John sees Mary

Greg Kobele

March 4, 2014

1 Overview

The basic structure I will be assigning to the sentence "*John sees Mary*" is as in figure 1. This structure can be thought of as familiar label-free structure,

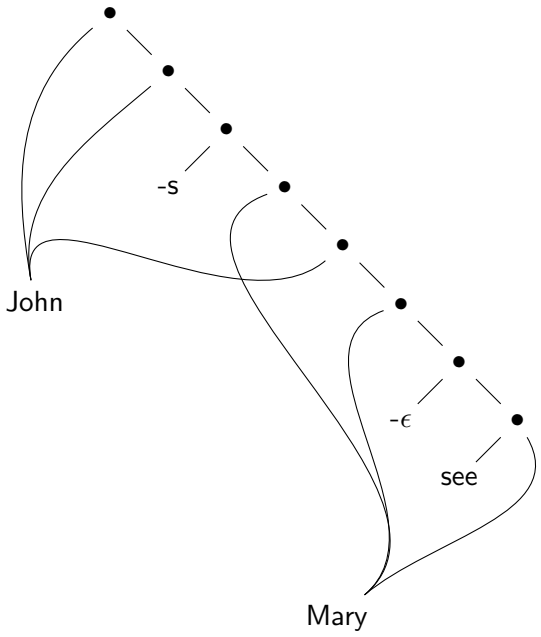

Figure 1: Structure of "*John sees Mary*"

with a multi-domainance perspective on movement chains. Although the nodes are ordered on the page, this is not part of this structure; this can be thought of along the standard LCA lines [Kayne, 1994].

It is perhaps easier to understand this structure if we add in node labels. A perhaps more helpful presentation of this structure is given as figure 2.

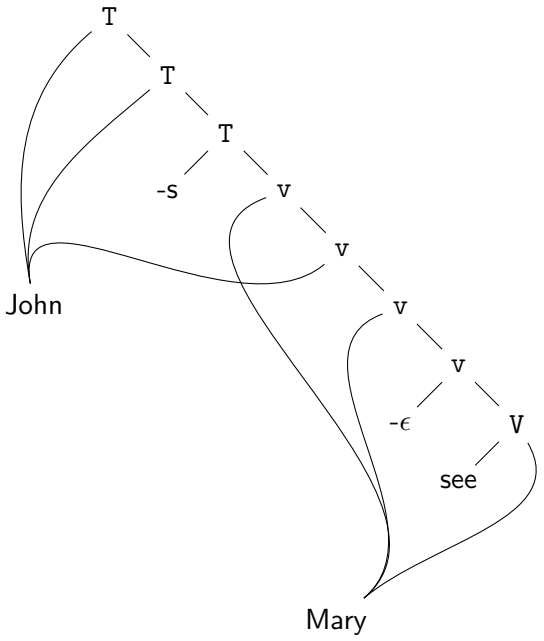

Figure 2: Structure of "*John sees Mary*", with labels

I am assuming post-syntactic head movement of one form [Stabler, 1997] or another [Kobele, 2002], to derive the complex head *sees* from the pieces see, -ϵ, and -s.

This is mostly a familiar analysis of this sentence, assuming that objects check case before the subject is merged [Koizumi, 1995]. (The head labeled -ϵ is a fusion of little-v and AgrO. I have treated this as one head here for simplicity.) The only unusual aspect of this is the extra chain position for both subject and object. This is because I follow Hornstein [1995] in adopting a reconstruction approach to quantifier scope, and the object must therefore be in a position higher than the base position of the subject. (For other reasons, I want the object to check its case before the subject is merged.)

I prefer yet another labeling scheme, given in figure 3. I prefer this scheme because I know how to generate it exactly; it is simply the derivation for the sentence "*John sees Mary*" using the lexical items given in table 1. Viewed as a derivation from the lexical items in table 1, the arcs from the

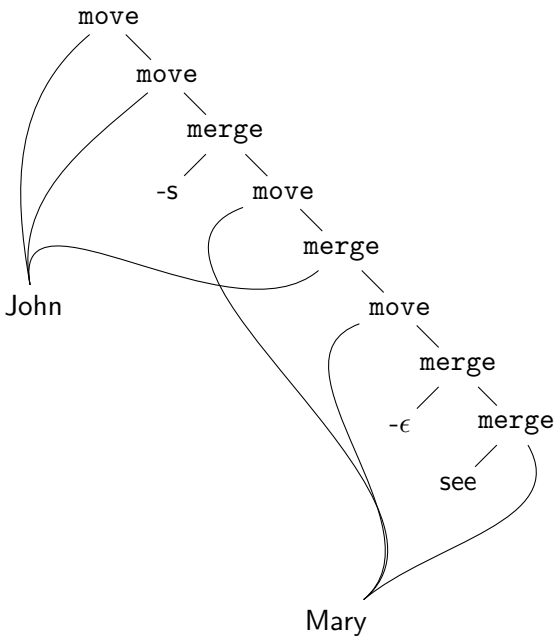

Figure 3: Derivation tree for "*John sees Mary*"

nodes labeled `move` to the movers are redundant, in the sense that they are recoverable from the architecture of the system. Figure 4 shows them removed. Finally, for purely aesthetic purposes, I prefer the expression with the selector feature (`=x`) to come before the one with the selectee feature (`x`) in derivation trees. My preferred representation is given as figure 5 below. In this case, only the relative ordering of *John* and its sister have changed.

I want to emphasize that the structure in figure 5 just *is* the structure we began with in figure 1. It looks different only because of the irrelevant

⟨John, d -k -q⟩	⟨-s, =v +k +q s⟩
⟨Mary, d -k -q⟩	⟨-ϵ, =V +k =d +q v⟩
	⟨see, =d V⟩

Table 1: Lexical items (from Kobele [2006])

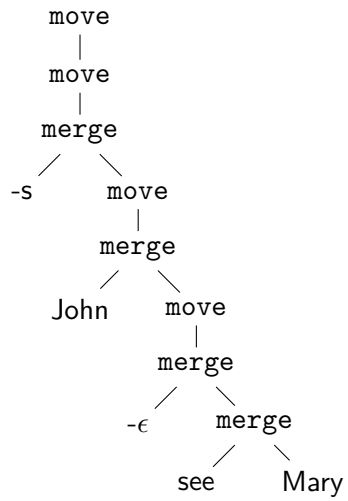

Figure 4: Derivation tree for "*John sees Mary*", with reentrant arcs suppressed

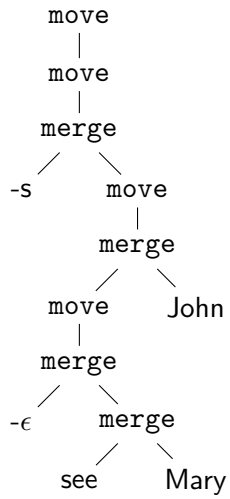

Figure 5: Derivation tree for "*John sees Mary*", without reentrant arcs, and with selectors before selectees

orderings of subtrees, and because of the suppressed information about reen-

trant arcs. This information is completely recoverable from the features on the lexical items.

1.1 $\texttt{merge}(A, B) = \{A, B\}$

Because of various properties of the MG system, the tree in figure 5 can be encoded as the following set-theoretic object:

$$\{\{\{\text{-s}, \{\{\{\{\text{-}\epsilon, \{\mathsf{see}, \mathsf{Mary}\}\}, \mathsf{John}\}\}\}\}\}$$

The encoding ϕ works as follows:

$$\begin{aligned} \phi(\ell) &= \ell \\ \phi(\texttt{move}(t)) &= \{\phi(t)\} \\ \phi(\texttt{merge}(t_1, t_2)) &= \{\phi(t_1), \phi(t_2)\} \end{aligned}$$

Crucially, this can be *decoded* as well, via a map ψ (in the third case I am assuming that $t_1 \neq t_2$):

$$\begin{aligned} \psi(\ell) &= \ell \\ \psi(\{t\}) &= \texttt{move}(\psi(t)) \\ \psi(\{t_1, t_2\}) &= \texttt{merge}(\psi(t_1), \psi(t_2)) \end{aligned}$$

Because whenever a subtree of the form $\texttt{merge}(t_1, t_2)$ occurs in a derivation, $t_1 \neq t_2$, we have that $\psi(\phi(t)) = t$ for all t; in other words $\psi \circ \phi$ is the identity function, and ψ is a left inverse of ϕ. Thus MG derivations can be given equivalently in terms of sets. This means, in particular, that any operation one would like to perform on one of these representations (sets or trees), one can define an equivalent operation on the other representation.

One way of thinking about the above is that we are here saying that $\texttt{merge}(A, B) = \{A, B\}$, and we are reducing `move` to `merge` in a non-canonical way by setting $\texttt{move}(A) = \texttt{merge}(A, A) = \{A\}$. As a slogan, move is self-merge. To recap: 1. MG derivations can be given solely in terms of the equation $\texttt{merge}(A, B) = \{A, B\}$, 2. there is no separate operation of movement; move is simply self-merge, and 3. all operations defined over derivation trees (like transfer §2) can be extended to these sets via the operations ϕ and ψ.

Since many people seem to desire that `move` be reducible to `merge`, and that `merge` be simply set formation, it behooves us to reflect upon this extremely straightforward answer to this problem.

First, observe that I have done exactly nothing interesting; I have simply encoded the usual MG derivation tree in terms of sets. Because of the fact that the two arguments to the `merge` operation are always distinct (an empirical observation?), this allows us formal room to encode a unary operation (in particular, `move`) as this unused case of `merge`. This is perhaps sneaky, but I do not see any deep linguistic insight herein.

Second, observe that my solution is perhaps the simplest possible; I do not need to allow `merge` to apply both to an expression and a subpart of itself (as would be required for `move`). Instead, only entire expressions are ever merged (i.e. put into sets) together.

Third, and most importantly, with this best possible solution to the problem at hand, we seem nowhere nearer to an evolutionary explanation of the emergence of the combinatory operations (`merge` and `move`) of syntax.[1]

2 Transfer

Structures like the ones above are only good in so far as they correctly pair up sounds and meanings.[2] We have seen (or at least, I have sketched) that the structures above contain the same information, in the sense that each can be transformed into any of the others, and back again, without any loss of information. Still, one might attribute some kind of importance to one representational scheme over the others, as seems to be behind the desire to represent things set theoretically (as in §1.1).

However, the representations we have been looking at *never need to be explicitly computed*; Michaelis [2001] and Kobele [2006] have shown how to perform transfer (-PF and -LF respectively) at each derivational step. What this means is that there can be no answer to the question of which representation is 'right', as the representation is a description of a process (of parsing, and generation), and is not a static thing which exists.[3] As Steedman [2000] puts it, these structures are *"no more than the trace of the algorithm that delivers the [...] interpretation"*.

[1] The most interesting ideas I have heard in this regard are those of Steedman [2002].

[2] *I* would say rather in so far as they help us account for actual observable properties of the world (e.g. linguistic behaviour). This seems (perversely in my opinion) an unfashionable view, but, as this note is not an apology for a philosophical position, I will not pursue this here further.

[3] To be a little more responsible, I should rather say that everything we would like to do can be accomplished without ever needing to postulate an object like the ones in the previous section. It could still be that this perspective is wrong. However, it is not obviously wrong, and that means that the question about notation only even makes sense under dubious assumptions about the architecture of the language faculty.

Both interface maps operate on objects with the same structure; finite sequences. Each position in the sequence (other than the first) corresponds to the interface-legible interpretation of a moving expression. In the case of the PF interface, I will write the interpretation of expressions as familiar strings, and in the case of the LF interface, as λ-terms.

To make things convenient, I number the nodes of the derivation tree for future reference in figure 6.

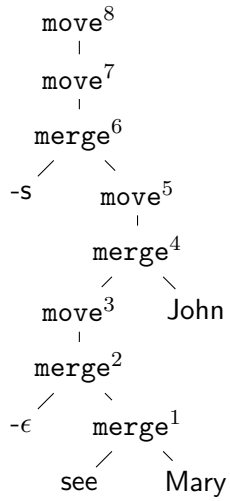

Figure 6: Numbers at nodes for easy reference

2.1 Transfer-PF

Here, our interface legible objects are finite sequences whose first component is a triple of strings (of the form $\langle u, v, w\rangle$), and all of whose other components are strings. In fact, all sequences will be of length three. Michaelis [2001] proves that this works, here I simply give the interface objects at each derivational step. Given an interface object, the string it corresponds to is obtained by simply concatenating the components of the triples in its first component.

To begin, the PF-interpretation of a lexical item like $\langle \mathsf{John}, \mathtt{d\ -k\ -q}\rangle$ is the tuple $\langle\langle\epsilon, \mathit{John}, \epsilon\rangle, \epsilon, \epsilon\rangle$. Similarly for the other lexical items.[4] The expression ϵ denotes the empty string, which is the identity for concatentation ($\epsilon^\frown w = w = w^\frown\epsilon$).

[4]This means that a lexical item like $\langle\alpha, \delta\rangle$ is interpreted as the tuple $\langle\langle\epsilon, \alpha, \epsilon\rangle, \epsilon, \epsilon\rangle$.

The interpretation of the remaining nodes of the derivation tree is given as follows:

1. merger of see and Mary

$$\langle\langle \epsilon, see, \epsilon\rangle, Mary, \epsilon\rangle$$

2. merger of -ϵ and the interpretation of 1

$$\langle\langle \epsilon, see^\frown\text{-}\epsilon, \epsilon\rangle, Mary, \epsilon\rangle$$

3. move applied to 2

$$\langle\langle \epsilon, see^\frown\text{-}\epsilon, \epsilon\rangle, \epsilon, Mary\rangle$$

4. merger of 3 and John

$$\langle\langle \epsilon, see^\frown\text{-}\epsilon, \epsilon\rangle, John, Mary\rangle$$

5. move applied to 4

$$\langle\langle Mary, see^\frown\text{-}\epsilon, \epsilon\rangle, John, \epsilon\rangle$$

6. merger of -s and 5

$$\langle\langle \epsilon, see^\frown\text{-}\epsilon^\frown\text{-}s, Mary\rangle, John, \epsilon\rangle$$

7. move applied to 6

$$\langle\langle \epsilon, see^\frown\text{-}\epsilon^\frown\text{-}s, Mary\rangle, \epsilon, John\rangle$$

8. move applied to 7

$$\langle\langle John, see^\frown\text{-}\epsilon^\frown\text{-}s, Mary\rangle, \epsilon, \epsilon\rangle$$

The string this corresponds to is obtained by concatenating the triple of strings in the first component, giving "*John sees Mary*".[5]

$\mathcal{I}(\mathsf{John})$	$\lambda P, c, \phi.P(\mathbf{j})(c)(\lambda d.\phi(\mathbf{j}{::}d))$
$\mathcal{I}(\mathsf{Mary})$	$\lambda P, c, \phi.P(\mathbf{m})(c)(\lambda d.\phi(\mathbf{m}{::}d))$
$\mathcal{I}(\mathsf{see})$	$\lambda x, y, c, \phi.\mathbf{see}(x)(y) \wedge \phi(c)$
everything else	the identity function

Table 2: The meanings of lexical items

2.2 Transfer-LF

Here, our interface legible objects are finite sequences of λ-terms. Again, all sequences will be of length three. (This is not a coincidence.) Kobele [2012] gives more explication, here I simply give the interface objects at each derivational step. Given an interface object, the meaning it corresponds to is its first component.

To begin, the meanings of the lexical items is given in table 2 (from Kobele [2012]). The interface object associated with a lexical item ℓ is then $\langle \mathcal{I}(\ell), \mathtt{id}, \mathtt{id}\rangle$, where $\mathtt{id}$ is the identity function. Note that, for purists, this can be encoded as a single λ-term:

$$\lambda f.f(\mathcal{I}(\ell))(\lambda x.x)(\lambda x.x)$$

The identity function is a left identity for function application ($\mathtt{id}(f) = f$).

The interpretation of the remaining nodes of the derivation tree is given as follows:

1. merger of see and Mary

$$\langle \lambda x_k.\mathcal{I}(\mathsf{see})(x_k), \mathcal{I}(\mathsf{Mary}), \mathtt{id}\rangle$$

2. merger of -ϵ and the interpretation of 1

$$\langle \lambda x_k.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(x_k)), \mathcal{I}(\mathsf{Mary}), \mathtt{id}\rangle$$

3. move applied to 2

$$\langle \lambda x_q.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(x_q)), \mathtt{id}, \mathcal{I}(\mathsf{Mary})\rangle$$

[5] I have erased dashes and the empty string ϵ. A more sophisticated approach to morphology could do something more interesting.

4. merger of 3 and John

$$\langle \lambda x_k, x_q.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(x_q))(x_k), \mathcal{I}(\mathsf{John}), \mathcal{I}(\mathsf{Mary})\rangle$$

5. move applied to 4

$$\langle \lambda x_k.\mathcal{I}(\mathsf{Mary})(\lambda y.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(y))(x_k)), \mathcal{I}(\mathsf{John}), \mathtt{id}\rangle$$

6. merger of $\text{-}\mathsf{s}$ and 5

$$\langle \lambda x_k.\mathcal{I}(\text{-}\mathsf{s})(\mathcal{I}(\mathsf{Mary})(\lambda y.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(y))(x_k)), \mathcal{I}(\mathsf{John}), \mathtt{id}\rangle$$

7. move applied to 6

$$\langle \lambda x_q.\mathcal{I}(\text{-}\mathsf{s})(\mathcal{I}(\mathsf{Mary})(\lambda y.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(y))(x_q)), \mathtt{id}, \mathcal{I}(\mathsf{John})\rangle$$

8. move applied to 7

$$\langle \mathcal{I}(\mathsf{John})(\lambda z.\mathcal{I}(\text{-}\mathsf{s})(\mathcal{I}(\mathsf{Mary})(\lambda y.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(y))(z)))), \mathtt{id}, \mathtt{id}\rangle$$

The meaning this corresponds to is simply the first component of the tuple, namely,

$$\mathcal{I}(\mathsf{John})(\lambda z.\mathcal{I}(\text{-}\mathsf{s})(\mathcal{I}(\mathsf{Mary})(\lambda y.\mathcal{I}(\text{-}\epsilon)(\mathcal{I}(\mathsf{see})(y))(z))))$$

Replacing all $\mathcal{I}(\ell)$ by their meanings as in table 2 (but abbreviating the meanings of the contentful lexical items by writing these in SMALL CAPS), we obtain:

$$\textsc{John}(\lambda z.\mathtt{id}(\textsc{Mary}(\lambda y.\mathtt{id}(\textsc{see}(y))(z))))$$

Reducing next the identity functions under the rule $\mathtt{id}(x) \to x$,[6] we obtain:

$$\textsc{John}(\lambda z.\textsc{Mary}(\lambda y.\textsc{see}(y)(z)))$$

Unpacking the abbreviation JOHN (from 2) we have:

$$(\lambda P, c, \phi.P(\mathbf{j})(c)(\lambda d.\phi(\mathbf{j}::d)))(\lambda z.\textsc{Mary}(\lambda y.\textsc{see}(y)(z)))$$

After β-reduction, we have:

$$\lambda c, \phi.\textsc{Mary}(\lambda y.\textsc{see}(y)(\mathbf{j}))(c)(\lambda d.\phi(\mathbf{j}::d))$$

[6]This reduction follows from the definition $\mathtt{id} := \lambda z.z$.

Doing the same for MARY, we obtain:

$$\lambda c, \phi.\textsc{see}(\mathbf{m})(\mathbf{j})(c)(\lambda d.\phi(\mathbf{j}::\mathbf{m}::d))$$

And doing the same for SEE, we arrive at the following.

$$\lambda c, \phi.\mathbf{see}(\mathbf{m})(\mathbf{j}) \wedge \phi(\mathbf{j}::\mathbf{m}::c)$$

This term encodes dynamic discourse context updates in the following way [de Groote, 2006]. Given as a discourse context c (here understood as a database of antecedents), rest of the discourse ϕ (a discourse continuation), the rest of the discourse is interpreted in the new discourse context updated with discourse referents for John and Mary, and the meaning of the sentence is that John saw Mary.

References

P. de Groote. Towards a montagovian account of dynamics. In M. Gibson and J. Howell, editors, *Proceedings of SALT 16*, pages 1–16, 2006.

N. Hornstein. *Logical Form: From GB to Minimalism*. Blackwell, Cambridge, Massachusetts, 1995.

R. Kayne. *The Antisymmetry of Syntax*. MIT Press, Cambridge, Massachusetts, 1994.

G. M. Kobele. Formalizing mirror theory. *Grammars*, 5(3):177–221, 2002.

G. M. Kobele. *Generating Copies: An investigation into structural identity in language and grammar*. PhD thesis, University of California, Los Angeles, 2006.

G. M. Kobele. Importing montagovian dynamics into minimalism. In D. Béchet and A. Dikovsky, editors, *Logical Aspects of Computational Linguistics*, volume 7351 of *Lecture Notes in Computer Science*, pages 103–118, Berlin, 2012. Springer.

M. Koizumi. *Phrase Structure In Minimalist Syntax*. PhD thesis, Massachusetts Institute of Technology, 1995.

J. Michaelis. *On Formal Properties of Minimalist Grammars*. PhD thesis, Universität Potsdam, 2001.

E. P. Stabler. Derivational minimalism. In C. Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95. Springer-Verlag, Berlin, 1997.

M. Steedman. *The Syntactic Process*. MIT Press, 2000.

M. Steedman. Plans, affordances, and combinatory grammar. *Linguistics and Philosophy*, 25(5-6):723–753, 2002.