The Semantics of Movement and Relative Clauses [1]

1. The Basic Problem

Syntacticians have long recognized a category of structures dubbed 'movement structures'. The structures in (1) all contain examples of 'movement'.

(1) **Examples of Movement**

a. Topicalization: **John**, Bill likes

b. Relativization: the man [**who** Bill likes **]**

c. Wh-Questions: **Who** does Bill like?

(2) **The Key Properties of Movement Structures**

a. There is a portion of the sentence that seems to be missing something

(*e.g.* the sequence Bill likes in (1a-c), which is missing a direct object)

b. The element 'missing' from that part of the sentence appears at the beginning of the sentence

(*e.g.* the element **John** or **who** in (1a-c))

(3) **The Overarching Question for This Unit**

How can we augment our semantic system so that such structures can be interpreted?

(4) **Our Starting Place**

- We'll begin by examining topicalization structures like (1a), since they differ from a simple declarative sentence (*e.g.* Bill likes John) only in the presence of movement.
- Having developed a semantics for sentences like (1a), we'll see how those basic ideas can also be employed to develop a semantics for relative clauses like (1b).

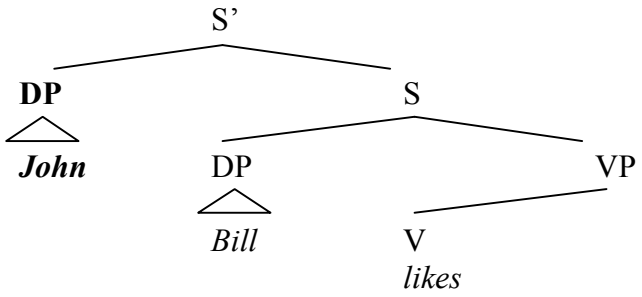
But before we can do any of that, we need to precisely lay out our assumptions about the syntactic structure of sentences like (1a).

[1] These notes are based on the material in Heim & Kratzer (1998: 86-115, 239-245).

2. The Syntax of Movement Structures

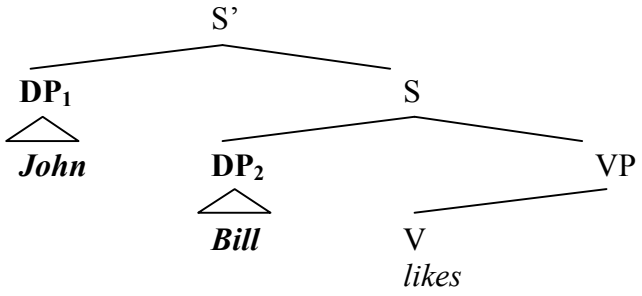
The assumptions in (5)-(8) have been standard for about 30 years, and are probably familiar to you from earlier classes (or LING 601).

(5) **Assumption 1: The Moved Phrase C-Commands The Material that Follows it**



(6) **Assumption 2: All DPs – Including Non-Pronominals – Bear Indices**

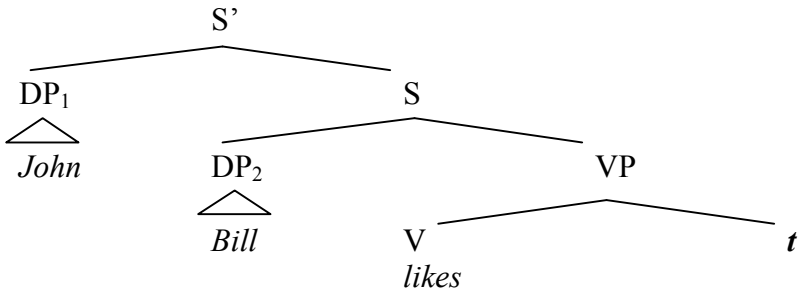
a. Syntax of Movement Structure



b. Irrelevance of Indices to Non-Pronominals
If DP_i is not headed by a pronoun, then $[[\ DP_i\ ]]^g = [[\ DP\ ]]$

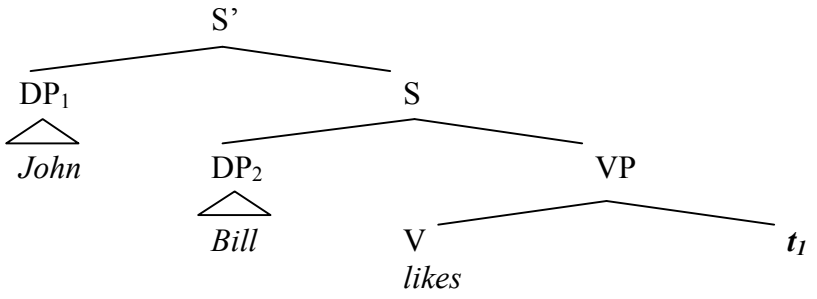
(7) **Assumption 3: Movement Leaves 'Traces'**

The 'missing material' in the movement structure is actually a phonologically empty element referred to as a 'trace'.



(8) **Assumption 4: Traces Are Coindexed with Moved Phrase**

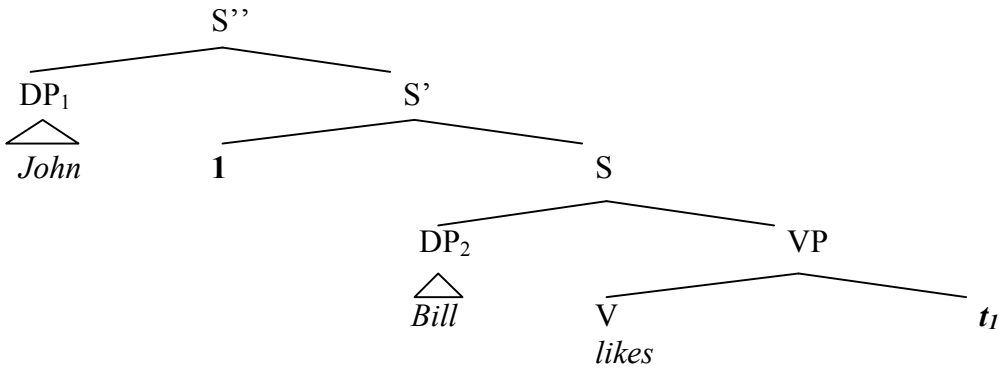
Like DPs, traces also bear indices. However, they necessarily bear the same index as the moved phrase.



The following assumption, critical for our semantic account, will probably be new to you…

(9) **Assumption 5: Index Copying**

The index of the moved phrase is copied as a sister to the sister of the moved phrase.



(10) **The Main Upshot**

The structure of (1a) will be assumed to be that in (9), which – except for 'index copying' – is pretty much what syntacticians have thought since the early 1980s.

(11) **A Quick Note on 'Index Copying'**

- This syntactic assumption was introduced by Heim & Kratzer 1998.
- Since its introduction, it has gained much ground with semanticists, and with 'semantically-minded' syntacticians…

3. Working Towards a Semantics of Movement

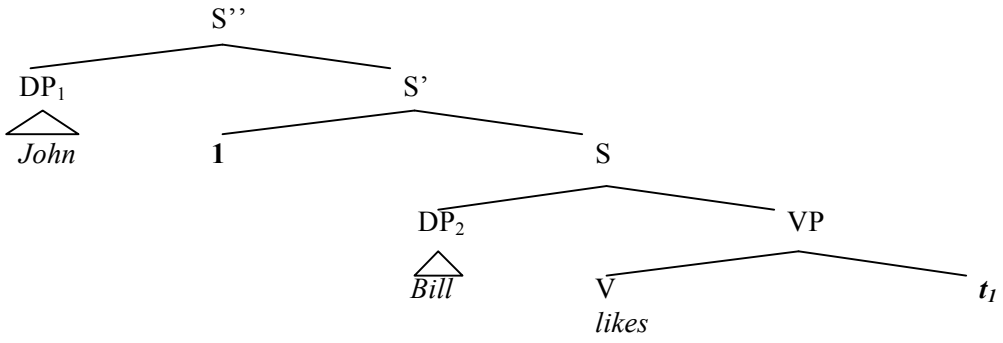
Now that we have a syntactic structure for (1a), let's start to work out its compositional semantics…

… we'll begin with a plausible assumption regarding the T-conditional statement for (1a) that our semantics should derive…

(12) **Assumed T-Conditional Statement**

"John, Bill likes" is T *iff* Bill likes John.

(13) **Working Out the Types**



[[S'']] :: Type *t*

[[John]] :: Type *e* *Therefore…* **[[S']] :: Type <*e,t*>**

(14) **Working Out the Extension of *S'***

a. *Starting T-Conditional Statement:*

[[S'']] = T *iff* Bill likes John.

b. *Key Consequence of the Types Deduced in (13)*

[[S'']] = [[S']] (John)

c. *Key Deductive Step*

[[S']] (John) = T *iff* Bill likes John.

d. *Key Generalization*

[[S']] is an <et> function will takes an entity x as argument and gives back T *iff* Bill likes x.

(15) **Deduced Extension for S'**

[[S']] = [λx_e : Bill likes x]

So, clearly, we want a semantics for movement structures that will deliver the equation in (15)…
but how do we do this?...

(16) **Key Observation: A Parallelism in Structure**

[[S']] =

[[[**1** [Bill likes t_1]]] =

[$\boldsymbol{\lambda x_e}$: Bill likes **x**]

- There are two key syntactic pieces to *S'*
  - (i) The index '1'
  - (ii) The trace bearing index 1
- There are two key syntactic parts to the extension of *S'*
  - (i) The lambda operator '$\boldsymbol{\lambda x_e}$'
  - (ii) The variable **x** bound by the lambda operator
- Even more interesting…these two pieces seem to correspond to each other nicely…

(17) **Working Hypothesis Regarding Movement Structures**

a. The index in S' is interpreted as the lambda operator in the extension of S'

b. The trace in S' is interpreted as the bound variable in the extension of S'

So, let's now work to formally implement the hypotheses in (17a) and (17b)…

... Doing so, however, will require us to introduce some new notation and a new semantic rule.

3. Formally Implementing Our Hypothesis

(18) **First Ingredient: Modified Variable Assignments**

Let *g* be a variable assignment, let *n* be an index, and let *a* be some entity.

We will write '*g* (*n* / *a*)' to mean 'the variable assignment which is exactly like *g*, *except that* it maps the index *n* to the entity *a*.

(19) **Illustrating the New Notation**

Let *g* be the following variable assignment: { <1 , Joe> , <2 , Barack> , <3 , Michelle> }

a. *g* (2 / Barney) = { <1 , Joe> , <2 , Barney> , <3 , Michelle> }

b. g (4 / Hillary) = { <1 , Joe> , <2 , Barack> , <3 , Michelle> , <4, Hillary> }

c. g (3/Hillary) (1/Nancy) = { <1 , Nancy> , <2 , Barack> , <3 , Hillary> }

The second ingredient in our formalization will be a new rule of semantic composition, specifically tied to the 'copied index' present in movement structures…

(20) **The Rule of Predicate Abstraction (PA) [Heim & Kratzer (1998: 186)]**

If X is a branching node whose daughters are Y and Z, and if Y is the index *n*, then for **any variable assignment g**,

$$[[\ X\ ]]^{g} \quad = \quad [\ \lambda x_{e} : \underline{[[\ Z\ ]]^{g(n/x)} = T}\ ]$$

(21) **Some Comments**

- The rule in (20) implements our 'working hypothesis' in (17a): *the 'copied index' in S' is interpreted (more or less) as the lambda operator in the extension of S'*
- The rule in (20) states that the sister to the 'copied index' (*i.e.*, S) is interpreted relative to the *new* variable assignment '*g* (n / x)'.

  This is the variable assignment just like *g* except the 'copied index' (n) is mapped to the variable bound by the lambda operator (x).

Finally, we need a rule that will allow us to interpret traces, and which is in line with our 'working hypothesis' in (17b)…

(22) **The Pronouns-and-Traces Rule (PR) [Heim & Kratzer (1998: 111)]**

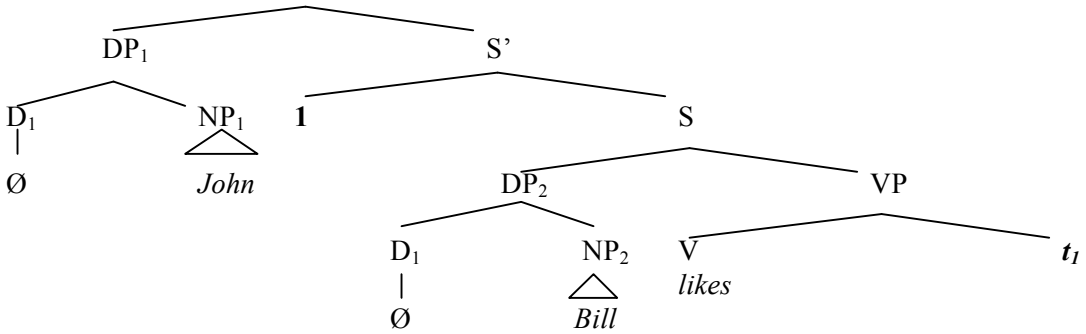
If X is a pronoun *or a trace*, and *g* is a variable assignment, and n is an index in the domain of *g*, then $[[X_n]]^g = g(n)$.

These three ingredients are enough to deliver our desired equation in (15)!!!

(23) **Sample Derivation for Topicalization Structure**

Let g be any variable assignment

a. " S'' " is T iff (by notation)



b. $[[\text{ S'' }]]^g = T$

c. **Subproof**
(i) $[[\text{ NP}_1\text{ }]]^g$ = (by NNx2, TN)
(ii) John

d. **Subproof**
(i) $[[\text{ D}_1\text{ }]]^g$ = (by NN, TN)
(ii) $[\ \lambda x_e : x\ ]$

e. **Subproof**
(i) $[[\text{ DP}_1\text{ }]]^g$ = (by FA, c, d)
(ii) $[[\text{ D}_1\text{ }]]^g ([[\text{ NP}_1\text{ }]]^g)$ = (by c, d)
(iii) $[\ \lambda x_e : x\ ]$ (John) = (by LC)
(iv) John

f. **Subproof**
(i) $[[\text{ S' }]]^g$ = **(by PA)**

(ii) $[\ \lambda x_e : \underline{[[\text{ S }]]^{g(1/x)} = T}\ ]$

Note: *To further compute out the extension of* ***S'****, we're going to need to compute the conditions under which* $[[\text{ S }]]^{g(1/x)} = T$

g. **Subproof**

(i)	$[\![NP_2]\!]^{g(1/x)}$	=	(by NNx2, TN)
(ii)	Bill		

h. **Subproof**

(i)	$[\![D_2]\!]^{g(1/x)}$	=	(by NN, TN)
(ii)	$[\lambda x_e : x]$		

i. **Subproof**

(i)	$[\![DP_2]\!]^{g(1/x)}$	=	(by FA, g, h)
(ii)	$[\![D_2]\!]^{g(1/x)} ([\![NP_2]\!]^{g(1/x)})$	=	(by g, h)
(iii)	$[\lambda x_e : x] (Bill)$	=	(by LC)
(iv)	Bill		

j. **Subproof**

(i)	$[\![V]\!]^{g(1/x)}$	=	(by NN, TN)
(ii)	$[\lambda y_e : [\lambda z_e : \underline{z \text{ likes } y}]]$		

Key part of proof!

k. **Subproof**

(i)	**$[\![t_1]\!]^{g(1/x)}$**	**=**	**(by PR)**
(ii)	**g(1/x) (1)**	**=**	**(by definition of 'g(1/x)')**
(iii)	**x**		

l. **Subproof**

(i)	$[\![VP]\!]^{g(1/x)}$	=	(by FA, j, k)
(ii)	$[\![V]\!]^{g(1/x)} ([\![t_1]\!]^{g(1/x)})$	=	(by j, k)
(iii)	$[\lambda y_e : [\lambda z_e : \underline{z \text{ likes } y}]] (x)$	=	(by LC)
(iv)	$[\lambda z_e : \underline{z \text{ likes } x}]$		

Key part of proof!

m. **Subproof**

(i)	$[\![S]\!]^{g(1/x)} = T$	*iff*	(by FA, l, i)
(ii)	$[\![VP]\!]^{g(1/x)} ([\![DP_2]\!]^{g(1/x)}) = T$	*iff*	(by l)
(iii)	$[\lambda z_e : \underline{z \text{ likes } x}] ([\![DP_2]\!]^{g(1/x)}) = T$	*iff*	(by i)
(iv)	$[\lambda z_e : \underline{z \text{ likes } x}] (Bill) = T$	*iff*	(by LC)
(v)	Bill likes x		

n. **Subproof**

(i) $[[\ S'\ ]]^g$ = **(by PA)**

(ii) $[\ \lambda x_e : \underline{[[\ S\ ]]^{g(1/x)} = T}\ ]$ = (by notation, m)

(iii) $[\ \lambda x_e : \underline{\text{Bill likes x}}\ ]$

o. $[[\ S''\ ]]^g = T$ *iff* (by FA, e, n)

p. $[[\ S'\ ]]^g\ (\ [[\ DP_1\ ]]^g\ ) = T$ *iff* (by e)

q. $[[\ S'\ ]]^g\ (\ \text{John}\ ) = T$ *iff* (by n)

r. $[\ \lambda x_e : \underline{\text{Bill likes x}}\ ]\ (\ \text{John}\ ) = T$ *iff* (by LC)

s. Bill likes John.

(24) **What Just Happened**

Because of the rule of PA in (20), the S [*Bill likes* t_1] is interpreted relative to a variable assignment where the index 1 is sent to the variable x…

> the same variable bound by the lambda introduced by the 'copied index 1' that is sister to S…

Consequently, according to the rule of PA, a structure of the form in (24a) is interpreted as the lambda expression in (24b)

a. [*index* […. $trace_{index}$ …]]

b. [λx_e : …. x ….]

Observation:
According to (22), the rule of PR now interprets *both* pronouns *and* traces.

Some Motivation:

- In classic GB theory, traces *were* a kind of pronoun.
- **Once we introduce quantifiers in the following unit, we will find further, empirical motivation for this choice:**
  The phenomenon of pronominal binding.

4. Extending the System to Relative Clauses

The semantics proposed for movement structures in (20) and (22) can be directly applied to provide a semantics for relative clauses like the following.

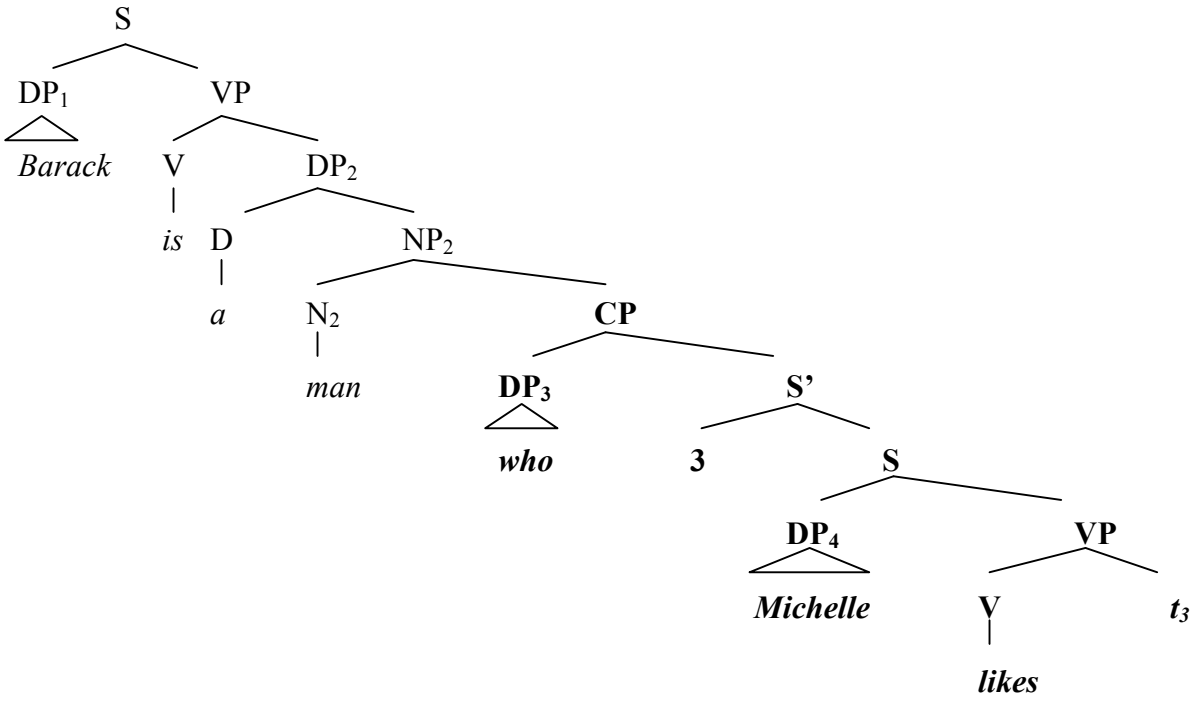
(25) **Examples of Relative Clauses**

a. Barack is a man **who Michelle likes.**
b. The man **who I voted for** won the election.
c. Mr. Pants is a cat **who needs a bath desperately.**

To begin, let's focus on the relatively simple example in (25a)…
… and, as always, let's lay out some clear assumptions about its syntax and its T-conditions

(26) **Assumed Syntax for Relative Clauses**

- The relative clause is a sister to (adjunct to) the NP (*man*)
- The relative clause is a movement structure where the relative pronoun (*who*) undergoes movement to its surface position.



(27) **Assumed T-Conditions**

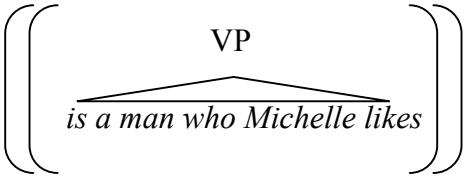
"Barack is a man who Michelle likes" is T *iff* Barack is a man and Michelle likes Barack.

[[S]] = T *iff* Barack is a man and Michelle likes Barack

From the assumptions in (26) and (27), we can begin to work towards a semantics for the relative clause….

(28) **First Key Observation**

Given the T-conditions in (27), it follows that:



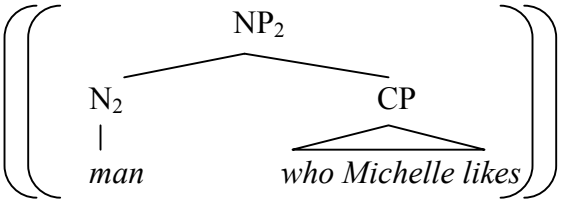
= [λz_e : z is a man and Michelle likes z]

(29) **Second Key Observation**

In the sentence in (25a)/(26), [[VP]] = [[NP_2]].

- This is because [[*is*]] = [[a]] = [$\lambda f_{<et>}$: f]

Thus, we can conclude:



= [λz_e : z is a man and Michelle likes z]

(30) **Third Key Observation**

If the relative clause *who Michelle likes* had the semantics in (30a), we could derive the equation in (29) via our rule of Predicate Modification (PM).

a. [[*who Michelle likes*]] = [λx_e : Michelle likes x]

b. **Derivation of (29) on the Basis of (30a)**

(i) [[NP_2]] = (by PM)

(ii) [λz_e : [[N_2]](z) = T and [[CP]](z) = T] = (by NN, TN)

(iii) [λz_e : [λx_e : x is a man](z) = T and [[CP]](z) = T] = (by LC)

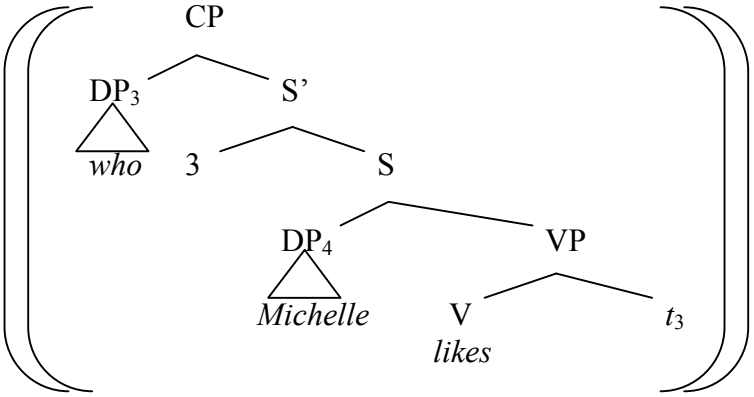
(iv) [λz_e : z is a man and [[CP]](z) = T] = (by 30a)

(v) [λz_e : z is a man and [λx_e : Michelle likes x] (z) = T] = (by LC)

(vi) [λz_e : z is a man and Michelle likes z]

(31) **Conclusion**

We should add a lexical entry to our system that interprets relative pronouns, and which will allow us to derive the equation below.



= [λx_e : Michelle likes x]

(32) **Fourth Key Observation**

Given our semantics for movement, the S' in the relative clause in (26)/(31) is such that:

[[S']] = [λx_e : Michelle likes x]

Derivation of the Equation

a. $[[S']]^g$ = (by PA)

b. [λx_e : $[[S]]^{g(3/x)} = T$]

c. **Subproof**
(i) $[[DP_4]]^{g(3/x)}$ = (by FA, etc.)
(ii) Michelle

d. **Subproof**
(i) $[[V]]^{g(3/x)}$ = (by NN, TN)
(ii) [λy_e : [λz_e : z likes y]]

e. **Subproof**
(i) $[[t_3]]^{g(3/x)}$ = (by PR)
(ii) x

f. **Subproof**
(i) $[[VP]]^{g(3/x)}$ = (by FA, d, e)
(ii) $[[V]]^{g(3/x)} ([[t_3]]^{g(3/x)})$ = (by d , e)
(iii) [λy_e : [λz_e : z likes y]] (x) = (by LC)
(iv) [λz_e : z likes x]

g. **Subproof**

(i)	$[\![S]\!]^{g(3/x)} = T$ *iff*	(by FA, c, f)
(ii)	$[\![VP]\!]^{g(3/x)} ([\![DP_4]\!]^{g(3/x)}) = T$ *iff*	(by c, f)
(iii)	$[\lambda z_e$: z likes x] (Michelle) = T *iff*	(by LC)
(iv)	Michelle likes x	

h. $[\lambda x_e : [\![S]\!]^{g(3/x)} = T]$ = (by notation, g)

i. $[\lambda x_e$: Michelle likes x]

(33) **What We Know So Far**

a. [[[who$_3$ [3 [Michelle likes t_3]]]]] = $[\lambda x_e$: Michelle likes x]

b. [[[3 [Michelle likes t_3]]]]] = $[\lambda x_e$: Michelle likes x]

So, how can we add a lexical entry for "who" so that we can derive the equation in (33a) from that in (33b), which already follows from our system???

(34) **The Big Idea**

Relative pronouns are identity functions on <et> functions

[[who]] = $[\lambda f_{<et>} : f]$

The assumption in (34) validates the following calculation:

(35) **Deriving the Equation in (31)/(33a)**

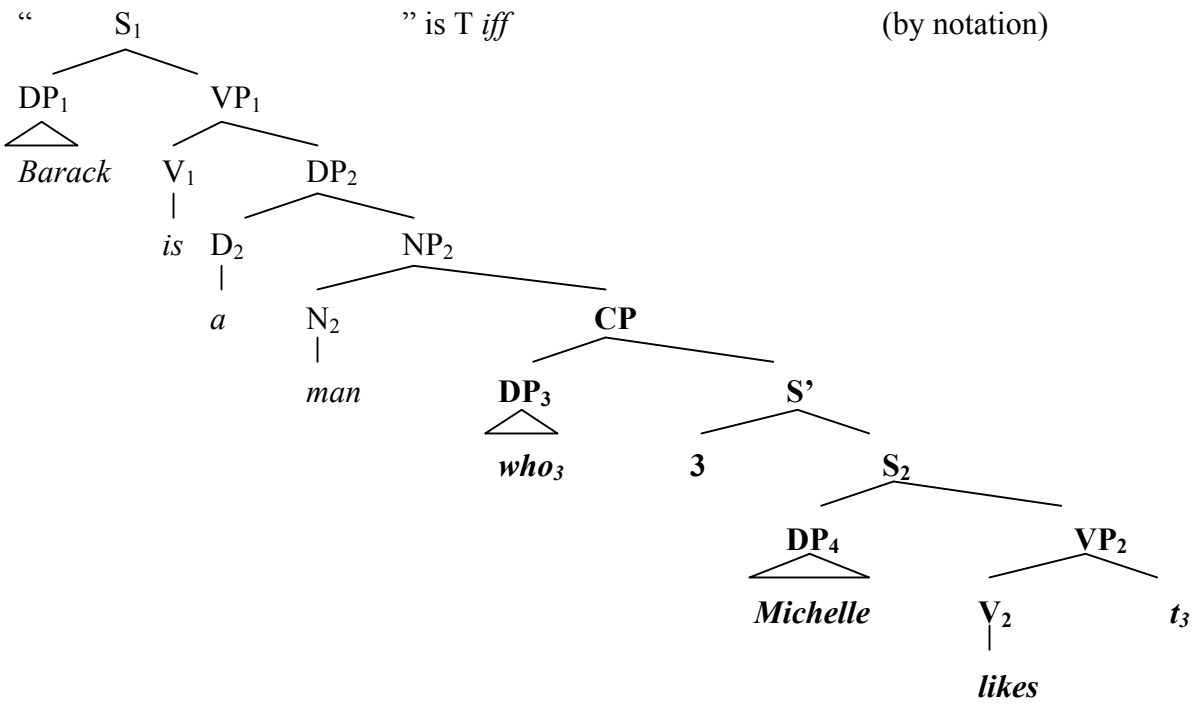
a.	[[[who$_3$ [3 [Michelle likes t_3]]]]]^g	=	(by FA, (32))
b.	[[who$_3$]]^g ([[[3 [Michelle likes t_3]]]]^g)	=	(by TN)
c.	$[\lambda f_{<et>} : f]$ ([[[3 [Michelle likes t_3]]]]^g)	=	(by LC)
d.	[[[3 [Michelle likes t_3]]]]^g	=	(by (32))
e.	$[\lambda x_e$: Michelle likes x]		

With the lexical entry in (34), we now have enough to interpret the entire sentence in (26).

(36) **The Full T-Conditional Derivation of (26)**

Let g be any variable assignment.

a.



b. $[\![S_1]\!]^g = T$

c. **Subproof**
(i) $[\![D_1]\!]^g$ = (by NN, TN)
(ii) $[\lambda x_e : x]$

d. **Subproof**
(i) $[\![NP_1]\!]^g$ = (by NNx2, TN)
(ii) Barack

e. **Subproof**
(i) $[\![DP_1]\!]^g$ = (by FA, c, d)
(ii) $[\lambda x_e : x]$(Barack) = (by LC)
(iii) Barack

f. **Subproof**
(i) $[\![V_1]\!]^g$ = (by NN, TN)
(ii) $[\lambda f_{<et>} : f]$

g. **Subproof**
(i) $[\![D_2]\!]^g$ = (by NN, TN)
(ii) $[\lambda f_{<et>} : f]$

h. **Subproof**

(i)	$[\![N_2]\!]^g$	=	(by NN, TN)
(ii)	$[\lambda x_e : \underline{x \text{ is a man}}]$		

i. **Subproof**

(i)	$[\![DP_3]\!]^g$	=	(by NNx2)
(ii)	$[\![who_3]\!]^g$	=	(by TN)
(iii)	$[\lambda f_{<et>} : f]$		

j. **Subproof**

(i)	$[\![S']\!]^g$	=	(by PA)
(ii)	$[\lambda x_e : \underline{[\![S]\!]^{g(3/x)} = T}]$		

k. **Subproof**

(i)	$[\![D_4]\!]^{g(3/x)}$	=	(by NN, TN)
(ii)	$[\lambda x_e : x]$		

l. **Subproof**

(i)	$[\![NP_4]\!]^{g(3/x)}$	=	(by NNx2, TN)
(ii)	Michelle		

m. **Subproof**

(i)	$[\![DP_4]\!]^{g(3/x)}$	=	(by FA, k, l)
(ii)	$[\lambda x_e : x]$(Michelle)	=	(by LC)
(iii)	Michelle		

n. **Subproof**

(i)	$[\![V_2]\!]^{g(3/x)}$	=	(by NN, TN)
(ii)	$[\lambda y_e : [\lambda z_e : \underline{z \text{ likes } y}]]$		

o. **Subproof**

(i)	$[\![t_3]\!]^{g(3/x)}$	=	(by PR)
(ii)	x		

p. **Subproof**

(i)	$[\![VP_2]\!]^{g(3/x)}$	=	(by FA, n, o)
(ii)	$[\![V_2]\!]^{g(3/x)} ([\![t_3]\!]^{g(3/x)})$	=	(by n, o)
(iii)	$[\lambda y_e : [\lambda z_e : \underline{z \text{ likes } y}]] (x)$	=	(by LC)
(iv)	$[\lambda z_e : \underline{z \text{ likes } x}]$		

q. **Subproof**

(i)	$[\![S]\!]^{g(3/x)} = T$ *iff*	(by FA, m, p)
(ii)	$[\![VP_2]\!]^{g(3/x)} ([\![DP_4]\!]^{g(3/x)}) = T$ *iff*	(by m, p)
(iii)	$[\lambda z_e : \underline{z \text{ likes } x}]$ (Michelle) = T *iff*	(by LC)
(iv)	Michelle likes x	

r. **Subproof**

(i)	$[\![\text{ S' }]\!]^g$	=	(by PA)
(ii)	$[\ \lambda x_e : \underline{[\![\text{ S }]\!]^{g(3/x)} = \text{T}} \ ]$	=	(by notation, q)
(iii)	$[\ \lambda x_e : \underline{\text{Michelle likes x}} \ ]$		

s. **Subproof**

(i)	$[\![\text{ CP }]\!]^g$	=	(by FA, i, r)
(ii)	$[\![\text{ DP}_3]\!]^g ([\![\text{ S' }]\!]^g)$	=	(by i, r)
(iii)	$[\lambda f_{<et>} : f]([\ \lambda x_e : \underline{\text{Michelle likes x}} \ ])$	=	(by LC)
(iv)	$[\ \lambda x_e : \underline{\text{Michelle likes x}} \ ]$		

t. **Subproof**

(i)	$[\![\text{ NP}_2]\!]^g$	=	(by PM, h, s)
(ii)	$[\ \lambda z_e : \underline{[\![\text{ N}_2]\!]^g (z) = \text{T and } [\![\text{ CP }]\!]^g (z) = \text{T}} \ ]$	=	(by h)
(iii)	$[\ \lambda z_e : \underline{[\lambda x_e : \text{x is a man }] (z) = \text{T and } [\![\text{ CP }]\!]^g (z) = \text{T}} \ ]$	=	(by LC)
(iv)	$[\ \lambda z_e : \underline{\text{z is a man and } [\![\text{ CP }]\!]^g (z) = \text{T}} \ ]$	=	(by s)
(v)	$[\ \lambda z_e : \underline{\text{z is a man and } [\ \lambda x_e : \text{Michelle likes x }] (z) = \text{T}} \ ]$	=	(by LC)
(vi)	$[\ \lambda z_e : \underline{\text{z is a man and Michelle likes z}} \ ]$		

u. **Subproof**

(i)	$[\![\text{ DP}_2]\!]^g$	=	(by FA, g, t)
(ii)	$[\![\text{ D}_2]\!]^g (\ [\![\text{ NP}_2]\!]^g)$	=	(by g, t)
(iii)	$[\lambda f_{<et>} : f] ([\lambda z_e : \underline{\text{z is a man and Michelle likes z}} \ ])$	=	(by LC)
(iv)	$[\lambda z_e : \underline{\text{z is a man and Michelle likes z}} \ ]$		

v. **Subproof**

(i)	$[\![\text{ VP}_1]\!]^g$	=	(by FA, f, u)
(ii)	$[\![\text{ V}_1]\!]^g ([\![\text{ DP}_2]\!]^g)$	=	(by f, u)
(iii)	$[\lambda f_{<et>} : f] ([\lambda z_e : \underline{\text{z is a man and Michelle likes z}} \ ])$	=	(by LC)
(iv)	$[\lambda z_e : \underline{\text{z is a man and Michelle likes z}} \ ]$		

w. $[\![\text{ S}_1]\!]^g = \text{T}$ *iff* (by FA, e, v)

x. $[\![\text{ VP}_1]\!]^g \ (\ [\![\text{ DP}_1]\!]^g) = \text{T}$ *iff* (by e, v)

y. $[\lambda z_e : \underline{\text{z is a man and Michelle likes z}} \ ]$ (Barack) = T *iff* (by LC)

z. Barack is a man and Michelle likes Barack

(37) **Summary Conclusion**

a. We assume that movement structures have a syntax like the following:

[XP_1 [1 […. t_1 ….]]]

- The index of the moved phrase is copied onto the sister of its sister
- The position originally occupied by the moved phrase is occupied by a *trace* which is co-indexed with the moved phrase.

b. We interpret moved structures according to our rule of PA, whereby:

[[[**1** […. $\boldsymbol{t_1}$ ….]]]] = [λx_e : …. x ….]

- The 'copied index' next to the moved phrase is interpreted as a lambda
- The trace of the moved phrase is interpreted as the variable bound by the lambda operator

c. **Thus, in our system, movement of a phrase XP has the following semantics:**
- **The sister of the moved phrase is interpreted as an <et> function**

d. This semantics for movement structures offers a straightforward semantics for:

- Topicalization Structures:
  The topicalized DP is argument to the <et> function created by movement
- Relative Clauses
  The relative operator is interpreted as an <et,et> identity function, which results in the relative clause itself being of type <et>.
  The relative clause then combines with the NP it modifies via our rule of PM.