Context and the Composition of Meaning

Jan van Eijck (jve@cwi.nl)
CWI and ILLC, Amsterdam, Uil-OTS, Utrecht

Abstract. Key ingredients in discourse meaning are reference markers: objects in the formal representation that the discourse is about. It is well-known that reference markers are not like first order variables. Indeed, it is the received view that reference markers are like the variables in imperative programming languages. However, in a computational semantics of discourse that treats reference markers as 'dynamically bound' variables, every noun phrase will get linked to a dynamic variable, so it will give rise to a marker index. Where do these indices come from? How do we handle them when combining (or 'merging') pieces of discourse?

We will argue that reference markers are better treated as indices into context, and we will present a theory of context and context extension based on this view. In context semantics, noun phrases do not come with fixed indices, so the merge problem does not arise. This solves a vexing issue with coordination that causes trouble for all current versions of compositional discourse representation theory.

1. Introduction

We will start by briefly reviewing the discussion of DRT and compositionality, by presenting the standard view on how DRSs should be merged. We show that this view leads to a puzzle with coordination. This unsolved problem motivates the switch to a more sophisticated theory of context and context extension than is present in current rational reconstructions of DRT. We show that under this new reconstruction the need for merge has disappeared, and the coordination puzzle can be solved. Next we briefly turn to issues of salience and salience update and the use of salience in pronoun reference resolution, and list our conclusions.

2. Linking Pronouns: the Standard Dynamic Account

The by now standard dynamic account of the way pronouns get linked to their antecedents has the following two kinds of basic ingredients:

- contexts,
- constraints on contexts.

A DRT-style representation (Kam81; KR93) for a piece of text uses these ingredients as follows:

context
constraints on context

Information conveyed by a piece of text grows, and this growth of information is reflected in a representation update:

context		new context
constraints on context	⟶ update ⟶	new constraints on context

In DRT, the details of this scheme are filled out as follows. An initial DRS represents context and constraints on context for 'a man entered':

x
Mx Ex

This initial representation changes through successive updates, as follows:

x
Mx Ex

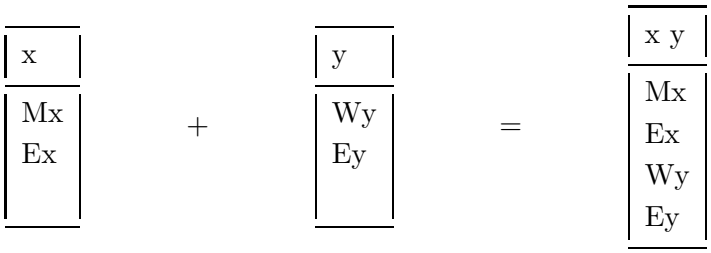
→ 'A woman entered' →

x y
Mx Ex Wy Ey

→ 'He smiled at her' →

x y
Mx Ex Wy Ey Sxy

Assume, now, that sentences to be added to an existing representation have a representation of their own. Then we are faced with the problem of how an initial piece representation has to be merged with a new piece of representation to effect an information update.



The example illustrates how we get in trouble in cases where the representation of 'a man' and of 'a woman' employ the same variable. This merge problem in the presence of clashing variables does not occur in (Kam81) and in the DRT textbook (KR93), for these presentations of DRT work with a top-down DRS construction algorithm, where merging of DRSs is avoided because a new DRS is always constructed in the context of an already existing DRS. The Classic DRT construction algorithms always parses new sentences in the context of an existing representation structure.

3. Merging DRSs

When dynamic semantics for NL first was proposed in (Kam81) and (Hei82), the approach invoked strong opposition from the followers of Montague (Mon73). Rational reconstructions to restore compositionality were announced in (GS91) and carried out in (GS90; Chi92; Jan98; Mus95; Mus96; Mus94; Eij97; EK97; KKP96; Kus00; Bek00), among others. All of these reconstructions are based in some way or other on DPL (GS91), and they all inherit the main flaw of this approach: the destructive assignment problem, i.e., the fact that assigning a value to a variable x destroys the old value of x.

Interestingly, DRT itself did not suffer from this problem: the discourse representation construction algorithms of (Kam81) and (KR93) are stated in terms of functions with finite domains, and carefully talk about 'taking a fresh discourse referent' to extend the domain of a verifying function, for each new NP to be processed.

The merge problem arises in carrying out a Montagovian or Fregean programme of natural language analysis in a setting that takes context and context change into account. Compositional versions of DRT presuppose a definition of 'merge'. In compositional DRT, a lexical entry for the determiner 'a' might look like this.

$$\lambda PQ.\ \boxed{\begin{array}{|c|}\hline x \\ \hline \\ \hline\end{array}} \bullet Px \bullet Qx.$$

And a lexical entry for the determiner 'every' might look like this:

$$\lambda PQ.\ \boxed{\begin{array}{|c|}\hline \\ \hline \boxed{\begin{array}{|c|}\hline x \\ \hline \\ \hline\end{array}} \bullet Px \Rightarrow Qx \\ \hline\end{array}}$$

Two obvious questions: (i) How should the reference marker x be picked? (ii) how should $\bullet$ be defined?

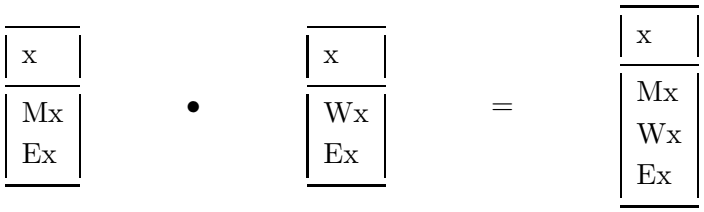
The classical DRT view on these questions can be found in (Zee89), where the following compositional DRS definition is proposed.

- Basic DRSs: $(\emptyset, \emptyset)$, $(\emptyset, \{Pr_0 \cdots r_{n-1}\})$, $(\emptyset, \{\bot\})$, $(\{x\}, \emptyset)$.
- Merger of DRSs: $\delta \bullet \delta' := (V_\delta \cup V_{\delta'}, C_\delta \cup C_{\delta'})$.
- Implication of DRSs: $\delta \rightarrow \delta' := (\emptyset, \{\delta \Rightarrow \delta'\})$.

The semantics that goes with this, with respect to some First Order Model $\mathcal{M} = (M, I)$, is given by:

- $[\![(\emptyset, \emptyset)]\!] = (\emptyset, M^U)$,
- $[\![(\emptyset, \{Pr_0 \cdots r_{n-1}\})]\!] = (\emptyset, \{f \in M^U \mid \mathcal{M} \models_f Pr_0 \cdots r_{n-1}\})$,
- $[\![(\emptyset, \{\bot\})]\!] = (\emptyset, \emptyset)$,
- $[\![(\{x\}, \emptyset)]\!] = (\{x\}, M^U)$,
- $[\![\delta \bullet \delta']\!] := [\![\delta]\!] \oplus [\![\delta']\!]$, where $(X, F) \oplus (Y, G) := (X \cup Y, F \cap G)$,
- $[\![\delta \Rightarrow \delta']\!] := [\![\delta]\!] \rightarrow [\![\delta']\!]$, where
  $(X, F) \rightarrow (Y, G) :=$
  $(\emptyset, \{h \in M^U \mid \forall f \in F(\text{if } h[X]f \text{ then } \exists g \in G \text{ with } f[Y]g)\})$.

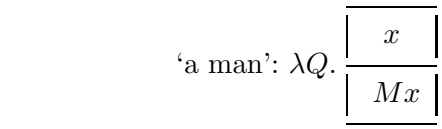
These definitions suggest that merging proceeds as follows in our example case.

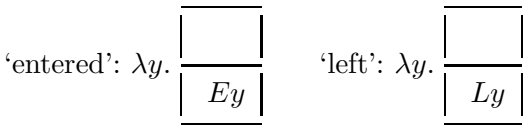


This is certainly *not* the outcome one would like. So can such variable clashes be avoided? Or can they get repaired (e.g., by means of an alternative merge operation)? To see that these are vexing problems, consider the following puzzle with coordination (cf. also (BB99)):

A man entered and a man left. (1)

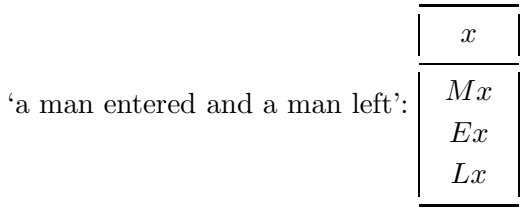
A treatment of this example in compositional DRT will be based on the following ingredients:





'and': $\lambda pq.p \bullet q$.

Composing these ingredients with functional applications, using the above definition of •, gives the following (wrong) result:



4. Attempts at a Solution

One way to solve the merge problem is by a change in the representation of variables. In (Ver95), variables get replaced by so-called referent systems. The basic idea of this modification is to distinguish between the following two aspects of a variable:

– variable name

– variable address or memory slot

Referent systems are like pointer structures in imperative programming, with the slight twist that input under a name gets distinguished from output under a name.

$$\begin{array}{lll} x \rightarrow & [\cdot] & \\ & [\cdot] & \rightarrow x \\ y \rightarrow & [\cdot] & \rightarrow y \\ z \rightarrow & [\cdot] & \\ & [\cdot] & \rightarrow u \end{array}$$

Merging referent systems is done by tracing variables from input to output.

$$\left(\begin{array}{lll} x \rightarrow & [\cdot] & \\ & [\cdot] & \rightarrow x \\ y \rightarrow & [\cdot] & \rightarrow y \\ z \rightarrow & [\cdot] & \\ & [\cdot] & \rightarrow u \end{array}\right) \bullet \left(\begin{array}{lll} & [\cdot] & \\ x \rightarrow & [\cdot] & \rightarrow x \\ y \rightarrow & [\cdot] & \rightarrow y \\ & [\cdot] & \\ & [\cdot] & \end{array}\right) = \begin{array}{lll} x \rightarrow & [\cdot] & \\ & [\cdot] & \rightarrow x \\ y \rightarrow & [\cdot] & \rightarrow y \\ z \rightarrow & [\cdot] & \\ & [\cdot] & \end{array}$$

This way of merging can be applied to merging 'referent system' DRSs, as follows. In the following example, x is first linked to the referent m, next to the referent n.

$$\begin{array}{|l|} \hline [m] \rightarrow x \\ \hline \text{Mx} \\ \text{Ex} \\ \hline \end{array} \quad \bullet \quad \begin{array}{|l|} \hline [n] \rightarrow x \\ \hline \text{Wx} \\ \text{Ex} \\ \hline \end{array} \quad = \quad \begin{array}{|ll|} \hline [m] & \\ [n] & \rightarrow x \\ \hline \text{Mx'} & \\ \text{Ex'} & \\ \text{Wx} & \\ \text{Ex} & \\ \hline \end{array}$$

The reference to m gets destroyed by the merge. To indicate that two instances of the variable name x point to different data, a renaming of the instances that refer to m is mandatory. The example shows that replacing variables by referent systems does not in itself solve the merge problem. In the example the referent m becomes unaccessible: there is no variable name attached to it anymore.

A genuine solution to the merge problem in dynamic semantics is provided by sequence semantics, also proposed by Vermeulen, in (Ver93). In sequence semantics, a variable x gets interpreted as a stack

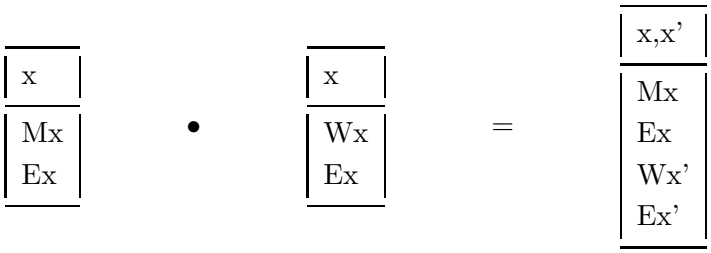
$[d_0, \ldots, d_{n-1}]$. Each new introduction for x extends the stack by means of an operation

$$[d_0, \ldots, d_{n-1}] \mapsto [d_0, \ldots, d_{n-1}, d_n]$$

The key idea of sequence semantics can be restated as follows: assume that each variable x comes with a sequence x', x'', x''' ... of extensions.

x	x'	x''	x'''	x''''	$\cdots$
d_0	d_1	d_2	d_3	d_4	$\cdots$

New introductions are never destructive, for they refer to an extension:



Note that both x and x' remain accessible.

5. Abstraction over Context

In this section we will propose an account of contexts, context extension and abstraction over context based on what can be viewed as a combination and simplification of referent systems and sequence semantics. The main ingredients of our accounts are (i) passing contexts as parameters and (ii) using types for constraining the indices used for pointing into contexts.

Our account can also be viewed as a simplification of (Dek96) where a rational reconstruction of DRT is given in terms of a system of predicate logic extended with stack pointers, together with an encoding in polymorphic relational type theory. The basic difference is that while Dekker starts out from an version of predicate logic containing both variables and stack pointers (Dek94), in our set-up all the stack manipulation tools that we need get introduced by the type theory.

$$\lambda \boxed{\text{context}} \cdot \left(\boxed{\text{context}} + \begin{array}{|l|}\hline \text{context extension} \\ \hline \text{constraints} \\ \\ \hline \end{array} \right)$$

We will take contexts to be essentially lists of reference markers that encode discourse information and allow us to keep track of topics of discourse.

In discourse processing, the *order* in which discourse topics are introduced is crucial. Topics mentioned most recently are (other things being equal) more readily accessible for reference resolution. Context also provides additional information:

- Gender and number information.
- actor focus: agent of the sentence.
- discourse focus: 'what is talked about' in the sentence.

In this paper, we will take this additional information as secondary.

So how does one abstract over context? By representing a context as a stack of items, and by handling these stacks as suggested in Incremental Dynamics (Eij01).

$$\boxed{c_0 \mid c_1 \mid c_2 \mid c_3 \mid c_4 \mid \cdots}$$

Now existential quantification can be modelled as context extension:

$$\boxed{c_0 \mid c_1 \mid c_2 \mid c_3} + d = \boxed{c_0 \mid c_1 \mid c_2 \mid c_3 \mid d}$$

Indices are used to refer to context elements:

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 0 & 1 & 2 & 3 & 4 & \cdots & n-1 & n \\ \hline c_0 & c_1 & c_2 & c_3 & c_4 & \cdots & c_{n-1} & d \\ \hline \end{array}$$

Given this representation of context, we can replace merge by context composition. It is convenient to introduce some type abbreviations for that. We use $[e]$ for the type of contexts, $[e] \to [e] \to t$ for the type of context transitions (characteristic functions of binary relations on the type of contexts). Use $a :: \alpha$ for "a is of type α". Assume $c, c' :: [e]$ and $x :: e$. Let $c\hat{}x$ be the result of extending context c with element

x. This assumes $(\hat{})$ :: $[e] \to e \to [e]$. Note that we assume that type arrows associate to the right, so that $[e] \to e \to [e]$ gets interpreted as $[e] \to (e \to [e])$. Now define context extension as follows:

$$\exists \;:=\; \lambda cc'.\exists x(c\hat{}x = c')$$

This definition of context extension essentially uses polymorphic type theory (Hin97; Mil78). The type polymorphism is crucial, for it is implicit in $c\hat{}x = c'$ that the lengths of the contexts c and c' match. In other words, the type of contexts is polymorphic, for a context may have any finite length. The type of $\exists$ is given by $\exists :: [e] \to [e] \to t$. This is a polymorphic type, for $\exists$ relates contexts of length n to contexts of length $n+1$, for any n.

The type polymorphism can be made explicit by writing the type of a context c as $[e]_i$, with i a type variable indicating the context length, but for convenience we omit these indices. What we do need, however, is a means of referring to the length of a context c in a generic way. We will use $|c|$ for the length of context c.

Let T be an abbreviation of $[e] \to [e] \to t$, i.e., let T be the type of context transitions. Then $\exists :: T$.

Composition of context transitions is also easily defined in polymorphic type theory. Assume $\phi, \psi :: T$, let $c, c' :: [e]$ and define $(\;;\;)$ as follows:

$$\phi \;;\; \psi \;:=\; \lambda cc'.\exists c''(\phi cc'' \wedge \psi c''c')$$

Now the type of $\;;\;$ is given by:

$$(\;;\;) :: T \to T \to T.$$

What this says is that $(\;;\;)$ takes two context transitions and produces another context transition.

If a context c has length n, i.e., if the context elements run from c_0 to c_{n-1}, indices referring to context elements should run from 0 to $n-1$:

0	1	2	3	4	$\cdots$	$n-1$
c_0	c_1	c_2	c_3	c_4	$\cdots$	c_{n-1}

This can be achieved in a natural way by using natural numbers as index types. Recall that under the Von Neumann encoding of natural numbers it holds that $n = \{0, \ldots, n-1\}$. Thus, an index of type n is an index ranging over $\{0, \ldots, n-1\}$, and this is precisely what is needed for indexing into a context of length n. If c is a context of length n,

i.e., $c :: [e]_n$, then $i :: n$ indicates that i is of the appropriate type for indexing into c. If we assume that $\lambda ci.c[i] :: [e]_n \to n \to e$, where n is the type variable indicating context length, then we get $(\lambda i.c[i]) :: n \to e$. This illustrates how polymorphic type assigment can enforce the index to be of the type that fits the size of the context.

It is convenient to gloss over these details by using ι as a type for context indices, and assuming that indices fit the sizes of contexts throughout. More specifically, in types of the form $\iota \to [e] \to \alpha$ we will tacitly assume that the index fits the size of the *initial* context $[e]$. Thus, $\iota \to [e] \to [e] \to t$ is really a type scheme rather than a type, although the type polymorphism remains hidden from view. Since $\iota \to [e] \to [e] \to t$ generalizes over the size of the context, it is shorthand for the types $0 \to [e]_0 \to [e] \to t$, $1 \to [e]_1 \to [e] \to t$, $2 \to [e]_2 \to [e] \to t$, and so on.

In what follows, we will employ variables P, Q of type $\iota \to [e] \to [e] \to t$, variables i, j, j' of type ι, and variables c, c' of type $[e]$.

We call a function of type $\iota \to T$ an indexed context transition. In the treatment of the indefinite determiner, we assume that the determiner combines with two indexed context transitions and produces a context transition. Assume that P and Q are indexed context transitions, i.e., assume $P, Q :: \iota \to T$. Then the new version of the lexical entry for determiner 'a' runs like this:

$$\lambda PQc.(\exists \mathbf{\ ;\ } Pi \mathbf{\ ;\ } Qi)c \text{ where } i = |c|.$$

The "$\phi(i)$ where $i = |c|$" is a piece of syntactic sugar that can be removed, at the penalty of unreadability, by generic substitution of $|c|$ for i in ϕ.

The type of this determiner entry is $(\iota \to T) \to (\iota \to T) \to T$. In a phrase

$$[_{\mathrm{S}}[_{\mathrm{NP}}[_{\mathrm{DET}}\ a\ ][_{\mathrm{CN}}\ A\ ]][_{\mathrm{VP}}\ B\ ]],$$

the common noun A and the verb phrase B get interpreted as indexed context transitions, to be combined by the interpretation of the determiner into a new context transition that interprets the sentence. This illustrates the type lift * involved in incremental dynamics.

$$\begin{array}{lcl} t^* & = & T \\ e^* & = & \iota \\ (\alpha \to \beta)^* & = & \alpha^* \to \beta^* \end{array}$$

In fact, the shift from e to ι will always take place "in context". To make this explicit, one might wish to define the type lift for a type language built from the primitive t with the operations $e \to \alpha$ and $\alpha \to \beta$.

To define universal quantification, we first need a definition of context negation:

$$\neg\!\!\!\neg\phi \;:=\; \lambda cc'.(c = c' \wedge \neg\exists c''\phi cc'')$$

If ϕ is a context transition, then $\neg\!\!\!\neg\phi$ is a new context transition, and $\neg\!\!\!\neg\phi$ expresses the familiar negation as test from dynamic semantics: input context is equal to output context, and the test succeeds for a given input c iff there are no ϕ-transitions from c. Note that $\neg\!\!\!\neg :: T \to T$.

Dynamic implication $\Rightarrow$ can now be defined in terms of $\neg\!\!\!\neg$ and $\mathbf{;}$, as follows:

$$\phi \Rightarrow \psi \;:=\; \neg\!\!\!\neg(\phi \;\mathbf{;}\; \neg\!\!\!\neg\psi)$$

The type of $\Rightarrow$ is given by $(\Rightarrow) :: T \to T \to T$. In terms of this, we phrase the lexical entry for the determiner 'every' as follows:

$$\lambda PQc.((\exists \;\mathbf{;}\; Pi) \Rightarrow Qi))c \text{ where } i = |c|.$$

This has the correct type $(\iota \to T) \to (\iota \to T) \to T$, and it assigns the dynamic meaning familiar from DRT and DPL.

Predicate Lifting
We assume that the lexical meanings of CNs, VPs are given to us as one place predicates (type $e \to t$) and those of TVs as two place predicates (type $e \to e \to t$). We therefore define blow-up operations for lifting one-placed and two-placed predicates to the dynamic level. Assume A to be an expression of type $e \to t$, and B an expression of type $e \to e \to t$; we use c, c' as variables of type $[e]$, and j, j' as variables of type ι, and we employ postfix notation for the lifting operations:

$$\begin{aligned} A^{\circ} \;&:=\; \lambda jcc'.(c = c' \wedge Ac[j]) \\ B^{\bullet} \;&:=\; \lambda jj'cc'.(c = c' \wedge Bc[j]c[j']) \end{aligned}$$

Note that $(^{\circ}) :: (e \to t) \to \iota \to T$ and $(^{\bullet}) :: (e \to e \to t) \to \iota \to \iota \to T$. The operation $\circ$ lifts a one-place predicate to a function of type $\iota \to T$, the operation $\bullet$ lits a two-place predicate to a function of type $\iota \to \iota \to T$. These type lifts are in accordance with the type lifting operation $*$, for $A^{\circ} :: (e \to t)^*$ and $B^{\bullet} :: (e \to e \to t)^*$.

It is instructive to compare our typing $\iota \to T$ for a common noun like *man* with the typing in (Dek96), where the translation of *man* has the following type:

$$e \to \langle e_1, \ldots, e_n\rangle \to \langle e_1, \ldots, e_n\rangle,$$

where $\langle e_1, \ldots, e_n\rangle$ is the type of n-ary relations in the relational type theory of (Ore59). This makes a common noun into a function for mapping individuals to relation transformers. This detour via polymorphic

relational type theory turns out to be unnecessary. Passing contexts as parameters, together with the use of typed indices, allows us to remain within simple polymorphic type theory. Implementation as a functional program in a functional programming language based on polymorphic type theory is therefore straightforward.

6. Multidimensional Grammar

Following (Cre73), (Oeh94) and (Mus02), we use a multi-dimensional set-up for our grammar formalism. We will use signs consisting of three components: a syntactic term, a semantic term, and a sign type.

Syntactic Terms
For syntactic terms, we take closed linear lambda terms over word lists, with the conventions of list notation and list concatenation adopted from functional programming languages like Haskell (HT). A lambda term is *linear* if each lambda operator binds exactly one variable. We take S as the type of a word list.

Word list concatenation is given by $+\!\!+ :: S \to S \to S$. We abbreviate $\lambda x \lambda y$ to λxy, and so on. We use x, y as variables over word lists, i.e., $x, y :: S$, and X as a variable over functions from word lists to word lists, i.e., $X :: S \to S$. Here are some example syntactic terms, with their syntactic types.

$$
\begin{array}{l}
[\text{john}] :: S \\
[\text{john}, \text{loves}, \text{mary}] :: S \\
\lambda x.[\text{loves}] +\!\!+ x :: S \to S \\
\lambda xy.y +\!\!+ [\text{loves}] +\!\!+ x :: S \to S \to S \\
\lambda x.x +\!\!+ [\text{loves}, \text{mary}] :: S \to S \\
\lambda X.X[\text{loves}, \text{mary}] :: (S \to S) \to S
\end{array}
$$

Semantic Terms
Semantic terms are expressions from polymorphic typed logic, over basic types e and t, with the conventions for the use of $[e]$ and ι that were explained above.

Sign Types and Sign Type Reduction
The language of sign types is given by:

$$\text{SignType} \;:=\; \text{Ref} \mid \text{Noun} \mid \text{Sent} \mid \text{SignType} \to \text{SignType}$$

The functions $^{\diamondsuit}$,$^{\heartsuit}$ reduce a sign type Φ to a pair consisting of a syntactic type $\Phi^{\diamondsuit}$ and a semantic type $\Phi^{\heartsuit}$.

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{Ref}^{\diamondsuit} & := & S & \mathrm{Ref}^{\heartsuit} & := & \iota \\ \mathrm{Noun}^{\diamondsuit} & := & S & \mathrm{Noun}^{\heartsuit} & := & \iota \to T \\ \mathrm{Sent}^{\diamondsuit} & := & S & \mathrm{Sent}^{\heartsuit} & := & T \\ (A \to B)^{\diamondsuit} & := & A^{\diamondsuit} \to B^{\diamondsuit} & (A \to B)^{\heartsuit} & := & A^{\heartsuit} \to B^{\heartsuit} \end{array}$$

Note that the links $\mathrm{Ref}^{\heartsuit} = \iota$, $\mathrm{Noun}^{\heartsuit} = \iota \to T$ and $\mathrm{Sent}^{\heartsuit} = T$ constitute the basic type assignment of context logic.

Signs
Signs are triples consisting of a syntactic term $\mathcal{S}$, a semantic term $\mathcal{T}$ and a sign type Φ, under the constraint that $\mathcal{S} :: \Phi^{\diamondsuit}$ and $\mathcal{T} :: \Phi^{\heartsuit}$.

Assume that *Woman* is a constant of type $e \to t$. Then the following triple is a sign:

$$([\mathrm{woman}], \mathit{Woman}^{\circ}, \mathrm{Noun}).$$

To see that this is a sign, note that the typing constraints are satisfied, for $\mathrm{Noun}^{\diamondsuit} = S$, which matches $[\mathrm{woman}] :: S$, and $\mathrm{Noun}^{\heartsuit} = \iota \to T$, which matches $\mathit{Woman}^{\circ} :: \iota \to T$.

Assume that *Love* is a constant of type $e \to e \to t$. Then the following triple is a sign:

$$(\lambda xy.y{+\!\!+}[\mathrm{loves}]{+\!\!+}x, \mathit{Love}^{\bullet}, \mathrm{Ref} \to \mathrm{Ref} \to \mathrm{Sent})$$

The typing constraints are satisfied, for $\lambda xy.y{+\!\!+}[\mathrm{loves}]{+\!\!+}x :: S \to S \to S$, $(\mathrm{Ref} \to \mathrm{Ref} \to \mathrm{Sent})^{\diamondsuit} = S \to S \to S$, $\mathit{Love}^{\bullet} :: \iota \to \iota \to T$, $(\mathrm{Ref} \to \mathrm{Ref} \to \mathrm{Sent})^{\heartsuit} = \iota \to \iota \to T$.

Here are some further examples of signs. The check that these are signs is left to the reader. (Assume *Smile* is a constant of type $e \to t$.)

- $\lambda xX.X([\mathrm{every}]{+\!\!+}x)$,
  $\lambda PQc.((\exists\ ;\ Pi) \Rightarrow Qi))c$ where $i = |c|$,
  $\mathrm{Noun} \to (\mathrm{Ref} \to \mathrm{Sent}) \to \mathrm{Sent}$.

- $\lambda X.X[\mathrm{every}, \mathrm{woman}]$,
  $\lambda Qc.((\exists\ ;\ \mathit{Woman}^{\circ} i) \Rightarrow Qi)c$ where $i = |c|$,
  $(\mathrm{Ref} \to \mathrm{Sent}) \to \mathrm{Sent}$.

- $\lambda x.x{+\!\!+}[\mathrm{smiled}]$,
  Smile°,
  $\mathrm{Ref} \to \mathrm{Sent}$.

In the treatment of proper names, we will assume that appropriate indices for proper names can get extracted from the input context.

Indeed, it will be assumed that all proper names are linked to anchored elements in context. The incrementality of the context update mechanism ensures that no anchored elements can ever be overwritten.

Suppose $(\dagger) :: (e, [e]) \to \iota$ is the function that gives the first index i in c with $c[i] = x$. Then if $n :: e$ and $c :: [e]$ is a context in which n occurs, $\dagger(n, c)$ gives the lowest index of n in c. In case there is no such an index, this is not well-defined, but we will assume contexts that have referents for all proper names. Let $name :: e$. We define:

$$name^{\odot} \;:=\; \lambda Pcc'.Picc' \text{ where } i = \dagger(name, c).$$

Note that $(^{\odot}) :: e \to (\iota \to T) \to T$, and $name^{\odot} :: (\iota \to T) \to T$. Let *John* be a constant of type e. Then the following triple is a sign:

$$(\lambda X.X[\text{john}], John^{\odot}, (\text{Ref} \to \text{Sent}) \to \text{Sent}).$$

Combining Signs
The simplest way of combining signs is by application in each of the first two dimensions.

$$\begin{aligned} & (\lambda X.X[\text{john}], John^{\odot})(\lambda x.x{+\!\!+}[\text{smiled}], Smile^{\circ}) \\ \overset{\beta}{\rightarrowtail} \; & ([\text{john}, \text{smiled}], \lambda cc'.c = c' \wedge Smile(c[i]) \text{ where } i = \dagger(John, c)) \end{aligned}$$

In this example we considered the first sign as a function that takes the second sign as its argument. This boils down to using the following application combinator:

$$\begin{aligned} C_1 \;&::\; (\alpha \to \beta) \to \alpha \to \beta \\ C_1 \;&=\; \lambda \mathcal{F}\mathcal{X} \cdot \mathcal{F}\mathcal{X}. \end{aligned}$$

The first line gives the type specification, the second line the definition. The type specification indicates that the first argument of C_1 can be any function and the second argument any argument to that function. We get:

$$\begin{aligned} & C_1(\lambda X.X[\text{john}], John^{\odot})(\lambda x.x{+\!\!+}[\text{smiled}], Smile^{\circ}) \\ \overset{\beta}{\rightarrowtail} \; & ([\text{john}, \text{smiled}], \lambda cc'.c = c' \wedge Smile(c[i]) \text{ where } i = \dagger(John, c)) \end{aligned}$$

Other ways of combining signs are possible. 'Consider the first sign as an argument of the second sign' would correspond to the following combinator:

$$\begin{aligned} C_2 \;&::\; \alpha \to (\alpha \to \beta) \to \beta \\ C_2 \;&=\; \lambda \mathcal{X}\mathcal{F} \cdot \mathcal{F}\mathcal{X}. \end{aligned}$$

An operator that we will need is C_3, for combining a transitive verb with its direct object.

$$\begin{aligned} C_3 &:: (\alpha \to \alpha \to \beta) \to ((\alpha \to \beta) \to \alpha) \to \alpha \to \beta \\ C_3 &= \lambda\mathcal{R}\mathcal{X}y.\mathcal{X}(\lambda x.\mathcal{R}xy). \end{aligned}$$

We illustrate this with an example in the syntax dimension:

$$\begin{aligned} & C_3(\lambda xy.y{+\!\!+}[\text{loved}]{+\!\!+}x)(\lambda X.X[\text{every}, \text{woman}]) \\ \overset{\beta}{\longrightarrow\!\!\!\!\!\rightarrow}\ & \lambda y.y{+\!\!+}[\text{loved}, \text{every}, \text{woman}] \end{aligned}$$

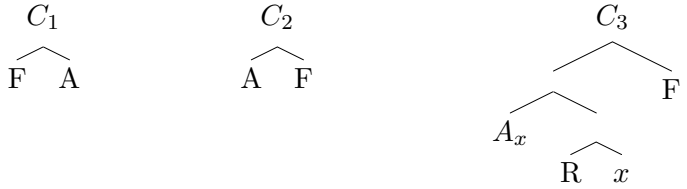
In the semantics dimension, this works out in the same way:

$$\begin{aligned} & C_3\ (\mathit{Love}^\bullet)\ (\lambda Qc.((\exists\ ;\ \mathit{Woman}^\circ i) \Rightarrow Qi)c \text{ where } i = |c|) \\ \overset{\beta}{\longrightarrow\!\!\!\!\!\rightarrow}\ & \lambda jc.((\exists\ ;\ \mathit{Woman}^\circ i) \Rightarrow (\lambda c'c''(c' = c'' \wedge \mathit{Love}^\bullet c'[j]c'[i]))c \\ & \quad \text{where } i = |c|) \end{aligned}$$

This can be simplified further by expansion of Woman°, $\mathit{Love}^\bullet$ and the definitions of $\exists$, ; and $\Rightarrow$:

$$\overset{\beta}{\longrightarrow\!\!\!\!\!\rightarrow}\ \lambda jcc'.c = c' \wedge \forall x(\mathit{Woman}(x) \to \mathit{Love}(c[j], x)).$$

The following tree structures summarize what these combinators do:



It is also possible to define combinators that achieve the effects of quantifier raising: see (Mus02) for details.

7. A Fragment

For the fragment, all we have to do is given specifications for the signs.

Sentence and Text Formation

$(\lambda xy.[\text{if}]{+\!\!+}x{+\!\!+}[\text{then}]{+\!\!+}y,\ \lambda pq.p \Rightarrow q,\ \text{Sent} \to \text{Sent} \to \text{Sent})$

$(\lambda xy.x{+\!\!+}[.]{+\!\!+}y,\ \lambda pq.p\ ;\ q,\ \text{Sent} \to \text{Sent} \to \text{Sent})$

If *NP* is an NP sign and *VP* a VP sign, then C_1 *NP VP* is a sentence sign.

Names

$$(\lambda X.X[\text{mary}],\ \ \mathit{Mary}^{\odot},\ \ (\text{Ref} \to \text{Sent}) \to \text{Sent})$$

Names combine with VPs to form sentences by means of combinator C_1.

Pronouns

Anaphoric reference resolution is the process of fixing the references of pronouns in a given context by linking the pronouns to appropriate context indices. After reference resolution, the interpretation of a pronoun is an index pointing to an appropriate context element.

$$([\text{PRO}_j],\ \ j,\ \ \text{Ref})$$

Pronouns combine with VPs to form sentences by means of combinator C_2.

Determiners

Let $\Downarrow$: $([e] \to t) \to t$ be the function that gives *true* for a context set $P :: [e] \to t$ just in case P is not empty. Then $\Downarrow$ is an operation for success. The sign type for all determiner signs is Noun $\to$ (Ref $\to$ Sent) $\to$ Sent.

$$\begin{array}{ll}
(\lambda xX.X[\text{every}{+\!\!+}x], & \lambda PQc.(\neg\!\!\neg(\exists\ ;\ P|c|\ ;\ \neg\!\!\neg Q|c|))c) \\
(\lambda xX.X[\text{some}{+\!\!+}x], & \lambda PQc.(\exists\ ;\ P|c|\ ;\ Q|c|)c) \\
(\lambda xX.X[\text{no}{+\!\!+}x], & \lambda PQc.(\neg\!\!\neg(\exists\ ;\ P|c|\ ;\ Q|c|))c) \\
(\lambda xX.X[\text{the}{+\!\!+}x], & \lambda PQc.((\lambda c'.c = c' \wedge \\
 & \quad \exists x \forall y(\Downarrow(\exists\ ;\ Pi\ (i|y)c) \leftrightarrow x = y)) \\
 & \quad\ ;\ \exists\ ;\ Pi\ ;\ Qi)c \text{ where } i = |c|)
\end{array}$$

Relative Clauses

Let ϵ be the empty list. The relative clause formator *that* takes a sentence with a gap for a referent in it, (syntactic type $S \to S$, semantic type $\iota \to T$, sign type Ref $\to$ Sent), fills the gap with the empty list, and produces a function from nouns to nouns. Thus, the sign type of the relative clause formator is (Ref $\to$ Sent) $\to$ Noun $\to$ Noun. The syntactic and semantic parts look like this:

$$(\lambda Xx.x{+\!\!+}[\text{that}]{+\!\!+}X(\epsilon),\ \ \lambda QPj.(Pj\ ;\ Qj))$$

Common Nouns

$$\begin{array}{lll}
([\text{woman}], & \mathit{Woman}^{\circ}, & \text{Noun}). \\
([\text{man}], & \mathit{Man}^{\circ}, & \text{Noun}).
\end{array}$$

If *CLAUSE* is a clause sign (sign type Ref → Sent), *THAT* is the relative clause operator sign, and *CN* is a noun sign, then a complex common noun is construed by the following rule:

$$C_2\,CN(C_1\,THAT\ CLAUSE).$$

VPs

$$(\lambda x.x{+\!\!+}[\text{laughed}],\ \mathit{Laugh}^{\circ},\ \text{Ref} \to \text{Sent})$$
$$(\lambda x.x{+\!\!+}[\text{smiled}],\ \mathit{Smile}^{\circ},\ \text{Ref} \to \text{Sent})$$

For VPs consisting of a TV and a direct object, the rule is given by $C_3\,TV\ NP$, where *TV* is the TV sign and *NP* the sign for the direct object.

TVs

$$(\lambda xy.y{+\!\!+}[\text{loved}]{+\!\!+}x,\ \mathit{Love}^{\bullet},\ \text{Ref} \to \text{Ref} \to \text{Sent})$$
$$(\lambda xy.y{+\!\!+}[\text{respected}]{+\!\!+}x,\ \mathit{Respect}^{\bullet},\ \text{Ref} \to \text{Ref} \to \text{Sent})$$

8. Solution of the Coordination Puzzle

Sign for 'a man':

$$\lambda X.X[\text{a}, \text{man}],$$
$$\lambda Qc.((\exists\ ;\ \mathit{Man}^{\circ} i)\ ;\ Qi)c \text{ where } i = |c|,$$
$$(\text{Ref} \to \text{Sent}) \to \text{Sent}$$

Sign for 'entered':

$$\lambda x.x{+\!\!+}[\text{entered}],\ \mathit{Enter}^{\circ},\ \text{Ref} \to \text{Sent}.$$

Sign for 'a man entered', after reduction:

$$[\text{a}, \text{man}, \text{entered}],$$
$$\lambda cc'.\exists x(c' = c\hat{}x \wedge \mathit{Man}\ x \wedge \mathit{Enter} x,$$
$$\text{Sent}$$

Sign for 'a man left', after reduction:

$$[\text{a}, \text{man}, \text{left}],$$
$$\lambda cc'.\exists x(c' = c\hat{}x \wedge \mathit{Man}\ x \wedge \mathit{Leave}\ x,$$
$$\text{Sent}$$

Sign for 'a man entered and a man left', after some reduction:

$$[\text{a}, \text{man}, \text{entered}, \text{and}, \text{a}, \text{man}, \text{left}],$$
$$\lambda cc'.\exists x(\mathit{Man}\ x \wedge \mathit{Enter}\ x \wedge c\hat{}x = c')$$
$$\ ;$$
$$\lambda cc'.\exists x(\mathit{Man}\ x \wedge \mathit{Leave}\ x \wedge c\hat{}x = c'),$$
$$\text{Sent}.$$

Sign for 'a man entered and a man left', after final reduction:

[a, man, entered, and, a, man, left],
$\lambda cc'.\exists x(\mathit{Man}\ x \wedge \mathit{Enter}\ x \wedge \exists y(\mathit{Man}\ y \wedge \mathit{Leave}\ y \wedge c\hat{}x\hat{}y = c'))$,
Sent.

9. Conclusion

Anaphoric reference resolution in context logic is determined on the fly, on the basis of:

- Syntactic properties of the sentence that contains the pronoun.
- Information conveyed in the previous discourse.
- Background information shared by speaker and hearer (the common ground).

An account of how the simple contexts discussed above can be enriched to allow for salience updates is provided in (Eij02). The basic change is the replacement of contexts with contexts under permutation, type $p[e]$. Let $x : c$ be the operation that extends context c with object x, while at the same time pushing x to the most salient position in the new context. Then the new definition of context extension runs like this:

$$\exists := \lambda cc'.\exists x((x : c) = c')$$

Here it is assumed that $c, c' :: p[e]$, $P, Q :: \iota \to p[e] \to p[e] \to t$. The new translation of 'a man' effects a salience reshuffle:

$$\lambda Qcc' \cdot \exists x(\mathit{Man}(x) \wedge Qi(x : c)c') \text{ where } i = |c|.$$

Context semantics is flexible enough to take syntactic effects on salience ordering into account, for lambda abstraction allows us to make flexible use of the salience updating mechanism. To see why this is so, note that in systems of typed logic, predicate argument structure is a feature of the 'surface syntax' of the logic. Consider the difference between the following formulas:

$$(\lambda xy.Kxy)(b)(j)$$
$$(\lambda xy.Kyx)(j)(b)$$
$$(\lambda x.Kbx)(j)$$

All of these reduce to Kbj, but the predicate argument structure is different. Surface predicate argument structure of lambda expressions can be used to encode the relevant salience features of surface syntax, and we can get the right salience effects from the surface word order of examples like *Bill kicked John* versus *John got kicked by Bill* versus *John, Bill kicked*.

Anaphoric reference resolution can now be implemented by a mechanism for picking the indices of the entities satifying the appropriate gender constraint from the current context, in order of salience. The result of reference resolution is a list of indices, in an order of preference determined by the salience ordering of the context.

Thus, the meaning of a pronoun, given a context, is an invitation to pick indices from the context. This can be further refined in a set-up that also stores syntactic information (about gender, case, and so on) as part of the contexts.

The proposed reference resolution mechanism provides an ordering of resolution options determined by syntactic structure, semantic structure, and discourse structure. This shows that pronoun reference resolution can be brought within the compass of dynamic semantics in a relatively straightforward way, and that the mechanism can be viewed as an extension of pronoun reference resolution mechanisms proposed for DRT (WA86; BB99). With minimal modification, the proposal also takes the so-called 'actor focus' from the *centering theory* of local coherence in discourse (GS86; GJW95) into account. Contexts ordered by salience are a suitable datastructure for further refinement of the reference resolution mechanism by means of modules for discourse focus and world knowledge (WJP98).

References

P. Blackburn and J. Bos. *Representation and Inference for Natural Language; A First Course in Computational Semantics — Two Volumes*. Internet, 1999. Electronically available from `http://www.coli.uni-sb.de/~bos/comsem/`.

D. Bekki. Typed dynamic logic for E-type link. In *Proceedings for Third International Conference on Discourse Anaphora and Anaphor Resolution (DAARC2000)*, pages 39–48. Lancaster University, U.K., 2000.

G. Chierchia. Anaphora and dynamic binding. *Linguistics and Philosophy*, 15(2):111–183, 1992.

M.J. Cresswell. *Logics and Languages*. Methuen, London, 1973.

P. Dekker. Predicate logic with anaphora. In L. Santelmann and M. Harvey, editors, *Proceedings of the Fourth Semantics and Linguistic Theory Conference*, page 17 vv, Cornell University, 1994. DMML Publications.

P. Dekker. Representation and information in dynamic semantics. In Jerry Seligman and Dag Westerståhl, editors, *Language, Logic and Computation*, pages 183–197. CSLI, Stanford, 1996.

J. van Eijck. Typed logics with states. *Logic Journal of the IGPL*, 5(5):623–645, 1997.

J. van Eijck. Incremental dynamics. *Journal of Logic, Language and Information*, 10:319–351, 2001.

J. van Eijck. Reference resolution in context. In M. Theune, A. Nijholt, and H. Hondorp, editors, *Computational Linguistics in the Netherlands 2001 Selected Papers from the Twelfth CLIN Meeting*, pages 89–103. Rodopi, 2002.

J. van Eijck and H. Kamp. Representing discourse in context. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 179–237. Elsevier, Amsterdam, 1997.

B. Grosz, A. Joshi, and S. Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21:203–226, 1995.

B.J. Grosz and C.L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12:175–204, 1986.

J. Groenendijk and M. Stokhof. Dynamic Montague Grammar. In L. Kalman and L. Polos, editors, *Papers from the Second Symposium on Logic and Language*, pages 3–48. Akademiai Kiadoo, Budapest, 1990.

J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.

I. Heim. *The Semantics of Definite and Indefinite Noun Phrases*. PhD thesis, University of Massachusetts, Amherst, 1982.

J. Roger Hindley. *Basic Simple Type Theory*. Cambridge University Press, 1997.

The Haskell Team. The Haskell homepage. `http://www.haskell.org`.

Martin Jansche. Dynamic Montague Grammar lite. Dept of Linguistics, Ohio State University, November 1998.

H. Kamp. A theory of truth and semantic representation. In J. Groenendijk et al., editors, *Formal Methods in the Study of Language*. Mathematisch Centrum, Amsterdam, 1981.

M. Kohlhase, S. Kuschert, and M. Pinkal. A type-theoretic semantics for λ-DRT. In P. Dekker and M. Stokhof, editors, *Proceedings of the Tenth Amsterdam Colloquium*, Amsterdam, 1996. ILLC.

H. Kamp and U. Reyle. *From Discourse to Logic*. Kluwer, Dordrecht, 1993.

S. Kuschert. *Dynamic Meaning and Accommodation*. PhD thesis, Universität des Saarlandes, 2000. Thesis defended in 1999.

R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17, 1978.

R. Montague. The proper treatment of quantification in ordinary English. In J. Hintikka e.a., editor, *Approaches to Natural Language*, pages 221–242. Reidel, 1973.

R. Muskens. A compositional discourse representation theory. In P. Dekker and M. Stokhof, editors, *Proceedings 9th Amsterdam Colloquium*, pages 467–486. ILLC, Amsterdam, 1994.

R. Muskens. Tense and the logic of change. In U. Egli et al., editor, *Lexical Knowledge in the Organization of Language*, pages 147–183. W. Benjamins, 1995.

R. Muskens. Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy*, 19:143–186, 1996.

R. Muskens. Language, lambdas and logic. Manuscript, Tilburg University, 2002.

R. Oehrle. Term-labeled categorial type systems. *Linguistics and Philosophy*, 17:633–678, 1994.

S. Orey. Model theory for the higher order predicate calculus. *Transactions of the American Mathematical Society*, 92:72–84, 1959.

C.F.M. Vermeulen. Sequence semantics for dynamic predicate logic. *Journal of Logic, Language, and Information*, 2:217–254, 1993.

C.F.M. Vermeulen. Merging without mystery. *Journal of Philosophical Logic*, 24:405–450, 1995.

H. Wada and N. Asher. BUILDRS: An implementation of DR theory and LFG. In *11th International Conference on Computational Linguistics. Proceedings of Coling '86*, University of Bonn, 1986.

M. Walker, A. Joshi, and E. Prince, editors. *Centering Theory in Discourse*. Clarendon Press, 1998.

H. Zeevat. A compositional approach to discourse representation theory. *Linguistics and Philosophy*, 12:95–131, 1989.