)$, the type of a dynamic proposition. Equations 24b, 24c and 24d extend dynamization to non-atomic formulas.

Note that Definitions 6.6 and 6.7 allow representing de Groote's [4] (Section 5) dynamic terms in a compact way. While interpretations in [4] explicitly show extra parameters, i.e. contexts and continuations, these new definitions make it possible to hide these parameters. Moreover, the resulting compact dynamic terms are structurally analogous to their original static counterparts and, hence, are more intuitive.

Remark 6.8. In equations below, terms on the left side of $\doteq$ abbreviate respective terms on the right side:

$$\mathbf{A} \Rightarrow \mathbf{B} \doteq \neg\!\shortmid(\mathbf{A} \mathbin{/\!\!\!\land} \neg\!\shortmid \mathbf{B}) \tag{25}$$

$$\forall\!\!\!\forall(\lambda x.\mathbf{P}[x]) \doteq \neg\!\shortmid \exists\!\shortmid(\lambda x. \neg\!\shortmid \mathbf{P}[x]) \tag{26}$$

Proposition 6.9 proves an important β-equivalence that can be useful when computing interpretations of certain phrases containing an existentially quantified variable.

Proposition 6.9. *For all terms* A *and* B *of type* o *such that* $x \in FV(\mathrm{A})$ *and* $x \notin FV(\mathrm{B})$ *for an* x *of type* ι*, the following equivalence holds:*

$$\exists\!\!\!\exists(\lambda x.\overline{\mathrm{A}[x]}) \mathbin{\wedge\!\!\!\wedge} \overline{\mathrm{B}} =_\beta \exists\!\!\!\exists(\lambda x.\overline{\mathrm{A}[x]} \mathbin{\wedge\!\!\!\wedge} \overline{\mathrm{B}})$$

Proof.

$$\begin{aligned}
&\exists\!\!\!\exists(\lambda x.\overline{\mathrm{A}[x]}) \mathbin{\wedge\!\!\!\wedge} \overline{\mathrm{B}} \\
\to_\beta^* \;& (\lambda e\phi.\exists(\lambda x.\mathrm{A}[x] \wedge \phi(x :: e))) \mathbin{\wedge\!\!\!\wedge} \overline{\mathrm{B}} && \text{(by (23c))} \\
=\;& \lambda e\phi.(\lambda e\phi.\exists(\lambda x.\mathrm{A}[x] \wedge \phi(x :: e)))e(\lambda e.\overline{\mathrm{B}}e\phi) && \text{(by (23b))} \\
\to_\beta^* \;& \lambda e\phi.(\lambda e\phi.\exists(\lambda x.\mathrm{A}[x] \wedge \phi(x :: e)))e(\lambda e.\mathrm{B} \wedge \phi e) && \text{(by (24a))} \\
\to_\beta^* \;& \lambda e\phi.\exists(\lambda x.\mathrm{A}[x] \wedge (\mathrm{B} \wedge \phi(x :: e))) && (27)
\end{aligned}$$

$$\begin{aligned}
&\exists\!\!\!\exists(\lambda x.\overline{\mathrm{A}[x]} \mathbin{\wedge\!\!\!\wedge} \overline{\mathrm{B}}) \\
=\;& \exists\!\!\!\exists(\lambda x.\lambda e\phi.\overline{\mathrm{A}[x]}e(\lambda e.\overline{\mathrm{B}}e\phi)) && \text{(by (23b))} \\
\to_\beta^* \;& \exists\!\!\!\exists(\lambda x.\lambda e\phi.(\lambda e\phi.\mathrm{A}[x] \wedge \phi e)e(\lambda e.\mathrm{B} \wedge \phi e)) && \text{(by (24a))} \\
\to_\beta^* \;& \exists\!\!\!\exists(\lambda x.\lambda e\phi.\mathrm{A}[x] \wedge (\mathrm{B} \wedge \phi e)) && (28) \\
\to_\beta^* \;& \lambda e\phi.\exists(\lambda x.\mathrm{A}[x] \wedge (\mathrm{B} \wedge \phi(x :: e))) && \text{(by (23c))}
\end{aligned}$$

□

Proposition 6.10. *For all* h *and* v *of type* o*, and for all* u *of type* γ*, the following holds:*

$$\overline{h}u(\lambda e.v) = h \wedge v$$

where e *is a variable of type* γ*.*

Proof. The proof is by induction on the structure of the term h.

- h is a proposition of the form $\mathrm{P}t_1 \ldots t_n$.

$$\begin{aligned}
\overline{\mathrm{P}t_1 \ldots t_n}u(\lambda e.v) &= (\lambda e\phi.\mathrm{P}t_1 \ldots t_n \wedge \phi e)u(\lambda e.v) && \text{(by (24a))} \\
&\to_\beta (\lambda \phi.\mathrm{P}t_1 \ldots t_n \wedge \phi u)(\lambda e.v) \\
&\to_\beta \lambda \phi.\mathrm{P}t_1 \ldots t_n \wedge (\lambda e.v)u \\
&\to_\beta \lambda \phi.\mathrm{P}t_1 \ldots t_n \wedge v
\end{aligned}$$

- h is a negated proposition $\neg w$.

$$\begin{aligned}
\overline{\neg w}u(\lambda e.v) &= \boldsymbol{\neg}\,\overline{w}u(\lambda e.v) && \text{(by (24b))}\\
&= (\lambda e\phi.\neg(\overline{w}e(\lambda e'.\top)) \wedge \phi e)u(\lambda e.v) && \text{(by (23a))}\\
&\rightarrow_\beta (\lambda \phi.\neg(\overline{w}u(\lambda e'.\top)) \wedge \phi u)(\lambda e.v) &&\\
&\rightarrow_\beta \neg(\overline{w}u(\lambda e'.\top)) \wedge (\lambda e.v)u &&\\
&\rightarrow_\beta \neg(\overline{w}u(\lambda e'.\top)) \wedge v &&\\
&= \neg(w \wedge \top) \wedge v && \text{(by I.H.)}\\
&= \neg w \wedge v && (29)
\end{aligned}$$

- h is a conjunction of two propositions $w \wedge z$.

$$\begin{aligned}
(\overline{w \wedge z})u(\lambda e.v) &= (\overline{w} \mathbin{\boldsymbol{\wedge}} \overline{z})u(\lambda e.v) && \text{(by (24c))}\\
&= (\lambda e\phi.\overline{w}e(\lambda e'.\overline{z}e'\phi))u(\lambda e.v) && \text{(by (23b))}\\
&\rightarrow_\beta (\lambda \phi.\overline{w}u(\lambda e'.\overline{z}e'\phi))(\lambda e.v) &&\\
&\rightarrow_\beta \overline{w}u(\lambda e'.\overline{z}e'(\lambda e.v)) &&\\
&= \overline{w}u(\lambda e'.z \wedge v) && \text{(by I.H., } e \notin FV(v))\\
&= w \wedge (z \wedge v) && \text{(by I.H., } e' \notin FV(z \wedge v))\\
&\equiv (w \wedge z) \wedge v &&
\end{aligned}$$

- h is an existentially quantified formula of the form $\exists(\lambda x.\mathrm{P}[x])$.

$$\begin{aligned}
\overline{\exists(\lambda x.\mathrm{P}[x])}u(\lambda e.v) &= (\boldsymbol{\exists}(\lambda x.\overline{\mathrm{P}[x]}))u(\lambda e.v) && \text{(by (24d))}\\
&= (\lambda e\phi.\exists(\lambda x.\mathbf{P}[x](x :: e)\phi))u(\lambda e.v) && \text{(by (23c))}\\
&\rightarrow_\beta (\lambda \phi.\exists(\lambda x.\mathbf{P}[x](x :: u)\phi))(\lambda e.v) &&\\
&\rightarrow_\beta \exists(\lambda x.\mathbf{P}[x](x :: u)(\lambda e.v)) &&\\
&= \exists(\lambda x.\mathrm{P}[x] \wedge v) && \text{(by I.H.)}\\
&= \exists(\lambda x.\mathrm{P}[x]) \wedge v && (x \notin FV(v))
\end{aligned}$$

□

Corollary 6.11. *For all propositions t of type o, and for all terms u of type γ, the following folds:*

$$\overline{t}u(\lambda e.\top) \equiv t$$

where e is a variable of type γ.

Proof. Take v equal to $\top$ in Proposition 6.10. Then

$$\bar{t}u(\lambda e.\top) = t \wedge \top \equiv t$$

□

Definition 6.12. A dynamic proposition $\mathbf{t}$ is true in a model $\mathcal{M}$, denoted $\mathcal{M} \models_{dyn} \mathbf{t}$, if and only if $\mathcal{M} \models \mathbf{t}u(\lambda e.\top)$ for every u of type γ.

Theorem 6.13 (Conservation)**.** *A proposition t is true in a model $\mathcal{M}$ if and only if its dynamic variant $\bar{t}$ is true in the same model:*

$$\mathcal{M} \models t \textit{ iff } \mathcal{M} \models_{dyn} \bar{t}$$

Proof. If $\mathcal{M} \models t$, then, by Corollary 6.11, $\mathcal{M} \models \bar{t}u(\lambda e.\top)$. Therefore, by Definition 6.12, $\mathcal{M} \models_{dyn} \bar{t}$.

If $\mathcal{M} \models_{dyn} \bar{t}$, then, by Definition 6.12, $\mathcal{M} \models \bar{t}u(\lambda e.\top)$. Therefore, by Corollary 6.11, $\mathcal{M} \models t$. □

The conservation theorem proves that a static proposition and its dynamic version defined in this section hold in the same models.

6.2 Donkey Sentences

Tables 2 and 3 show respectively static and dynamic (according to G) interpretations for the lexical items in the donkey sentence (5):

(5) *Every farmer who owns a donkey beats it.*

Note that the type of every dynamic term is analogous to its static type. The only difference is that each atomic type of a dynamic term is dynamized according to Definition 6.5. All terms in Table 3, except the interpretation of the pronoun *it*, are dynamized following the rules in Definition 6.7. These rules allow the presentation of dynamic terms in a compact way. They ensure that dynamic terms are structurally analogous to their static counterparts and, therefore, are more intuitive. The dynamic interpretation of *it* is constructed not by directly following the dynamization rules, because *it* is a unconventional lexical item: there is an anaphor to be solved. Therefore, $\widetilde{[\![it]\!]}$[4] contains the selection function sel that takes a context (from which a referent has to be chosen) as an argument.

[4] Here and further on, dynamic interpretations of unconventional lexical items are marked with tilde.

Lexical item	Static type	Static interpretation
farmer	$\iota \to o$	$\mathbf{f}$
donkey	$\iota \to o$	$\mathbf{d}$
owns	$((\iota \to o) \to o) \to ((\iota \to o) \to o) \to o$	$\lambda \mathrm{YX}.\mathrm{X}(\lambda x.\mathrm{Y}(\lambda y.\mathbf{o}xy))$
beats	$((\iota \to o) \to o) \to ((\iota \to o) \to o) \to o$	$\lambda \mathrm{YX}.\mathrm{X}(\lambda x.\mathrm{Y}(\lambda y.\mathbf{b}xy))$
every	$(\iota \to o) \to ((\iota \to o) \to o)$	$\lambda \mathrm{PQ}.\forall(\lambda x.\mathrm{P}x \to \mathrm{Q}x)$
a	$(\iota \to o) \to ((\iota \to o) \to o)$	$\lambda \mathrm{PQ}.\exists(\lambda x.\mathrm{P}x \wedge \mathrm{Q}x)$
who	$(((\iota \to o) \to o) \to o) \to (\iota \to o) \to (\iota \to o)$	$\lambda \mathrm{RQ}x.\mathrm{Q}x \wedge \mathrm{R}(\lambda \mathrm{P}.\mathrm{P}x)$
it	$(\iota \to o) \to o$	$\lambda \mathrm{P}.\mathrm{P}?$

Table 2: Static lexical interpretations.

Lexical item	Dynamic type	Dynamic interpretation in G
farmer	$\bar{\iota} \to \bar{o}$	$\bar{\mathbf{f}}$
donkey	$\bar{\iota} \to \bar{o}$	$\bar{\mathbf{d}}$
owns	$((\bar{\iota} \to \bar{o}) \to \bar{o}) \to ((\bar{\iota} \to \bar{o}) \to \bar{o}) \to \bar{o}$	$\lambda \mathbf{YX}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.\bar{\mathbf{o}}xy))$
beats	$((\bar{\iota} \to \bar{o}) \to \bar{o}) \to ((\bar{\iota} \to \bar{o}) \to \bar{o}) \to \bar{o}$	$\lambda \mathbf{YX}.\mathbf{X}(\lambda x.\mathbf{Y}(\lambda y.\bar{\mathbf{b}}xy))$
every	$(\bar{\iota} \to \bar{o}) \to ((\bar{\iota} \to \bar{o}) \to \bar{o})$	$\lambda \mathbf{PQ}.\forall\!\!\!\forall(\lambda x.\mathbf{P}x \Rightarrow \mathbf{Q}x)$
a	$(\bar{\iota} \to \bar{o}) \to ((\bar{\iota} \to \bar{o}) \to \bar{o})$	$\lambda \mathbf{PQ}.\exists\!\!\!\exists(\lambda x.\mathbf{P}x \mathbin{/\!\!\!\wedge} \mathbf{Q}x)$
who	$(((\bar{\iota} \to \bar{o}) \to \bar{o}) \to \bar{o}) \to (\bar{\iota} \to \bar{o}) \to (\bar{\iota} \to \bar{o})$	$\lambda \mathbf{RQ}x.\mathbf{Q}x \mathbin{/\!\!\!\wedge} \mathbf{R}(\lambda \mathbf{P}.\mathbf{P}x)$
it	$\lambda \mathbf{P}.\lambda e\phi.\mathbf{P}(\mathsf{sel}_{it}e)e\phi$	

Table 3: Dynamic lexical interpretations in framework G.

Taking these dynamic interpretations to compute the meaning of Sentence (5), term (30) β-reduces to term (31), which normalizes to (32):

$$\overline{[\![beats]\!]}\ \widetilde{[\![it]\!]}(\overline{[\![every]\!]}((\overline{[\![who]\!]}(\overline{[\![owns]\!]}(\overline{[\![a]\!]}\ \overline{[\![donkey]\!]})))\overline{[\![farmer]\!]})) \tag{30}$$

$$\to_\beta^* \forall\!\!\!\forall(\lambda x.(\bar{\mathbf{f}}x \mathbin{/\!\!\!\wedge} \exists\!\!\!\exists(\lambda z.\bar{\mathbf{d}}z \mathbin{/\!\!\!\wedge} \bar{\mathbf{o}}xz)) \Rightarrow (\lambda e\phi.\bar{\mathbf{b}}x(\mathsf{sel}_{it}e)e\phi)) \tag{31}$$

$$\to_\beta^* \lambda e\phi.\forall(\lambda x.\mathbf{f}x \to \forall(\lambda z.(\mathbf{d}z \wedge \mathbf{o}xz) \to \mathbf{b}x(\mathsf{sel}_{it}(x :: z :: e)))) \wedge \phi e \tag{32}$$

Resulting term (32) is equivalent to (20) obtained in framework G_0 interpretations. Indeed, framework G is equivalent to de Groote's [4] framework G_0. However, it is advantageous over G_0 due to the compact notations for dynamic terms. These notations significantly systematize the framework and make the interpretations more concise and intuitive. Moreover, the systematic translations of static terms into dynamic terms makes it possible to prove a conservation result 6.13 for G, that guarantees that static and dynamic interpretations are satisfied in the same models.

7 Comparison with related work

DPL, at least as it is presented in [5] does not provide a translation from natural language to the language of DPL. In contrast, this dissertation is focused exactly on providing a translation from natural language to the language of continuation based-dynamic logic. This makes it impossible to make a direct comparison between DPL and GLχ (as well as its predecessors G and GL). Nevertheless, it is still possible to compare the frameworks by defining certain translations between their different levels. Since, among G, GL and GLχ, the closest to DPL w.r.t. expressive power is G, translations between G and DPL are defined subsequently.

Consider Figure 4. The missing translation h_{DPL} of natural language into the language of DPL is represented by the dashed arrow. The translation h_G of natural language into the language G_L of continuation-based dynamic logic is represented by the green arrow.

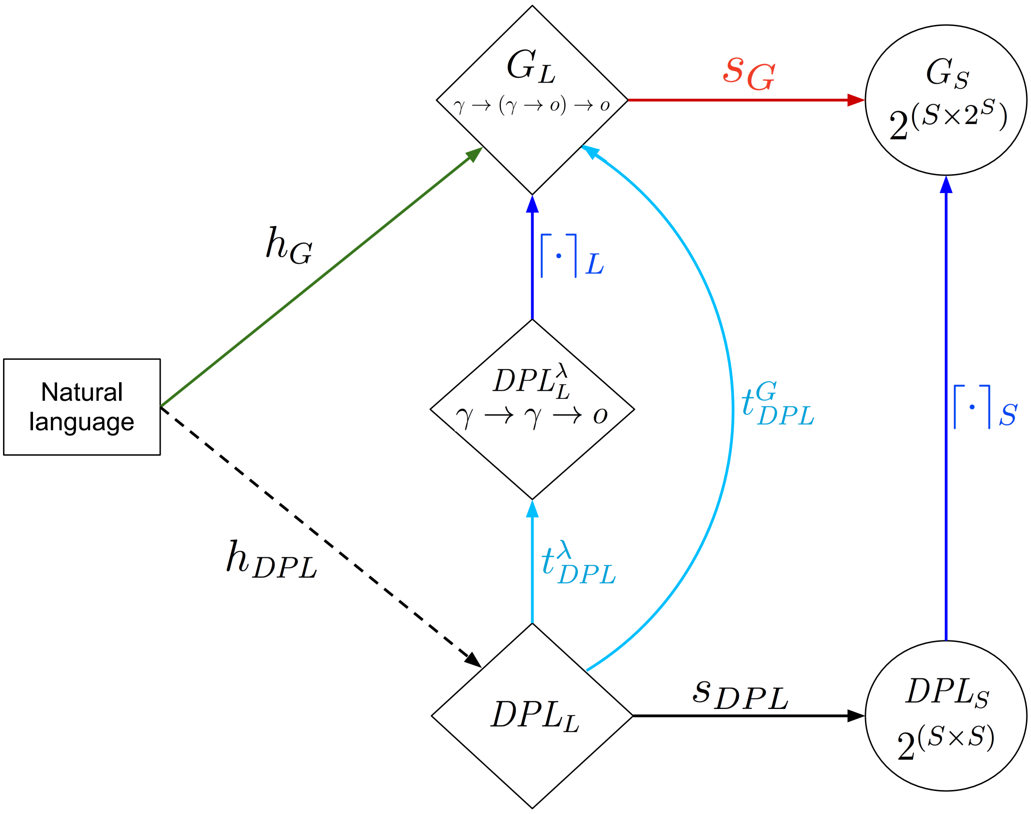

Figure 4: Comparison between DPL and G.

What Groenendijk and Stokhof [5] define is a translation s_{DPL} of the language of DPL into binary relations on states DPL_S. This translation is illustrated with the black solid arrow.

There are two possible ways to compare DPL and G. One possibility is to define a translation s_G (red arrow) of the language of G into relations on states G_S and to compare G_S with DPL_S. Another possibility is to define a translation t^{λ}_{DPL} (shorter light blue arrow) of the language DPL_L of DPL into a new language DPL^{λ}_L and to compare DPL^{λ}_L with the language G_L of G. These two comparisons are presented in more details below.

7.0.1 Comparison 1

DPL interprets propositions as binary relations on states and its semantics is defined as the following translation from DPL_L to DPL_S (solid black arrow):

Definition 7.1. [Translation s_{DPL}] Let A, B and P be terms in DPL_L of type o. Let h, g and k be assignments of values to variables (a.k.a. states). Translation $s_{DPL} : DPL_L \to DPL_S$ is defined as follows:

$$\begin{aligned}
(A)_{s_{DPL}} &= \{\langle g, h\rangle \mid h = g \wedge h \models A\} \\
(\neg P)_{s_{DPL}} &= \{\langle g, h\rangle \mid h = g \wedge \neg(\exists k.\ \langle h, k\rangle \in (P)_{s_{DPL}})\} \\
(A \wedge B)_{s_{DPL}} &= \{\langle g, h\rangle \mid \exists k.\ \langle g, k\rangle \in (A)_{s_{DPL}} \wedge \langle k, h\rangle \in (B)_{s_{DPL}}\} \\
(\exists i.P)_{s_{DPL}} &= \{\langle g, h\rangle \mid \exists k.\ k[i]g \wedge \langle k, h\rangle \in (P)_{s_{DPL}}\}
\end{aligned}$$

Thus, for every proposition P, $(P)_{s_{DPL}}$ is in $2^{(S \times S)}$.
A similar translation s_G (red arrow) of propositions in G_L into relations defined on states can be defined:

Definition 7.2. [Translation s_G] Let S, the set of states, be the semantic domain that interprets γ. The translation $s_G : G_L \to G_S$ is defined as follows:

$$\begin{aligned}
(A)_{s_G} &= \{\langle g, H\rangle \mid g \in H \wedge g \models A\} \\
(\neg\!\shortmid P)_{s_G} &= \{\langle g, H\rangle \mid g \in H \wedge \langle g, S\rangle \notin (P)_{s_G}\} \\
(A \mathbin{/\!\!\!\wedge} B)_{s_G} &= \{\langle g, H\rangle \mid \langle g, \{h \mid \langle h, H\rangle \in (B)_{s_G}\}\rangle \in (A)_{s_G}\} \\
(\exists\!\shortmid_i x.P)_{s_G} &= \{\langle g, H\rangle \mid \exists x.\langle (x, i) :: g, H\rangle \in (P)_{s_G}\}
\end{aligned}$$

Note that s_G interprets propositions not just as relations between states, but as relations betweens states and sets of states: for each proposition P, $(P)_{s_G}$ is in $2^{(S \times 2^S)}$. Therefore, s_G is richer than s_{DPL} and avoids existentially quantified states and equality relations on states by using set inclusion. Moreover, there exists a canonical embedding (larger blue arrow) of DPL_S into G_S:

Definition 7.3. [Embedding of DPL_S into G_S] Let R be set of pair of states (in $2^{(S \times S)}$) in DPL_S, the embedding $\lceil \cdot \rceil_S$ of DPL_S into G_S is as follows:

$$\lceil R \rceil_S \doteq \{\langle g, H\rangle \mid \exists h.\ h \in H \wedge \langle g, h\rangle \in R\}$$

$\lceil R \rceil_S$ is in $2^{(S \times 2^S)}$.

Note that there is a correspondence between types constructed from γ and o and sets constructed from S and the cross product powerset operations:

$$\begin{aligned}[\gamma] &\approx S\\ [\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \to o] &\approx 2^{[\alpha_1] \times [\alpha_2] \times \cdots \times [\alpha_n]}\end{aligned}$$

Consequently, the type $(\gamma \to (\gamma \to o) \to o)$ corresponds to the set $2^{(S \times 2^S)}$.

The following translation t^G_{DPL} (longer light blue arrow) of DPL_L to G_L can be defined:

Definition 7.4. [Translation t^G_{DPL}] Let A, B and P be propositions. The translation $t^G_{DPL} : DPL_L \to G_L$ is defined as follows:

$$\begin{aligned}(A)_{t^G_{DPL}} &= A\\ (\neg P)_{t^G_{DPL}} &= \boldsymbol{\neg}\!\shortmid (P)_{t^G_{DPL}}\\ (A \wedge B)_{t^G_{DPL}} &= (A)_{t^G_{DPL}} \mathbin{/\!\!\!\boldsymbol{\wedge}} (B)_{t^G_{DPL}}\\ (\exists i.P)_{t^G_{DPL}} &= \boldsymbol{\exists}\!\shortmid_i x.(P)_{t^G_{DPL}}\end{aligned}$$

Theorem 7.5. *Let P be a proposition. Then the following equivalence holds:*

$$\lceil (P)_{s_{DPL}} \rceil_S = ((P)_{t^G_{DPL}})_{s_G}$$

by postulating

$$g[i]h \quad \textit{iff} \quad \exists x.(h = (x, i) :: g)$$

Proof. The proof is by induction on the structure of P. □

By Theorem 7.5, transforming DPL_L to G_L and interpreting the result in G_S according to s_G is equivalent to DPL's original semantics.

7.0.2 Comparison 2

Each proposition A in DPL_L can be translated into a term of type $(\gamma \to \gamma \to o)$ in DPL^λ_L in a way that reflects the semantics of DPL. The translation is given by the following definition:

Definition 7.6. [Translation t^{λ}_{DPL}] Let T of type o be an atomic proposition, P, A, and B of type o be propositions, g, h and k be variables of type γ. The translation $t^{\lambda}_{DPL} : DPL_L \to DPL^{\lambda}_L$ is defined as follows:

$$(T)_{t^{\lambda}_{DPL}} \doteq \lambda gh.\ h = g \wedge T \tag{33a}$$

$$(\neg P)_{t^{\lambda}_{DPL}} \doteq \lambda gh.\ h = g \wedge \neg(\exists k.(P)_{t^{\lambda}_{DPL}} hk) \tag{33b}$$

$$(A \wedge B)_{t^{\lambda}_{DPL}} \doteq \lambda gh.\ \exists k.\ (A)_{t^{\lambda}_{DPL}} gk \wedge (B)_{t^{\lambda}_{DPL}} kh \tag{33c}$$

$$(\exists x.P)_{t^{\lambda}_{DPL}} \doteq \lambda gh.\ \exists k.\ k[x]g \wedge (P)_{t^{\lambda}_{DPL}} kh \tag{33d}$$

The resulting language DPL^{λ}_L can be compared with G_L.

Note that DPL^{λ}_L has the existential quantifier in the definitions of negation (33b) and conjunction (33c) and a subterm with equality in the definitions of the atomic proposition (33a) and negation (33b). DPL uses these existential quantifications and equalities in order to constraint the pairs of states that represent the final interpretation. Because of that, the final interpretation in DPL^{λ}_L would have chains of equalities of the form $\cdots \wedge g = g' \wedge \cdots \wedge g' = g'' \wedge \ldots$ and some deductive reasoner would be required for establishing the final interpretation of a sentence. In G, however, there is no need for this kind of equalities and existential quantification. All necessary restrictions that natural language poses on a model are interpreted using β-reduction only.

A canonical embedding (smaller blue arrow) of DPL^{λ}_L into G_L can be defined:

Definition 7.7. [Embedding of DPL^{λ}_L into G_L] Let A be a term of type $(\gamma \to \gamma \to o)$ in DPL^{λ}_L, the embedding $\lceil \cdot \rceil_L$ of DPL^{λ}_L into G_L is as follows:

$$\lceil A \rceil_L \doteq \lambda e\phi.\exists e'.\ \phi\ e' \wedge A\ e\ e'$$

$\lceil A \rceil_L$ is of type $(\gamma \to (\gamma \to o) \to o)$.

Theorem 7.8. *Let A, B and P be terms of type o in DPL_L. Then the following logical equivalences hold:*

$$\lceil (A)_{t^{\lambda}_{DPL}} \rceil_L \equiv \overline{A}$$

$$\lceil (\neg P)_{t^{\lambda}_{DPL}} \rceil_L \equiv \neg\!\shortparallel \lceil (P)_{t^{\lambda}_{DPL}} \rceil_L$$

$$\lceil (A \wedge B)_{t^{\lambda}_{DPL}} \rceil_L \equiv \lceil (A)_{t^{\lambda}_{DPL}} \rceil_L \mathbin{/\!\!\!\wedge} \lceil (B)_{t^{\lambda}_{DPL}} \rceil_L$$

Proof. The proof is by induction on the structure of terms. □

An analogous logical equivalence does not hold, however, for the existential quantifier: $\lceil (\exists x.P)_{t^{\lambda}_{DPL}} \rceil_L$ reduces to (34) and $\boldsymbol{\exists\!\!\mid} \lceil (P)_{t^{\lambda}_{DPL}} \rceil_L$ reduces to (35):

$$\lceil (\exists x.P)_{t^{\lambda}_{DPL}} \rceil_L = \lambda e \phi.\ \exists e'.\ \phi\ e' \wedge \exists k.\ k[x]e \wedge \lceil (P)_{t^{\lambda}_{DPL}} \rceil_L\ k\ e' \tag{34}$$

$$\boldsymbol{\exists\!\!\mid} \lceil (P)_{t^{\lambda}_{DPL}} \rceil_L = \lambda e \phi.\ \exists e'.\ \phi\ e' \wedge \exists x.\ \lceil (P)_{t^{\lambda}_{DPL}} \rceil_L\ (x :: e)\ e' \tag{35}$$

Nevertheless, terms (34) and (35) are similar from the point of view of constraining their interpretation in a model. Thus, the constraint in (34) that k is interpreted as e except for the values of x is expressed in (35) simply by explicit update of the context e with x. Note, moreover, that x is a free variable in (34), whereas (35) is a closed term. Therefore, interpretation (35) is advantageous over (34).

8 The higher-order case

$\lambda p.\, \lambda q.\, \forall\!\!\!\forall x.\, (p\, x) \Rightarrow (q\, x)$

9 Conclusions

References

[1] H. Barendregt. *The lambda calculus, its syntax and semantics.* North-Holland, revised edition, 1984.

[2] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II. Oxford University Press, 1992.

[3] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[4] P. de Groote. Towards a montagovian account of dynamics. In *Semantics and Linguistic Theory XVI*, 2006.

[5] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14(1):39–100, 1991.

[6] J. R. Hindley and J. P. Seldin. *Lambda-Calculus and Combinators, an Introduction.* Cambridge University Press, 2008.