Expanding Our Formalism, Part 1[1]

1. Review of Our System Thus Far

Thus far, we've built a system that can interpret a very narrow range of English structures: sentences whose subjects are proper names, and whose Vs are intransitive.

(1) **Main Components of the System Thus Far**

a. Rules:
(i) Function Application (FA)
(ii) Non-Terminal Nodes (NN)
(iii) Terminal Nodes (TN)

b. Lexical Entries (Primitives):

Names
(i) [[Barack]] = Barack
(ii) [[Joe]] = Joe

Intransitive Verbs
(iii) [[smokes]] = f : { x : x is an entity } → { T, F }
for every y ∈ { x : x is an entity }, f(y) = T *iff* y smokes

(iv) [[dances]] = h: { x : x is an entity } → { T, F }
for every y ∈ { x : x is an entity }, h(y) = T *iff* y dances

(2) **A Helpful New Shorthand**

a. D_e = { x : x is an entity } *the 'domain' of entities*

b. D_t = { T, F } *the 'domain' of truth-values*

Note: This use of the term 'domain' is different from its use when we are defining a function.

(3) **Illustration of the Shorthand**

a. [[smokes]] = f: $D_e \rightarrow D_t$
for every x ∈ D_e, f(x) = T *iff* x smokes

b. [[dances]] = h: $D_e \rightarrow D_t$
for every x ∈ D_e, h(x) = T *iff* x dances

[1] These notes are based on the material in Heim & Kratzer (1998: 26-29).

2. The System of Semantic Types

Besides being a helpful shorthand, the notation in (2) gives us an easy way to talk about the kinds – or *types* – of extensions that natural language expressions can have.

(4) **Example: The Semantic Type of Proper Names**

a. What We Already Know:

(i) Proper names like "Barack" have **entities** as their extension.

(ii) "Barack" has an **entity** as its extension.

b. Some 'Fancy' New Ways of Saying This:

(i) *Proper names like "Barack" are of* ***type e***

(ii) *"Barack" is an expression of* ***type e***

(iii) [[Barack]] $\in D_e$

(5) **Example: The Semantic Type of Sentences**

a. What We Already Know:

(i) Sentences like "Barack smokes" have **truth-values** as their extension.

(ii) "Barack smokes" has a **truth-value** as its extension.

b. Some 'Fancy' New Ways of Saying This:

(i) *Sentences like "Barack smokes" are of* ***type t***

(ii) *"Barack smokes" is an expression of* ***type t***

(iii) [[Barack smokes]] $\in D_t$

Question: What about intransitive verbs like "smoke"? What 'type' are they?

(6) **Another Helpful New Shorthand**

$D_{<e,t>}$ = all the logically possible functions from entities to truth-values

the 'domain' of functions from entities to truth-values

(6) **The Semantic Type of Intransitive Verbs**

a. <u>What We Already Know:</u>

(i) Intransitive verbs like "smokes" have as their extensions **functions from entities to truth-values.**

(ii) "Smokes" has as its extension **a function from entities to truth-values.**

b. <u>Some 'Fancy' New Ways of Saying This:</u>

(i) *Intransitive verbs like "smokes" are of* ***type <e,t>***

(ii) *"Smokes" is an expression of* ***type <e,t>***

(iii) [[smokes]] $\in D_{<e,t>}$

(7) **A Schematic of the Notation for 'Functional' Types**

< *Semantic-Type-of-Argument* , *Semantic-Type-of-Value* >

Examples:

a. <e,t> = the type of function whose arguments are of type e, and whose values are of type t (*e.g.* [[smokes]])

b. <t,t> = the type of function whose arguments are of type t, and whose values are of type t (*e.g.* [[it is not the case that]])

c. <t,e> = the type of function whose arguments are of type t, and whose values are of type e (*doesn't correspond to a meaning in language*)

d. <e,<e,t>> = the type of a function whose arguments are of type e, and whose values are of type <e,t>
(as we will see, this is the extension of a *transitive* verb…)

<u>What is the point of all this?</u>

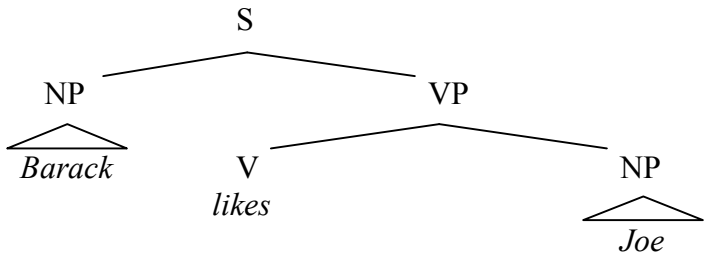
- This notation is useful for working out semantic problems
- It's especially useful for illustrating ***or reasoning about*** how the semantic derivation for a particular syntactic tree proceeds…

… as we will see presently!

3. The Semantics of Transitive Verbs

Question: Can our system interpret the syntactic structure below, where the main V of the clause is a *transitive* V?

(8) **Sentence Containing a Transitive Verb**



Problem 1: We don't have a lexical entry for *likes*.

Sub-Question: Which of the following semantic types (the ones currently in our system) is the semantic type of *likes*?

(9) **Our Current Semantic Types**
- a. Type *e*
- b. Type *t*
- c. Type <*e,t*>

Problem 2: None of the semantic types in (9) will work as the semantic type for *likes*!

- The extension of *likes* must combine with the extension of *Joe* to give us the extension of the VP *likes Joe*.
- The NP *Joe* is of type *e*, so the extension of *likes* must be some function that has D_e as its domain.
- In (9), the only such functional semantic type is <e,t>.
- But, if *likes* were type <e,t>, then the whole VP *likes Joe* would be of type *t*, and so it couldn't combine with the meaning of the subject *Barack* (which is type *e*).

Conclusion:

- The set of types in (9) is not enough to interpret transitive sentences of English.
- We have to introduce a new semantic type in order to write out a lexical entry for *likes*.

OK… But, what 'type' would work?

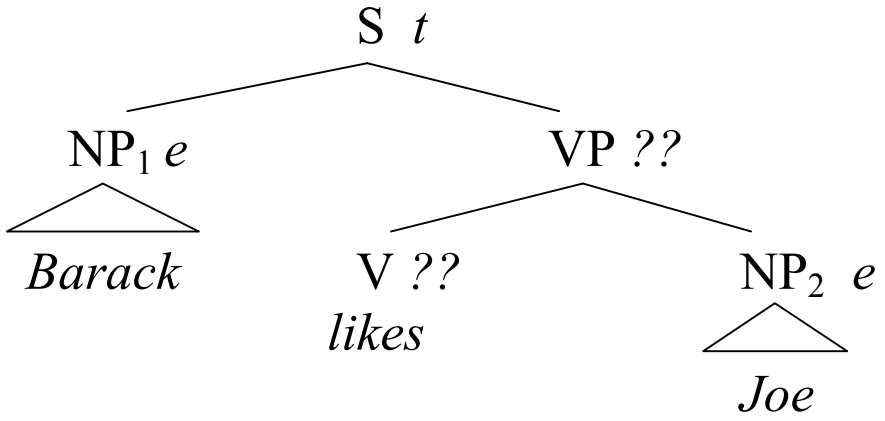
(10) **Reasoning Through the Type Assignments**

Here are the semantic type assignments we already know

[[S]] ∈ D_t ; [[Barack]] ∈ D_e ; [[Joe]]] ∈ D_e

Here are the semantic type assignments we don't yet know

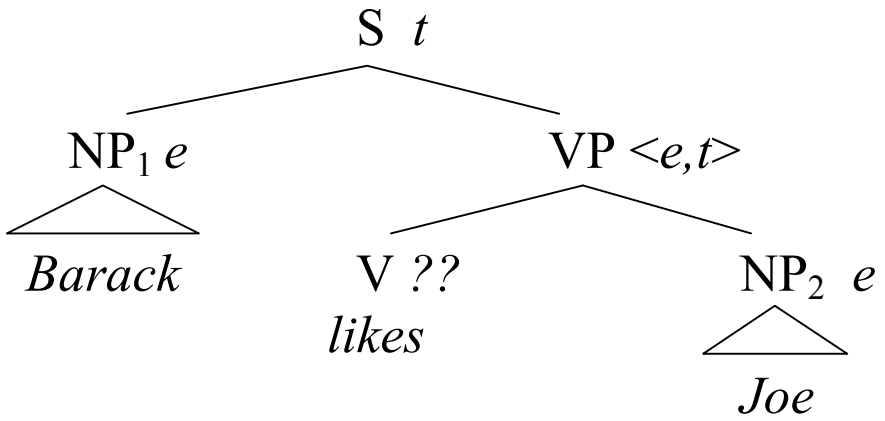
[[VP]] ∈ ?? ; [[V]] ∈ ??



(11) **Reasoning Out the Semantic Type of the VP**

- The entire sentence is type *t*
- The subject NP *Barack* is type *e*
- Now… the extension of the VP must 'combine' with the extension of the subject to give us the extension of the whole sentence…
- The only rule we have for such 'combination' is Function Application (FA)…
- So, [[VP]] has to be a function that takes *Barack* and gives back a T-value...

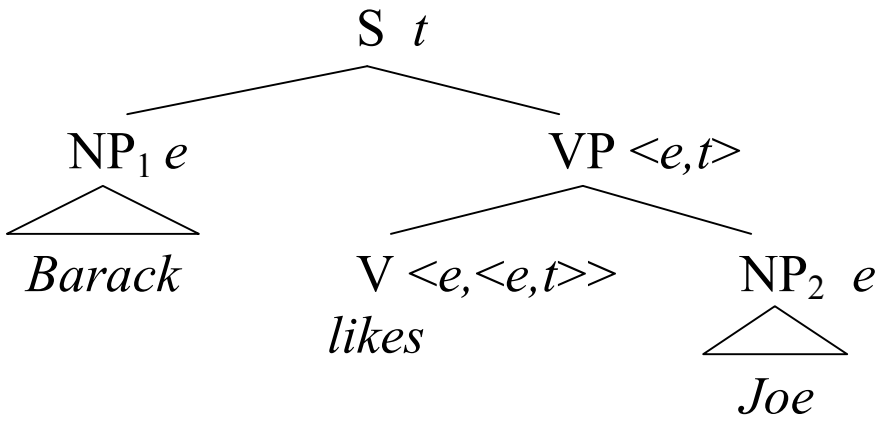
- **THEREFORE, the semantic type of the VP must be <*e,t*> (like an intransitive V)**



(12) **Reasoning Out the Type of the V *Likes***

- The entire VP *likes Joe* is of type <*e,t*>
- The direct object *Joe* is of type *e*
- Now, the extension of the V *likes* must 'combine' with the extension of the direct object *Joe* to give us the extension of the whole VP…
- The only rule we have for such 'combination' is FA…
- So, [[V]] has to be a function which takes *Joe* and gives back [[VP]] (of type <et>)

- **THEREFORE, the semantic type of the transitive V *likes* must be <e <e,t>>**



Conclusion:

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

a function from entities to
functions from entities to truth-values

(13) **New Goal**

Develop a lexical entry for *likes* that does the following:

a. Stipulates its extension as a function of type <*e, <e,t*>>

b. Allows our system to derive the following truth-conditional statement…

"Barack likes Joe" is T iff Barack likes Joe.

How do we do this?
Let's work backwards, step-by-step, starting with the larger VP "likes Joe"….

(14) **Reasoning Out the Extension of the VP *likes Joe***

a. Let's start off by considering the T-conditions of some other sentences containing the VP "*likes Joe*".

(i) "Mitch likes Joe" is T iff Mitch likes Joe
(ii) "Seth likes Joe" is T iff Seth likes Joe
(iii) "Rachael likes Joe" is T iff Rachael likes Joe

b. KEY GENERALIZATION

[[NAME likes Joe]] = T iff NAME likes Joe

c. KEY DEDUCTION

- We've already concluded that [[likes Joe]] is of type <e,t>
- Consequently, our rule of FA entails that
  [[NAME likes Joe]] = [[likes Joe]]([[NAME]])
- But, given our 'Key Generalization' in (14b) above, it follows that:
  [[likes Joe]]([[NAME]]) = T iff NAME likes Joe
- **KEY CONCLUSION**

  [[likes Joe]] is a function which takes as argument some entity x, and yields T iff x likes Joe.

d. DEDUCED SEMANTICS FOR THE VP

[[likes Joe]] = g: $D_e \rightarrow D_t$
for every $x \in D_e$, g(x) = T *iff* x likes Joe

So, we've deduced what the extension of the VP *likes Joe* is…
… but this equation can't just be stipulated in our lexicon.

***Instead, we need our system to derive* the equation in (14d) from the meaning (extension) of "likes" and the meaning (extension) of "Joe"…**

(15) **Reasoning Out the Extension of Extension of the V *likes***

a. Let's start off by considering the T-conditions of some other sentences that contain the V "likes"

(i) "Barack likes Seth" is T iff Barack likes Seth
(ii) "Joe likes Seth" is T iff Joe likes Seth

(iii) "Barack likes Nancy" is T iff Barack likes Nancy
(iv) "Joe likes Nancy" is T iff Joe likes Nancy

b. GENERALIZATIONS

(i) Given exactly the reasoning laid out in (14) earlier, we can deduce from the facts in (15ai) and (15aii) that:

[[likes Seth]] = i: $D_e \rightarrow D_t$
for every $x \in D_e$, i(x) = T *iff* x likes Seth

(ii) Given exactly the reasoning laid out in (14) earlier, we can deduce from the facts in (15aiii) and (15aiv) that:

[[likes Nancy]] = j: $D_e \rightarrow D_t$
for every $x \in D_e$, j(x) = T *iff* x likes Nancy

c. KEY GENERALIZATION

[[likes NAME]] = k: $D_e \rightarrow D_t$
for every $x \in D_e$, k(x) = T *iff* x likes NAME

d. KEY DEDUCTION

- We've already concluded that [[likes]] is of type <e,<e,t>>
- Consequently, FA entails that [[likes NAME]] = [[likes]]([[NAME]])
- Thus, given our 'Key Generalization' in (15c), it follows that:

[[likes]]([[NAME]]) = k: $D_e \rightarrow D_t$
for every $x \in D_e$, k(x) = T *iff* x likes NAME

- **KEY CONCLUSION**
[[likes]] is a function which takes as argument some entity y, and yields the following function:
f: $D_e \rightarrow D_t$
for every $x \in D_e$, f(x) = T *iff* x likes y

(16) **Deduced Semantics for the Transitive Verb *Likes***

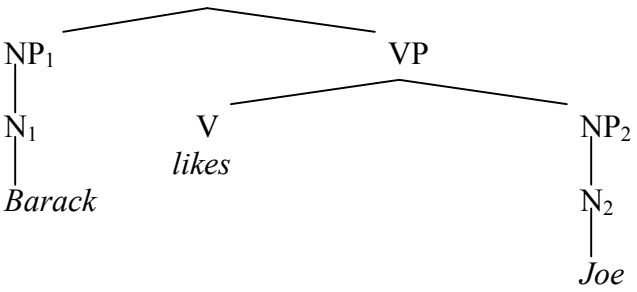
[[likes]] = g: $D_e \rightarrow D_{<e,t>}$
for every $x \in D_e$, g(x) = $i_x : D_e \rightarrow D_t$
for every $y \in D_e$, $i_x(y)$ = T iff y likes x

"the function g from entities to functions from entities to T-values, which
for every entity x, maps x to the function i_x from entities to T-values, which
for every entity y maps y to T iff y likes x"

With this lexical entry for "likes", we can now derive the T-conditions of the transitive sentence "*Barack likes Joe*"!

(17) **Derivation of the Truth-Conditions of "*Barack likes Joe*"**

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



b. [[S]] = T

c. **Subproof**

(i) [[NP_1]] = (by NN)

(ii) [[N_1]] = (by NN)

(iii) [[Barack]] = (by TN)

(iv) Barack

d. **Subproof**

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

(ii) [[N_2]] = (by NN)

(iii) [[Joe]] = (by TN)

(iv) Joe

e. **Subproof**

(i) [[V]] = (by NN)

(ii) [[likes]] = (by TN)

(iii) g

f. **Subproof**

(i) [[VP]] = (by FA, d, e)

(ii) [[V]]([[NP_2]]) = (by e)

(iii) g([[NP_2]]) = (by d)

(iv) g(Joe) = (by def. of 'g')

(v) i_{Joe}: $D_e \rightarrow D_t$
for every $y \in D_e$, $i_{Joe}(y)$ = T iff y likes Joe

g. [[S]] = T *iff* (by FA, c, f)

h. [[VP]]([[NP_1]]) = T*iff* (by c)

i. [[VP]](Barack) = T *iff* (by f)

j. i_{Joe} (Barack) = T *iff* (by def. of 'i_{Joe}')

k. Barack likes Joe

PROVEN: "Barack likes Joe" is T *iff* Barack likes Joe

The Upshot:

- In order to capture the semantics of transitive verbs, we must expand our system of types for natural language, so that it also includes functions of type <*e*, <*e, t*>>
- That is, we can model the extensions of transitive verbs as functions of type <*e,* <*e, t*>>

The extension of a transitive verb is a function that takes an entity (the direct object), and returns a function that takes an entity (the subject) and returns a truth-value!

---

4. The Semantics of "Or"

(18) **Key Feature of Our Semantics for Transitive Verbs**

The semantic type of a transitive verb is a function which takes entities as arguments…
*and then delivers **another function** as value (function from entities to T-values)…*

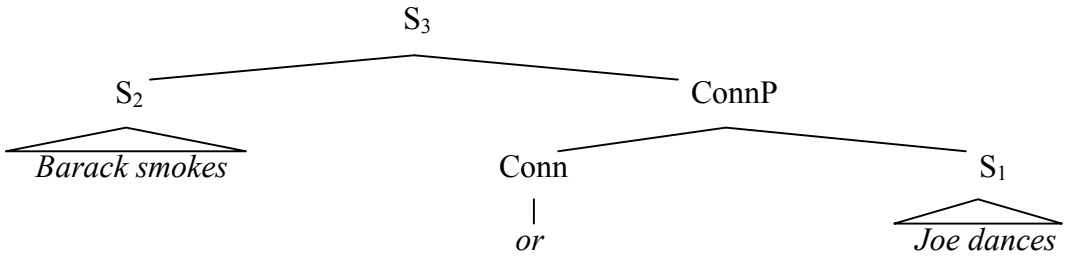
FACT: Such 'function-valued functions' will play a *huge* role in our overall system. Such functions can do a lot of work besides simply model transitive verbs.

(19) **Example: The Logical Connective "Or"**

We'll see in this section how the extension of the logical connective "or" can also be modeled as a function-valued function.

(20) **Background on "Or": Syntax**

The connective "or" combines with one sentence S_1 and then combines with another sentence S_2 to create a larger sentence S_3



(21) **Background: Truth Conditions**

The truth-conditions of a sentence containing "or" are as follows:

" [S_2 [or S_1]]" is T *iff* S_2 or S_1

Examples:

a. " [Barack smokes [or Joe dances]]" is T *iff* Barack smokes or Joe dances.

b. " [Seth bowls [or Peggy curls]]" is T *iff* Seth bowls or Peggy curls.

From these background facts/assumptions, we can start to build a theory of what [[or]] is.

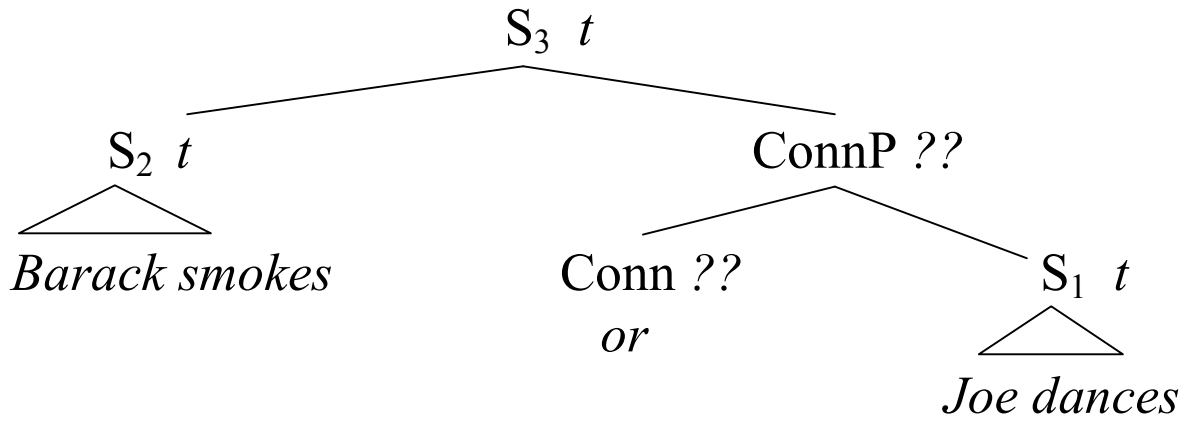
(22) **Reasoning Through the Type Assignments**

Here are the semantic type assignments we already know

[[S_3]] $\in D_t$; [[S_2]] $\in D_t$; [[S_1]]] $\in D_t$

Here are the semantic type assignments we don't yet know

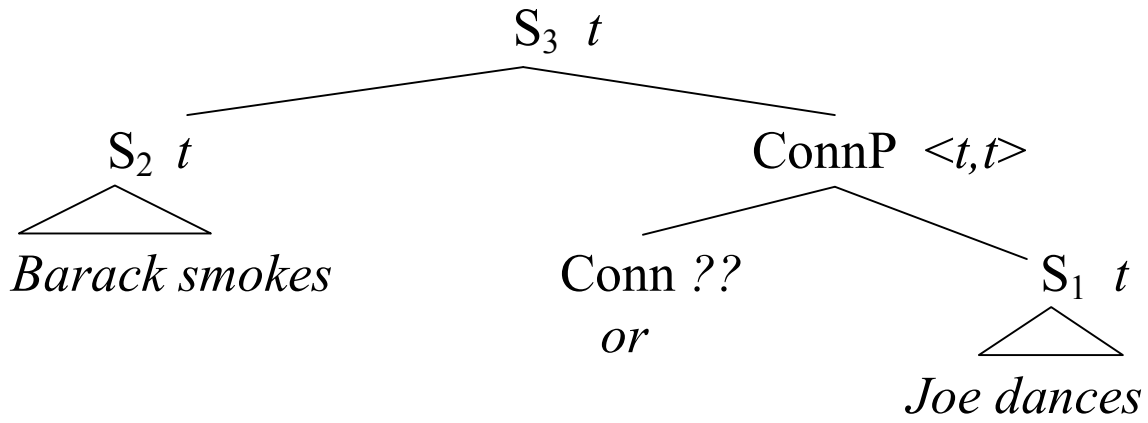
[[ConnP]] $\in$?? ; [[Conn]] $\in$??



(23) **Reasoning Out the Semantic Type of the ConnP**

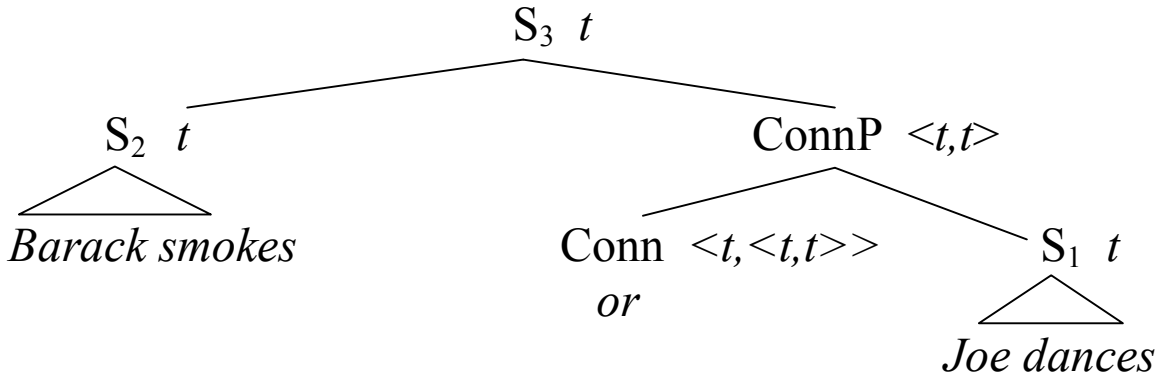
- The entire sentence S_3 is type *t*
- The sentence S_2 is of type *t*
- Now… the extension of ConnP must 'combine' with the extension of S_2 to give us the extension of the whole sentence, S_3
- The only rule we have for such 'combination' is Function Application (FA)…
- So, [[ConnP]] has to be a function that takes a T-value as argument and gives back a T-value…

- **THEREFORE, the semantic type of the ConnP must be *<t,t>* (like the logical connective "it is not the case that")**



(24) **Reasoning Out the Type of the Conn "or"**

- The entire ConnP "or Joe dances" is of type *<t,t>*
- The sentence S_1 is of type *t*
- Now… the extension of the Conn "or" must 'combine' with the extension of S_1 to give us the extension of the whole ConnP…
- The only rule we have for such 'combination' is FA…
- So, [[Conn]] has to be a function which takes as T-value as argument and gives back [[ConnP]] (a function of type <t,t>)
- **THEREFORE, the semantic type of the connective "or" must be <t <t,t>>**



Conclusion: Again, we know abstractly what *kind* of extension "or" should have:
a function from truth-values to a function from truth-values to truth-values

(25) **New Goal**

Develop a lexical entry for "or" that does the following:

a. Stipulates its extension as a function of type *<t, <t,t>>*

b. Allows our system to derive the following truth-conditional statement…

"Barack smokes or Joe dances" is T *iff* Barack smokes or Joe dances.

(26) **An Important Trick We Will Use**

a. Old, Intuitive Observation
The following two statements 'say the same thing':

(i) "S" is true *"Dave smokes" is True*
(ii) S *Dave smokes*

b. New Way of Stating This Equivalentce

[[S]] = T and S can be replaced in our metalanguage.

(27) **Reasoning Out the Extension of the ConnP "or Joe dances", Part 1**

a. Let's start off by considering the T-conditions of some other sentences containing the ConnP "or Joe dances"

(i) "Barack smokes or Joe dances" is T iff Barack smokes or Joe dances
(ii) "Seth drinks or Joe dances" is T iff Seth drinks or Joe dances

b. Now, let's use our 'cool trick' in (26b) to get the following equivalent statements:

(i) "Barack smokes or Joe dances" is T iff [[Barack smokes]] = T or Joe dances

(ii) "Seth drinks or Joe dances" is T iff [[Seth drinks]] = T or Joe dances

c. KEY GENERALIZATION

[[S [or Joe dances]]] = T iff [[S]] = T or Joe dances

(28) **Reasoning Out the Extension of the ConnP "or Joe dances", Part 2**

a. KEY DEDUCTION

- We've already concluded that [[or Joe dances]] is of type <t,t>
- Consequently, our rule of FA entails that

  [[S [or Joe dances]]] = [[or Joe dances]]([[S]])

- But, given our 'Key Generalization' in (27c) above, it follows that:

  [[or Joe dances]]([[S]]) = T iff [[S]] = T or Joe dances

- **KEY CONCLUSION**

  [[or Joe dances]] is a function which takes as argument a T-value x, and yields T iff *either* x = T *or* Joe smokes

b. Deduced Semantics for the ConnP

[[or Joe dances]] = k: $D_t \rightarrow D_t$
for every $x \in D_t$, k(x) = T *iff* x = T or Joe dances

So, we've deduced what the extension of the ConnP "or Joe dances" is…
… but this equation can't just be stipulated in our lexicon.

***Instead, we need our system to derive* the equation in (28b) from the meaning (extension) of "or" and the meaning (extension) of "Joe dances"…**

(29) **Reasoning Out the Extension of Extension of the Connective "Or", Part 1**

a. Let's start off by considering the T-conditions of some other sentences that contain the connective "or"

(i) "Barack smokes **or Seth drinks**" is T *iff* Barack smokes **or Seth drinks**.
(ii) "Joe dances **or Seth drinks**" is T *iff* Joe dances **or Seth drinks**.

(iii) "Barack smokes **or Peggy curls**" is T *iff* Barack smokes **or Peggy curls**.
(iv) "Joe dances **or Peggy curls**" is T *iff* Joe dances **or Peggy curls**.

(30) **Reasoning Out the Extension of Extension of the Connective "Or", Part 2**

a. GENERALIZATIONS

(i) Given exactly the reasoning laid out in (27)-(28) earlier, we can deduce from the facts in (29ai) and (29aii) that:

[[or Seth drinks]] = l: $D_t \rightarrow D_t$
for every $x \in D_t$, l(x) = T *iff* x = T or Seth drinks

(ii) Given exactly the reasoning laid out in (27)-(28) earlier, we can deduce from the facts in (29aiii) and (29aiv) that:

[[or Peggy curls]] = m: $D_t \rightarrow D_t$
for every $x \in D_t$, m(x) = T *iff* x = T or Peggy curls

b. KEY GENERALIZATION

(i) [[or **S**]] = n: $D_t \rightarrow D_t$
for every $x \in D_t$, n(x) = T *iff* x = T or **S**

...or equivalently (given (26))...

(ii) [[or **S**]] = n: $D_t \rightarrow D_t$
for every $x \in D_t$, n(x) = T *iff* x = T or **[[S]] = T**

c. KEY DEDUCTION

- We've already concluded that [[or]] is of type <t,<t,t>>
- Consequently, FA entails that [[or S]] = [[or]]([[S]])
- Thus, given our 'Key Generalization' in (30b), it follows that:

[[or]]([[S]]) = p: $D_t \rightarrow D_t$
for every $x \in D_t$, p(x) = T *iff* x = T or **[[S]] = T**

- **KEY CONCLUSION**
[[or]] is a function which takes as argument a truth-value y, and yields the following function:
p: $D_t \rightarrow D_t$
for every $x \in D_t$, p(x) = T *iff* x =T or y = T

(31) **Deduced Semantics for the Logical Connective "Or"**

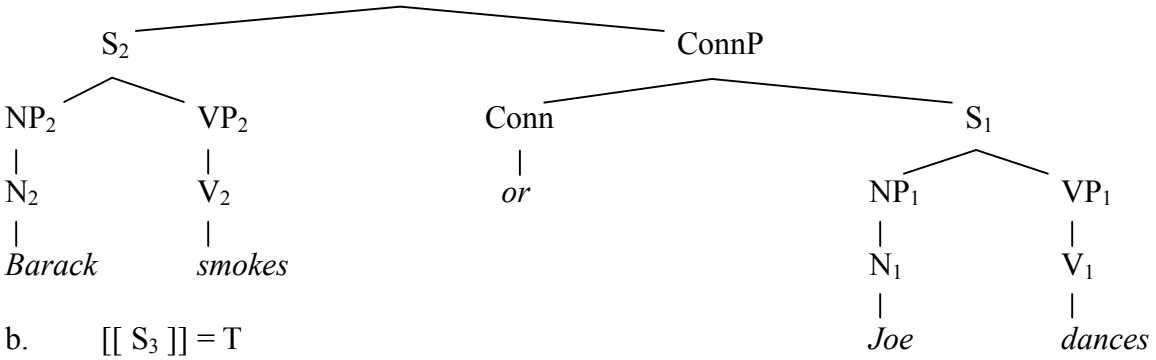
[[or]] = q: $D_t \rightarrow D_{<t,t>}$
for every $x \in D_t$, $q(x) = p_x : D_t \rightarrow D_t$
for every $y \in D_t$, $p_x(y) = T$ iff y=T or x=T

"the function q from T-values to functions from T-values to T-values, which
for every T-value x, maps x to the function p_x from T-values to T-values, which
for every T-value y maps y to T iff x=T or y=T"

With this lexical entry for "or", we can now derive the T-conditions of the complex sentence "*Barack smokes or Joe dances"*.

(32) **Derivation of the Truth-Conditions of "Barack smokes or Joe dances"**

a. " S_3 " is T *iff* (by notation)



b. [[S_3]] = T

c. **Subproof**

(i) [[Conn]] = (by NN)

(ii) [[or]] = (by TN, (31))

(iii) q

d. **Subproof**

(i) [[NP_1]] = (by NN)

(ii) [[N_1]] = (by NN)

(iii) [[Joe]] = (by TN)

(iv) Joe

e. **Subproof**

(i) [[VP_1]] = (by NN)

(ii) [[V_1]] = (by NN)

(iii) [[dances]] = (by TN, (3))

(iv) h

f. **Subproof**

(i) [[S_1]] = (by FA, d,e)

(ii) [[VP_1]]([[NP_1]]) = (by d, e)

(iii) h(Joe)

g. **Subproof**

(i) [[ConnP]] = (by FA, c, f)

(ii) [[Conn]]([[S_1]]) = (by c)

(iii) q([[S_1]]) = (by def. of 'q')

(iv) $p : D_t \rightarrow D_t$
for every $y \in D_t$, $p(y) = T$ iff $y=T$ or [[S_1]] = T (by f)

(v) $p : D_t \rightarrow D_t$
for every $y \in D_t$, $p(y) = T$ iff $y=T$ or h(Joe) = T (by def. of 'h')

(vi) $p : D_t \rightarrow D_t$
for every $y \in D_t$, $p(y) = T$ iff $y=T$ or Joe dances

h. **Subproof**

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

(ii) [[N_2]] = (by NN)

(iii) [[Barack]] = (by TN)

(iv) Barack

i. **Subproof**

(i) [[VP_2]] = (by NN)

(ii) [[V_2]] = (by NN)

(iii) [[smokes]] = (by TN, (3))

(iv) f

j **Subproof**

(i) [[S_2]] = (by FA, h, i)

(ii) [[VP_2]]([[NP_2]]) = (by h, i)

(iii) f(Barack)

k. [[S_3]] = T *iff* (by FA, g, j)

l. [[ConnP]]([[S_2]]) = T *iff* (by g)

m. p([[S_2]]) = T *iff* (by def. of 'p')

n. [[S_2]] = T or Joe dances *iff* (by j)

o. f(Barack) = T or Joe dances *iff* (by def. of 'f')

p. Barack smokes or Joe dances

The Upshot:

- In order to capture the semantics of sentential connectives like "or", we must expand our system of types for natural language, so that it also includes functions of type *<t, <t, t>>*
- That is, we can model the extensions of logical sentential connectives as functions of type *<t, <t, t>>*

5. The System of Semantic Types, A Second Pass

(33) **Our Semantic Types So Far**

a. The domain of entities: D_e

b. The domain of truth-values: D_t

c. The domain of functions from entities to truth-values: $D_{<et>}$

d. The domain of functions from entities to functions from entities to truth-values: $D_{<e,<et>>}$

(34) **Inductive Definition of *All* the Logically Possible Semantic Types**

a. *e* is a semantic type

b. *t* is a semantic type

c. If X is a semantic type and Y is a semantic type, then <X,Y> is a semantic type

(35) **Inductive Definition of *All* the Logically Possible Denotations**

a. D_e is the set of all entities

b. D_t is the set of truth-values

c. For any two domains D_X and D_Y , $D_{<X,Y>}$ is a possible domain, and is the set of all functions from D_X into D_Y

MAJOR ISSUE:

- The definition in (34) and (35) introduce a huge (infinite) set of possible types and denotation domains…
- …do expressions of natural language make use of *all* these types and domains? (obviously not…)

(36) **Open Question**

What are the semantic types/domains employed by natural languages? What is the proper characterization of them?