The Semantics of Transitive Verbs

Lecture 07

February 1, 2024

Announcements:

This lecture is supplemented by the following readings:

- Heim & Kratzer: Ch.1 (26–29)

Your third homework assignment is available and is due on February 8th.

1 Introduction

We have up to this point built a semantic system that consists of a set of rules for compositionally computing the truth conditions of a sentence.

This system relies on the following set of rules:

(1) **Functional Application (FA)**
If X is a node that has two daughters, Y and Z, and if ⟦ Y ⟧ is a function whose domain contains ⟦ Z ⟧, then ⟦ X ⟧ = ⟦ Y ⟧(⟦ Z ⟧).

(2) **Non-Branching Nodes (NN) Rule**
If X is a non-branching node that has Y as its daughter, then ⟦ X ⟧ = ⟦ Y ⟧

(3) **Terminal Nodes (TN) Rule**
If X is a terminal node, then ⟦ X ⟧ is specified in the lexicon.

These rules are able to interpret structures built from lexical entries like the following:

(4) **Proper Nouns as entities**
⟦ NAME ⟧ = the thing referred to with NAME

(5) **Intransitive predicates as functions**
$\llbracket \text{VERB}_{intrans} \rrbracket = f : \{ x : x \text{ is an entity} \} \rightarrow \{ T, F \}$
for every $y \in \{ x : x \text{ is an entity} \}$, $f(y) = T$ *iff* y VERBs

This system can interpret a very large, although still narrow, range of sentences across many languages.

One of the thing that we will do today is introduce a useful new shorthand notation to represent domains and ranges that refer to the **semantic type** of an expression.

(6) **Some Semantic Types**

a. D_e = the set of entities

b. D_t = the set of truth values

c. $D_{\langle e,t\rangle}$ = the set of functions from entities to truth values

This comes with the practical benefit of more succinctly representing the meaning of the predicates, as shown below, and talking about their compositional potential.

(7) **Shorthand notation for predicates**

$[\![\text{dances}]\!] = f : D_e \rightarrow D_t$
for every $x \in D_e$, $f(x) = T$ *iff* x dances

This will also help us in our primary goal for today, which is to determine the meaning of transitive predicates, as in (8), and incorporate them into our system.

(8) Travis **likes** Taylor.

With our current rules of composition, we will find that we can incorportate such items fairly straightforwardly by modeling them as **function-valued functions**: functions that take an argument and return another function as their value.

(9) **Transitive predicates as function-valued functions**

$[\![\text{likes}]\!] = f : D_e \rightarrow D_{\langle e,t\rangle}$
for every $y \in D_e$, $f(y) = g : D_e \rightarrow D_t$
for every $x \in D_e$, $g(x) = T$ *iff* x likes y

"The function f from entities to functions from entities to truth values, which
for every entity x maps x to the function g from entities to truth values, which
for every entity y maps y to T *iff* y likes x."

2 Semantic Types

In addition to being a helpful shorthand, the notation for **semantic types** provides several benefits.

We will see that it provides an easy way to talk about the "types" of extensions that natural language expression can/should have.

Included as part of this will be the ability to visualize the composition of an expression.

2.1 Semantic Type Notation

Proper Nouns. Our semantic system treats the extensions of proper names like *Sam* as the entity that those names refer to.

(10) **Proper nouns as entities**
$[\![\text{NAME}]\!]$ = the thing referred to with NAME

In other words, the extensions of proper nouns are members of the set of entities:

(11) $[\![\text{Sam}]\!] \in \{ x : x \text{ is an entity} \}$

It is possible to refer to the set of entities by designating a variable for it:

(12) **The Set of Entities**
$D_e = \{ x : x \text{ is an entity} \}$ = the set of entities

This makes it possible to talk and represent more efficiently the kinds of meanings that entities have.

(13) **Referencing the semantic type of entities**
a. *Sam* is an expressions of **type *e***.
b. *Sam* is of **type *e***.
c. *Sam* is **type *e***.
d. $[\![\text{Sam}]\!] \in D_e$

Sentences. The extensions of sentences in our semantic system are their truth value.

(14) **Sentences as truth values**
$[\![\text{S}]\!]$ = Truth Value

Thus, the extension of sentences are members of the set of truth values:

(15) $[\![\text{Sam dances}]\!] \in \{ T, F \}$

Just as we did above, we can designate a variable to refer to the set of truth values:

(16) **The Set of Truth Values**
$D_t = \{ T, F \}$ = the set of truth values

This makes it possible to talk and represent more efficiently the kinds of meanings that sentences have.

(17) **Referencing the semantic type of sentences**
a. *Sam dances* is an expressions of **type *t***.
b. *Sam dances* is of **type *t***.
c. *Same dances* is **type *t***.
d. $[\![\text{Sam dances}]\!] \in D_t$

Intransitive Verbs. The extensions of intransitive verbs in our semantic system are functions from entities to truth values.

(18) **Intransitive predicates as functions**

$[\![\text{VERB}_{intrans}]\!] = f : \{ x : x \text{ is an entity} \} \rightarrow \{ T, F \}$
for every $y \in \{ x : x \text{ is an entity} \}$, $f(y) = T$ *iff* y VERBs

Adopting some of the shorthand introduced above, (18) can be equivalently represented as (19):

(19) **Intransitive predicates as functions (shorthand)**

$[\![\text{VERB}_{intrans}]\!] = f : D_e \rightarrow D_t$
for every $y \in D_e$, $f(y) = T$ *iff* y VERBs

The extension of an intransitive verb, therefore, is a member of the set of functions that map entities to truth values:

(20) $[\![\text{dances}]\!] \in \{ f : f \text{ is a function from entities to truth values} \}$

Once again, it is possible to designate a variable to refer to the set of such functions:

(21) **The Set of Functions from Entities to Truth Values**

$D_{\langle e,t \rangle} = \{ f : f \text{ is a function from entities to truth values} \}$
$=$ the set of functions from entities to truth values

(22) **Referencing the semantic type of intransitive verbs**

a. *dances* is an expressions of **type** $\langle \boldsymbol{e}, \boldsymbol{t} \rangle$.
b. *dances* is of **type** $\langle \boldsymbol{e}, \boldsymbol{t} \rangle$.
c. *dances* is **type** $\langle \boldsymbol{e}, \boldsymbol{t} \rangle$.
d. $[\![\text{dances}]\!] \in D_{\langle e,t \rangle}$

As we have seen above, this type of notation can be useful for classifying lexical items on the basis of their meaning.

(23) **Schematic of Notation for Functional Types**

$\langle$ *semantic-type-of-argument*, *semantic-type-of-value* $\rangle$

(24) **Examples of Functional Semantic Types**

a. $\langle e, t \rangle$ = a function that maps arguments of type e to values of type t
b. $\langle e, e \rangle$ = a function that maps arguments of type e to values of type e
c. $\langle t, t \rangle$ = a function that maps arguments of type t to values of type t
e. $\langle e, \langle e, t \rangle \rangle$ = a function that maps arguments of type e to functions of type $\langle e, t \rangle$

With this notation in hand, it will be possible to designate a variable for expressions of any basic type or any functional type:

(25) **Schematic Notation for Sets of Particular Types**

D_x = the set of expressions of type x

$D_{\langle x,y\rangle}$ = the set of functions of type $\langle x, y\rangle$ with arguments of type x and values of type y

As we will see in what follows, this notation will also be especially useful for reasoning out how the semantic derivation for a particular tree precedes. This will in turn be useful for determining the meaning of new kinds of lexical entry in our semantic system.

2.2 Semantic Types and Composition

To get a sense of how we can reason out the semantic types, let's look at a familiar sentence with an intransitive verb.

(26) Sam dances

1. Known Semantic Types. We can start by listing the semantic types that we know for the lexical items and the semantic type that we know the entire sentence must end up with:

(27) **Known Semantic Types in (26)**

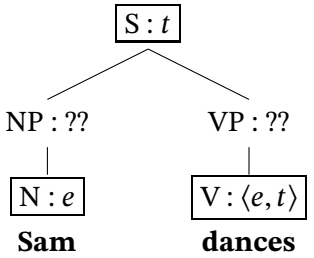
a. $[\![\text{S}]\!] \in D_t$

b. $[\![\text{Sam}]\!] \in D_e$

c. $[\![\text{dances}]\!] \in D_{\langle e,t\rangle}$

With this information, we can annotate the syntactic representation with the types for each of these nodes.

(28)



2. Reasoning out Unknown Semantic Types. It then becomes possible to reason out any unknown semantic types by appealing to our rules of composition.

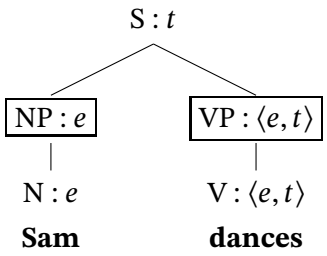
In the case at hand, the NP and VP nodes are of the form specified by the **Non-Branching Nodes Rule**.

(29) **Non-Branching Nodes (NN) Rule**

If X is a non-branching node that has Y as its daughter, then $[\![\text{X}]\!] = [\![\text{Y}]\!]$

Therefore, ⟦ VP ⟧ = ⟦ V ⟧ and ⟦ NP ⟧ = ⟦ N ⟧. This means that the extensions of the terminal nodes and, consequently, their semantic type will be passed up the tree.

(30)



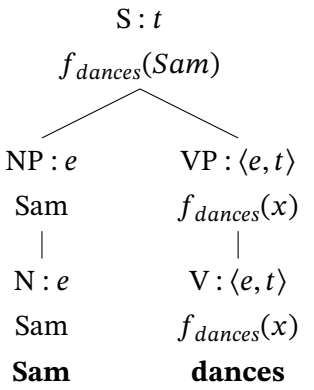
3. Visualizing Composition. The S node is of the form specified by **Functional Application**.

(31) **Functional Application (FA)**
If X is a node that has two daughters, Y and Z, and if ⟦ Y ⟧ is a function whose domain contains ⟦ Z ⟧, then ⟦ X ⟧ = ⟦ Y ⟧(⟦ Z ⟧).

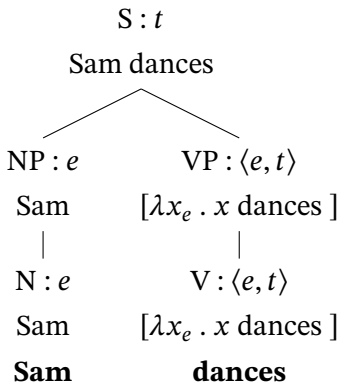
The VP denotes a function that takes an entity as its argument and returns a truth value. And the NP denotes an entity. They are, therefore, correctly expected to compose to create the node S, which denotes a truth value, and is of type t.

It can also be useful to annotate each node with its extension. This is a technique that is commonly used to demonstrate how the meaning of the sentence is computed in place of a full semantic proof:

(32) **Informal Function Notation**



(33) **Lambda Notation**



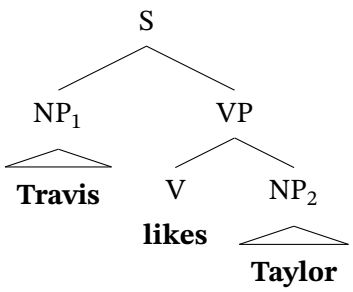
In addition to helping us visualize the composition of an expression, these kinds of representations will also allow us to reason out the meaning of novel expressions.

Let's see how.

3 The Semantics of Transitive Verbs

We are ready now to ask whether our current system can interpret structures with **transitive** verbs.

(34) a. Travis likes Taylor.

b. 

3.1 The Semantic Type of Transitive Verbs

We don't know the lexical entry for *likes*. So, we might start by asking what type of expression *likes* is.

(35) **Current Semantic Types of Expressions**

a. e
b. t
c. $\langle e, t \rangle$

This is where we run into a problem:

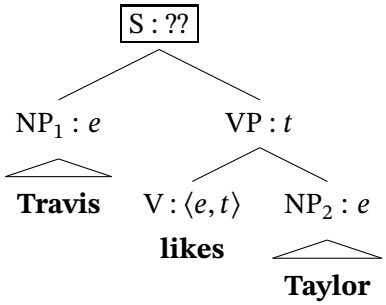
- Given our syntactic assumptions above and our current rules of composition (*viz.*, FA), the extension of *likes* and the extension of *Taylor* must combine to give us the extension of the VP.

  (36) $[\![\text{likes}]\!] + [\![\text{Taylor}]\!] = [\![\text{VP}]\!]$.

- We know that the NP *Taylor* is type e, so the extension of *likes* must be some function that has D_e as its domain.

  (37) $f : D_e \rightarrow ??$

- The only such functional semantic type in our current system is $\langle e, t \rangle$.
- But if *likes* were type $\langle e, t \rangle$, the VP *likes Taylor* would be of type t, meaning it couldn't then combine with the type e expression *Travis*. The S node is, at present, uninterpretable.

(38) 

Therefore, the set of expressions represented by the semantic types in (35) is insufficient for interpreting sentences with transitive verbs.

We need to introduce a new semantic type of expression in order to develop a lexical entry for the verb *likes* and other transitive verbs.

We can reason out what the semantic type of *likes*—and other transitive verbs—will have to have by going through the same process as above.

1. Known Semantic Types. We can start by listing out the semantic types we know and annotating the syntactic representation with this information

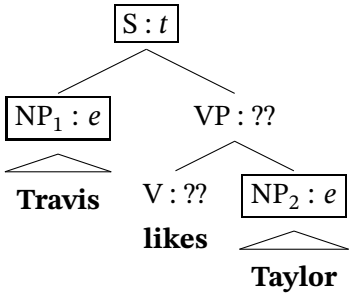
(39) **Known Semantic Types in (31a)**

a. $[\![\text{S}]\!] \in D_t$

b. $[\![\text{Travis}]\!] \in D_e$

c. $[\![\text{Taylor}]\!] \in D_e$

(40)



2. Reasoning out the Semantic Type of VP. We can appeal to these known semantic types, as well as the rules of composition, to reason out what the semantic type of the VP must be.

- The sentence must ultimately be an expression of type t.
- The subject NP_1 *Travis* is an expression of type e.
- The S node fits the description for Functional Application. So, the extension of the subject must combine with the extension of the VP to return the extension of the sentence.

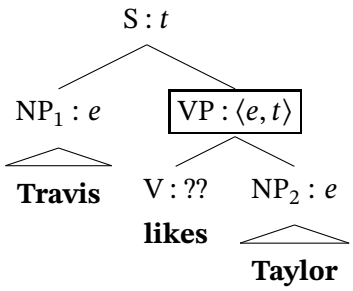
  (41) $[\![\text{Travis}]\!] + [\![\text{VP}]\!] = [\![\text{S}]\!]$

- Because NP_1 *Travis* is an expression of type e, the extension of **the VP must be a function that takes *Travis* as an argument and gives back a truth value**.

  (42) $f : D_e \rightarrow D_t$

- Thus, the VP must be an expression of type $\langle e, t \rangle$ (just like an intransitive verb).

(43)



3. Reasoning out the Type of V. We repeat the process to reason out the semantic type of the V.

- The VP *likes Taylor* was just determined to be an expression of type $\langle e, t \rangle$.
- The object NP_2 *Taylor* is an expression of type e.
- The VP node fits the description for Functional Application. So, the extension of the verb *likes* must combine with the extension of the object *Taylor* to return the extension of the VP.

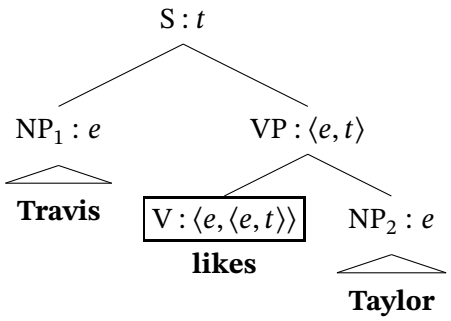
(44) $[\![\text{likes}]\!] + [\![\text{Taylor}]\!] = [\![\text{VP}]\!]$

- Because NP_2 *Taylor* is an expression of type e, the extension of **the V must be a function that takes *Taylor* as an argument and gives back a function from entities to truth values**.

(45) $f : D_e \rightarrow D_{\langle e,t \rangle}$

- Thus, the V must be an expression of type $\langle e, \langle e, t \rangle \rangle$.

(46)



We now know, abstractly speaking, what kind of function the V *likes* must have as its extension:

(47) **Type $\langle e, \langle e, t \rangle \rangle$ Function**
a function that maps entities to
functions that map entities to truth values

Such functions, which take an entity as an argument and then return another function, can be referred to as **function-valued functions**.

We will find moving forward that such functions are an essential tool of our semantic system.

For now, this discovery informs our path forward: we need to develop a lexical entry for *likes* that does the following:

- assigns as its extension a function of type $\langle e, \langle e, t \rangle \rangle$ and
- allows our system to derive the following truth-conditional statement:

(48) "Travis likes Taylor" is *T iff* Travis likes Taylor

3.2 The Extension of Transitive Verbs

Recall our method for determining the extension of a lexical item. We have a specific investigative methodology that looks for a systematic form-meaning correspondence:

(49) **Determining the Meaning of Lexical Items**

i.) We identify the truth conditions of sentences in which a particular lexical item appears.

ii.) We consider how the lexical item systematically contributes to the truth conditions of those sentences (i.e., a form-meaning correspondence).

iii.) We consider the known extensions of the other lexical items in these sentences.

iv.) We consider the available rules of composition in our system (FA, NN, TN).

v.) Then, we develop a lexical entry that correctly derives the truth condition of these sentences.

Let's see now how this process can be used to deduce the extension of the transitive verb *likes*. It will involve working backwards, step-by-step, from the extension of the VP to the extension of the V.

1. Reasoning out the Extension of the VP *likes Taylor*. Let's start by figuring out the meaning of the type $\langle e, t \rangle$ VP *likes Taylor*.

- We first gather up some more data and consider the truth conditions of other sentences that contain the meaning of the VP *likes Taylor*.

  (50) a. "Sam likes Taylor" is *T iff* Sam likes Taylor
  b. "Kuniko likes Taylor" is *T iff* Kuniko likes Taylor
  c. "Mike likes Taylor" is *T iff* Mike likes Taylor

- The key generalization to be appreciated here is that *likes Taylor* systematically combines with entities to generate sentences with meanings of the type below:

  (51) $[\![[_{\text{S}} \text{ NAME likes Taylor }]]\!] = T$ *iff* NAME likes Taylor

- We know from section 3.1 that $[\![$ likes Taylor $]\!]$ must be of type $\langle e, t \rangle$. Consequently, our rule of FA entails that the following equivalency holds:

  (52) $[\![[_{\text{S}} \text{ NAME likes Taylor }]]\!] = [\![[_{\text{VP}} \text{ likes Taylor }]]\!]([\![\text{ NAME }]\!])$

- It follows from the previous two points that:

  (53) $[\![[_{\text{VP}} \text{ likes Taylor }]]\!]([\![\text{ NAME }]\!]) = T$ *iff* NAME likes Taylor

- Therefore, $[\![$ likes Taylor $]\!]$ is a function which takes some entity $\underline{\boldsymbol{x}}$ as its argument and yields *T iff* x likes Taylor.

  (54) $[\![[_{\text{VP}} \text{ likes Taylor }]]\!] = g : D_e \rightarrow D_t$
  for every $x \in D_e$, $g(x) = T$ *iff* $\underline{\boldsymbol{x}}$ likes Taylor

So, we have deduced the extension of the VP *likes Taylor*. But, because this is a phrase derived in the syntax, this meaning can't simply be listed in the lexicon.

We need our system to derive the extension of *likes Taylor* from the extension of the component parts *likes* and *Taylor*. But we can now use the information above to reason our the extension of *likes*.

2. Reasoning out the Extension of the V *likes*. Let's use these results to determine the meaning of the V *likes*.

- We first gather up some more data and consider the truth conditions of other sentences that have VPs containing the V *likes*.

  (55) a. "Travis likes Sam" is *T iff* Travis likes Sam
  b. "Travis likes Kuniko" is *T iff* Travis likes Kuniko
  c. "Travis likes Mike" is *T iff* Travis likes Mike

- With the results above in hand, these data reveal that *likes* combines with entities to generate each of the VP meanings below:

  (56) a. $[\![\ [_{\text{VP}}\ \text{likes Sam}\ ]\ ]\!] = h : D_e \rightarrow D_t$
  for every $x \in D_e$, $h(x) = T$ *iff* x likes Sam
  b. $[\![\ [_{\text{VP}}\ \text{likes Kuniko}\ ]\ ]\!] = i : D_e \rightarrow D_t$
  for every $x \in D_e$, $i(x) = T$ *iff* x likes Kuniko
  c. $[\![\ [_{\text{VP}}\ \text{likes Mike}\ ]\ ]\!] = j : D_e \rightarrow D_t$
  for every $x \in D_e$, $j(x) = T$ *iff* x likes Mike

- The key generalization to be appreciated here is that *likes* systematically combines with entities to generate VPs with meanings of the type below:

  (57) $[\![\ [_{\text{VP}}\ \text{likes NAME}\ ]\ ]\!] = g : D_e \rightarrow D_t$
  for every $x \in D_e$, $g(x) = T$ *iff* x likes NAME

- We know from section 3.1 that $[\![\ \text{likes}\ ]\!]$ is of type $\langle e, \langle e, t\rangle\rangle$. Consequently, our rule of FA entails that the following equivalency holds:

  (58) $[\![\ [_{\text{VP}}\ \text{likes NAME}\ ]\ ]\!] = [\![\ \text{likes}\ ]\!]([\![\ \text{NAME}\ ]\!])$

- It follows from the previous two points that:

  (59) $[\![\ \text{likes}\ ]\!]([\![\ \text{NAME}\ ]\!]) = g : D_e \rightarrow D_t$
  for every $x \in D_e$, $g(x) = T$ *iff* x likes NAME

- Therefore, $[\![\ \text{likes}\ ]\!]$ is a function which takes some entity $\underline{\boldsymbol{y}}$ as its argument and yields the following function as its value:

  (60) $g : D_e \rightarrow D_t$
  for every $x \in D_e$, $f(x) = T$ *iff* x likes $\underline{\boldsymbol{y}}$

3. Putting Everything Together. We can now provide a lexical entry for the transitive verb *likes*.

We saw in section 3.1 that *likes* must be a function of type $\langle e, \langle e, t\rangle\rangle$ in order to ensure composition given our current semantic rules. Just as we did above, we can designate a variable to refer to this new kind of semantic type:

(61) **The Set of Functions from Entities to Function from Entities to Truth Values**

$D_{\langle e,\langle e,t\rangle\rangle} = \{\ f\ :\ f$ is a function from entities to function from entities to truth values $\}$
$=$ the set of functions from entities to function from entities to truth values

And we can use this notation to represent and refer to such expressions more efficiently:

(62) **Referencing the semantic type of transitive verbs**

a. *likes* is an expressions of **type** $\langle \boldsymbol{e}, \langle \boldsymbol{e}, \boldsymbol{t}\rangle\rangle$.
b. *likes* is of **type** $\langle \boldsymbol{e}, \langle \boldsymbol{e}, \boldsymbol{t}\rangle\rangle$.
c. *likes* is **type** $\langle \boldsymbol{e}, \langle \boldsymbol{e}, \boldsymbol{t}\rangle\rangle$.
d. $[\![\ \text{likes}\ ]\!] \in D_{\langle e,\langle e,t\rangle\rangle}$

We saw immediately above in Part 2 of section 3.2 that, as a V of type $\langle e, \langle e, t\rangle\rangle$, $[\![\ \text{likes}\ ]\!]$ is a function that takes an entity $\underline{\boldsymbol{y}}$ as its argument and returns a function as its value.

(63) **Part 1 of the Extension for *Likes***

$[\![\ \text{likes}\ ]\!] = f_{likes}\ :\ D_e \rightarrow D_{\langle e,t\rangle}$
for every $y \in D_e,\ f(y) = g$

We saw in Part 1 of section 3.2 that the returned function (i.e., the VP) is a function that takes an entity $\underline{\boldsymbol{x}}$ as its argument and returns the value *T iff x* likes *y*.

(64) **Part 2 of the Extension for *Likes***

$f_{likes}(y) = g\ :\ D_e \rightarrow D_t$
for every $x \in D_e,\ g(x) =$ *T iff* $\underline{\boldsymbol{x}}$ likes $\underline{\boldsymbol{y}}$

When we combine these parts together, we get the following lexical entry for *likes*:

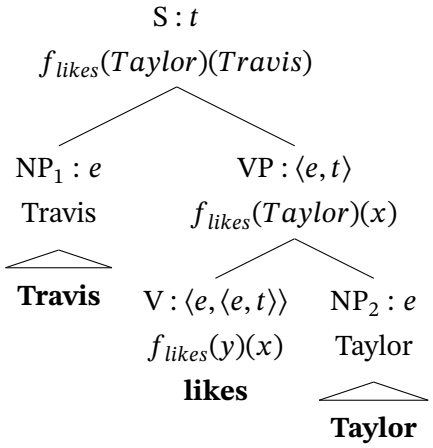
(65) **The extension of *likes***

$[\![\ \text{likes}\ ]\!] = f_{likes}\ :\ D_e \rightarrow D_{\langle e,t\rangle}$
for every $y \in D_e,\ f(y) = g\ :\ D_e \rightarrow D_t$
for every $x \in D_e,\ g(x) =$ *T iff x* likes *y*

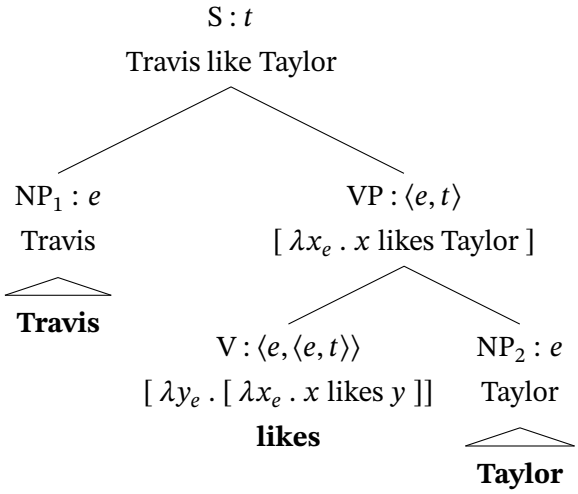
"The function f from entities to functions from entities to truth values, which
for every entity x maps x to the function g from entities to truth values, which
for every entity y maps y to *T iff y* likes x."

We can visualize how the composition of the sentence is intended to work.

(66) **Informal Function Notation**



(67) **Lambda Notation**



In sum, the extension of a transitive verb is a function that takes an entity (the object), and returns a function that takes an entity (the subject) and returns a truth-value.

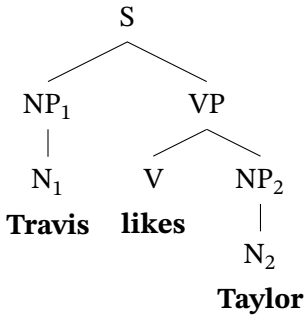
4 Deriving the Meaning of a Transitive Sentence

Now that we know the meaning of *likes*, we can compute the meaning of our target sentence.

We can start by making clear our syntactic assumptions regarding the structure of the sentence:

(68) a. Travis likes Taylor

b.



The lexical entries for the components parts of the sentence that are stored in the lexicon include:

(69) **Lexical entries**

a. $[\![\text{Travis}]\!] = \text{Travis}$

b. $[\![\text{Taylor}]\!] = \text{Taylor}$

c. $[\![\text{likes}]\!] = f_{likes} : D_e \rightarrow D_{\langle e,t \rangle}$
for every $y \in D_e$, $f(y) = g : D_e \rightarrow D_t$
for every $x \in D_e$, $g(x) = T$ *iff* x likes y

Our system currently employs the set of semantic rules listed below:

(70) **Functional Application (FA)**
If X is a node that has two daughters, Y and Z, and if ⟦ Y ⟧ is a function whose domain contains ⟦ Z ⟧, then ⟦ X ⟧ = ⟦ Y ⟧(⟦ Z ⟧).

(71) **Non-Branching Nodes (NN) Rule**
If X is a non-branching node that has Y as its daughter, then ⟦ X ⟧ = ⟦ Y ⟧

(72) **Terminal Nodes (TN) Rule**
If X is a terminal node, then ⟦ X ⟧ is specified in the lexicon.

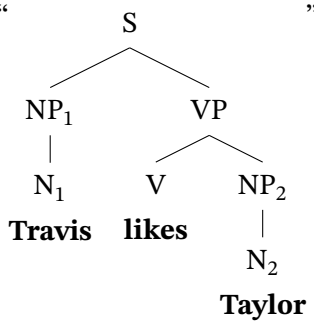
We can now see how the proof of the truth conditions can proceed with subproofs of the major constituents of the sentence.

(73) **Truth-conditional Statement of *Travis likes Taylor***
Travis likes Taylor is *T iff* Travis likes Taylor

(74) **Calculation of the Truth Conditions of *Travis likes Taylor***

i. "Travis likes Taylor" is *T iff* (by syntax)

ii. " " is *T iff* (by notation)



iii. ⟦ S ⟧ = *T*

iv. *Calculation of* ⟦ NP_1 ⟧

a. ⟦ NP_1 ⟧ = (by NN)

b. ⟦ Travis ⟧ = (by TN)

c. Travis

v. *Calculation of* ⟦ NP_2 ⟧

a. ⟦ NP_2 ⟧ = (by NN)

b. ⟦ Taylor ⟧ = (by TN)

c. Taylor

vi. *Calculation of* ⟦ VP ⟧

a. ⟦ VP ⟧ = (by FA, v.)

b. ⟦ V ⟧(⟦ NP_2 ⟧) = (by TN)

c. f_{likes}(⟦ NP_2 ⟧) = (by v.)

d. f_{likes}(Taylor) = (by def. of f_{likes})

e. $g_{VP} : D_e \rightarrow D_t$
for every $x \in D_e$, $g_{VP}(x) = T$ *iff* x likes Taylor

vii. ⟦ S ⟧ = T *iff* (by FA, iv., vi.)

viii. ⟦ VP ⟧(⟦ NP_1 ⟧) = T *iff* (by v.)

ix. ⟦ VP ⟧(Travis) = T *iff* (by vi.)

x. g_{VP}(Travis) = T *iff* (by def. of g_{VP})

xi. Travis likes Taylor (nailed it)

Thus, we conclude that (64i) *iff* (64xi), or **"Travis likes Taylor" is *T iff* Travis likes Taylor**.

By repeating the process made explicit in this handout, we would find that a similar lexical entry would work for a wide range of verbs in English and beyond. For example, this system will similarly derive the following truth-conditional statements for several transitive verbs, which seem to be correct:

(75) a. "Travis **sees** Taylor" is *T iff* Travis **sees** Taylor.
b. "Travis **invites** Taylor" is *T iff* Travis **invites** Taylor.
c. "Travis **hates** Taylor" is *T iff* Travis **hates** Taylor.

As mentioned previously, the approach we have taken to semantics admittedly does not contribute much to the meaning of content words like *dances*, *sings*, *likes*, or even *desk*. Indeed, it makes it possible to largely ignore much of their meaning, which is arguably a weakness of our approach. But what's in a name?

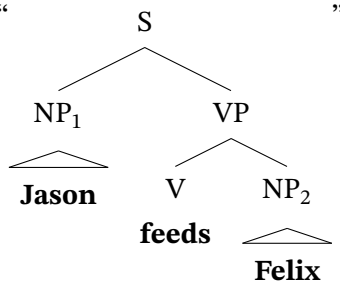
Also, the lexical entry for transitive verbs is really cumbersome! Let's find an easier way to do this:

(76) **Function Notation**
⟦ likes ⟧ = $f_{likes} : D_e \rightarrow D_{\langle e,t \rangle}$
for every $y \in D_e$, $f(y) = g : D_e \rightarrow D_t$
for every $x \in D_e$, $g(x) = T$ *iff* x likes y

(77) **Lambda Notation**
⟦ likes ⟧ = [$\lambda y : y \in D_e$. [$\lambda x : x \in D_e$. T *iff* x likes y]]
= [λy_e . [λx_e . x likes y]]

5 Practice

Please provide a semantic proof of the truth-conditional statement below:

(78) “ 

” is *T iff* Jason feeds Felix

In order to do this, we will need to provide the extensions of the lexical items and provide a step-by-step proof of these reported truth conditions.