Semantics Bootcamp (I): Basics of semantics

Yimei Xiang, Harvard University

yxiang@fas.harvard.edu

Roadmap

- Compositional semantics
  - The principle of compositionality
  - Type theory
  - λ-calculus
  - Composition rules
  - Determiners, generalized quantifier
  - Quantifier raising, phrasal movement
- Intensional semantics
- Canonical approaches to question semantics
  - Categorial approach
  - Hamblin-Karttunen Semantics
  - Partition Semantics
  - Comparing the approaches

1. Compositional Semantics and λ-calculas

(In this section, we ignore the extension-intension contrast; in other words, we ignore the evaluation world.)

1.1. The principle of compositionality

- In generative grammar, a central principle of formal semantics is that the relation between syntax and semantics is *compositional*.

(1) **The principle of compositionality (Fregean Principle):** The meaning of a complex expression is determined by the meanings of its parts and the way they are syntactically combined.

The meaning of (2) is the result of applying the unsaturated part of the sentence (a function) to the saturated part (an argument).

(2) Kitty meows.
  a. $[\![\text{Kitty}]\!] = \textsf{Kitty}$
  b. $[\![\text{meows}]\!] = \{x : x \text{ meows}\}$
  c. $[\![\text{meows}]\!] = f : D_e \rightarrow \{1, 0\}$ such that for every x: $f(x) = 1$ iff x meows.

S
(saturated)
Kitty (Saturated) meows (Unsaturated)

- If we think of predicates as denoting sets of entities, then the composition of ”Kitty” and “meows” proceeds via *set membership*:

  $[\![\text{Kitty meows}]\!] = 1$ iff $[\![\text{Kitty}]\!] \in [\![\text{meows}]\!]$, iff $\textsf{Kitty} \in \{x : x \text{ meows}\}$

- If we think of predicates as denoting functions (from sets of entities to truth values), then the composition of “Kitty” and “meows” proceeds via *functional application*:

  $[\![\text{Kitty meows}]\!] = [\![\text{meows}]\!]([\![\text{Kitty}]\!]) = 1$ iff Kitty meows.

1.2. Semantic Types

- The basic types correspond to the objects that Frege takes to be saturated.
  - e for individuals, in D_e
  - t for truth values, in D_t (viz., $\{1, 0\}$)

From these basic types, we can recursively define complex types:

- $\langle e,t\rangle$ for intransitive verbs, predicative adjectives, and common nouns
- $\langle e,\langle e,t\rangle\rangle$ for transitive verbs

A recursive definition of semantic types:

(3) a. **Basic types**: e (individuals/entities) and t (truth values).
b. **Functional types**: If α and β are types, then $\langle\alpha,\beta\rangle$ is a type. A function of type $\langle\alpha,\beta\rangle$ is one whose arguments/inputs are of type α and whose values/outputs are of type β.

- Syntactic categories and their semantic types (an inclusive list)

Syntactic category	Label	**English expressions**	**Semantic type** (extensionalized)
Sentence	S		t
Proper name	ProperN	*John*	e
e-type/referential NP	DP	*the king*	e
Common noun	CN	*cat*	$\langle e,t\rangle$
IV, VP	$\mathrm{V_{itr}}$, VP	*run, love Kitty*	$\langle e,t\rangle$
TV	$\mathrm{V_{tr}}$	*love, buy*	$\langle e,et\rangle$
Predicative ADJ	Adj	*happy, gray*	$\langle e,t\rangle$
Predicate modifier	Adj, Adv	*skillful, quickly*	$\langle et,et\rangle$
Sentential modifier		*perhaps, not that*	$\langle t,t\rangle$
Generalized quantifier	DP	*someone, every cat*	$\langle et,t\rangle$
Quantificational determiner	D	*some, every, no, a*	$\langle et,\langle et,t\rangle\rangle$
Definite determiner	D	*the*	$\langle et,e\rangle$
Relative clause	REL	*who invited Andy*	$\langle e,t\rangle$
		a	$\langle et,et\rangle$, or $\langle et,\langle et,t\rangle\rangle$
		is	$\langle et,et\rangle$, or $\langle e,et\rangle$
		that	$\langle t,t\rangle$, or $\langle et,e\rangle$

1.3. Lambda calculas

1.3.1. Functions

- A function f from A to B is a relation such that (i) f maps every element in A to some element in B, and (ii) each element in A is paired with just one element in B.

(4) a. $[\![\text{the mother of}]\!] = f : D_e \to D_e$ such that for all $x \in D_e$, $f(x)$ is the mother of x.
b. $[\![\text{meow}]\!] = f : D_e \to \{1,0\}$ such that for all $x \in D_e$, $f(x) = 1$ iff x meows.

If the domain or range of a function is of a complex type, the notations could be quite complex:

(5) $[\![\text{hit}]\!]^w = f : D_e \to D_{\langle e,t\rangle}$ such that
for all $x \in D_e$, $f(x) = g : D_e \to \{1,0\}$ such that for all $y \in D_e$, $g(y) = 1$ iff y hits x.

1.3.2. λ-calculus

- It is more handy and common to write functions in **lambda (λ)-notations**.

(6) **Schema of lambda terms:**
$\lambda v[\beta.\alpha]$ read as "the function which maps every v such that β to α"
a. v is the argument variable
b. β is the domain condition (the domain over which the function is defined)

c. α is the value description (a specification of the value/output of the function)

(7) **Lambda reduction/conversion**
$(\lambda v.\alpha)(a) = \alpha'$ where α' is like α but with every *free* occurrence of v replaced by a.
(Note: Occurrences of v that are free in α are bound λv in $\lambda v.\alpha$)

(8) Examples in math
$\lambda x[x \in N.\ x+1]$ read as "the function that maps every x such that x is in N to $x+1$."

a. $(\lambda x[x \in N.\ x+1])(2) = 2+1$
$= 3$

b. $(\lambda x[x \in N.\ x+1])(a)$ is undefined

(9) **Semantic types of lambda terms**
If v is of type σ and α is of type τ, then $\lambda v.\alpha$ is of type $\langle\sigma,\tau\rangle$.

Exercise: Specify its semantic types of the following λ-abstracts.

(10) a. $\lambda f_{\langle e,t\rangle}\lambda x_e[f(x) \wedge \mathsf{gray}(x)]$

b. $\lambda f_{\langle e,t\rangle}\lambda g_{\langle e,t\rangle}.\exists x[f(x) \wedge g(x)]$

1.3.3. Defining semantics of natural languages expressions using lambda-notations

- Predicates

(11) Verbal predicates:

a. $[\![\text{meow}]\!] = \lambda x_e.\ \mathsf{meow}(x)$

b. $[\![\text{hit}]\!] = \lambda y_e\lambda x_e.\ \mathsf{hit}(x,y)$

(12) Non-verbal predicates:

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

b. $[\![\text{larger than}]\!]=$

Dicussion: Why is the following notation incorrect?

(13) $\times$ $[\![\text{hit}]\!] = \lambda x_e\lambda y_e.\ \mathsf{hit}(x,y)$

- Other functions

(14) Sentential connectives

a. $[\![\text{not}]\!] = \lambda p_t.\neg p$

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

(15) Functions over functions

a. $[\![\text{fast}]\!] = \lambda P_{\langle e,t\rangle}\mathsf{fast}(P)$

b. $[\![\text{fast}]\!] = \lambda P_{\langle e,t\rangle}\mathsf{fast}(\lambda x.P(x))$

Exercise: Simplify the following formulas:

(16) a. $(\lambda f_{\langle e,t\rangle}\lambda x_e[f(x) \wedge \mathsf{gray}(x)])(\lambda y_e.\mathsf{cat}(y))$

b. $(\lambda P_{\langle e,t\rangle}.P(k))(\lambda y_e.\mathsf{cat}(y))$

1.4. Syntactic rules and composition rules

- **Syntactic rules**

(17) **Phrase structure rules** (an inclusive list)

S → DP VP	DP → ProperN
VP → V_{itr}	DP → (D) NP
VP → V_{tr} DP	NP → CN

(18) **Vocabulary**
V_{itr} → ran, meows
V_{tr} → likes, hit
ProperN → John, Mary
D → a, the, some, every
CN → student, cat

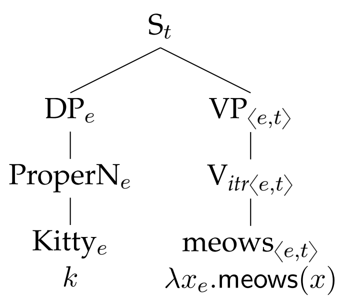
- **Basic composition rules**

(19) **Terminal Nodes (TN)**
If α is a terminal node, $[\![\alpha]\!]$ is specified in the lexicon.

Non-Branching Nodes (NN)
If α is non-branching node, and β is its daughter node, then $[\![\alpha]\!] = [\![\beta]\!]$.

Functional Application (FA)
If $\{\beta, \gamma\}$ is the set of α's daughters, $[\![\beta]\!] \in D_{\langle\sigma,\tau\rangle}$, and $[\![\gamma]\!] \in D_\sigma$, then $[\![\alpha]\!] = [\![\beta]\!]([\![\gamma]\!])$

Example:



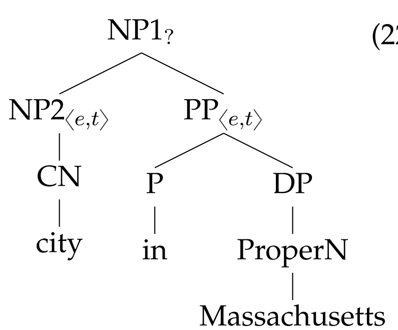
(20) Kitty meows.

a. $[\![DP]\!] = [\![ProperN]\!] = [\![Kitty]\!] = k$ — By TN, NN
b. $[\![VP]\!] = [\![V_{itr}]\!] = [\![meows]\!] = \lambda x_e.\mathsf{meows}(x)$ — By TN, NN
c. $[\![S]\!] = [\![VP]\!]([\![NP]\!])$ — By FA
$= [\lambda x_e.\mathsf{meows}_w(x)](k)$
$= \mathsf{meows}(k)$

- **Other composition rules:**

(21) **Predicate Modification (PM)**
If α is a branching node, $\{\beta, \gamma\}$ is the set of α's daughters, and $[\![\beta]\!]$ and $[\![\gamma]\!]$ are both in $D_{\langle\sigma,t\rangle}$, then $[\![\alpha]\!] = \lambda x_\sigma[[\![\beta]\!](x) = [\![\gamma]\!](x) = 1]$ (Or equivalently: $[\![\alpha]\!] = \lambda x_\sigma[[\![\beta]\!](x) \wedge [\![\gamma]\!](x)]$)

Example:

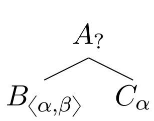


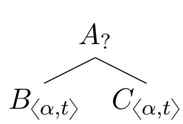
(22) Cambridge is a <u>city in Massachusetts</u>.

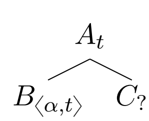
a. $[\![NP2]\!] = ... = \lambda x_e \mathsf{city}(x)$
b. $[\![PP]\!] = ... = \lambda x_e[\mathsf{in}(x, \mathsf{m})]$
c. $[\![NP1]\!] = \lambda x_e[[\![NP2]\!](x) \wedge [\![PP]\!](x)]$ — By PM
$= \lambda x_e[(\lambda y_e.\mathsf{city}(y))(x) \wedge (\lambda z_e.\mathsf{in}(z, \mathsf{m}))(x)]$
$= \lambda x_e[\mathsf{city}(x) \wedge \mathsf{in}(x, \mathsf{m})]$

- **Type-mismatch:** In case that none of the composition rules can proceed (i.e., two sister nodes neither hold a function-argument relation, nor be of the same type $\langle\sigma, t\rangle$), we say that the composition suffers *type-mismatch*.

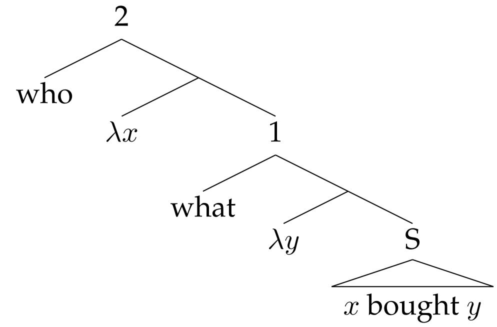
Exercise: Determine types of nodes in a tree:







Discussion: Traditional categorial approaches of questions treat *wh*-words as λ-operators. Let's try to compose the following structure while assuming the lexical entries in (23a-b). What composition rules can we use for composing Node 1? What about for Node 2?



(23) Who bought what?

a. $[\![\text{who}]\!] = \lambda P_{\langle e,t\rangle}\lambda x_e[\mathsf{human}(x) \wedge P(x)]$

b. $[\![\text{what}]\!] = \lambda P_{\langle e,t\rangle}\lambda x_e[\mathsf{thing}(x) \wedge P(x)]$

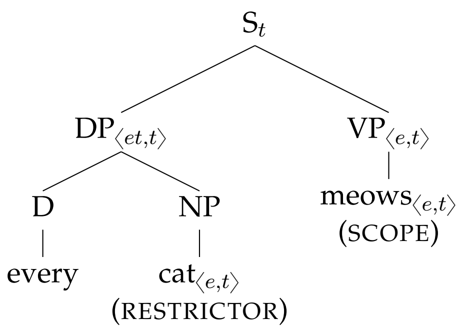
1.5. Generalized quantifiers, quantifier raising and phrasal movement

1.5.1. Generalized quantifiers and quantificational determiners

- Quantificational DPs (e.g. *everything*, *something*, *every cat*, *some cat*) are not individuals (cf. proper names like *John*), nor individual sets (cf. common nouns like *cat*).

  We treat quantificational DPs as second-order functions of type $\langle et, t\rangle$, called **generalized quantifiers (GQs)**. In (24), *moews* is an argument of *every cat*. The **quantificational determiner** *every* combines with a common noun of type $\langle e, t\rangle$ to return a generalized quantifier of type $\langle et, t\rangle$. Therefore, its type is quite complex: $\langle et, \langle et, t\rangle\rangle$.

(24)



a. $[\![\text{every}]\!] = \lambda Q_{\langle e,t\rangle}\lambda P_{\langle e,t\rangle}.\forall x[Q(x) \rightarrow P(x)]$

b. $[\![\text{every cat}]\!] = \lambda P_{\langle e,t\rangle}.\forall x[\mathsf{cat}(x) \rightarrow P(x)]$

c. $[\![\text{every cat meows}]\!]$
$= [\![\text{every cat}]\!]([\![\text{meows}]\!])$
$= (\lambda P_{\langle e,t\rangle}.\forall x[\mathsf{cat}(x) \rightarrow P(x)])(\lambda y_e.\mathsf{meows}(y))$
$= \forall x[\mathsf{cat}(x) \rightarrow \mathsf{meows}(x)])$

- Other quantificational determiners:

(25) a. $[\![\text{some}]\!] = \lambda Q_{\langle e,t\rangle}\lambda P_{\langle e,t\rangle}.\exists x[Q(x) \wedge P(x)]$

b. $[\![\text{no}]\!] = \lambda Q_{\langle e,t\rangle}\lambda P_{\langle e,t\rangle}.\neg\exists x[Q(x) \wedge P(x)]$

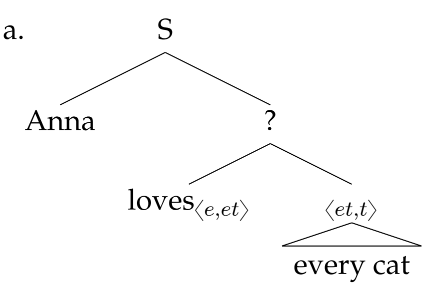
1.5.2. Quantifier raising and phrasal movement

- Problem in (26a): A **type-mismatch** arises when a GQ appears at a non-subject position.

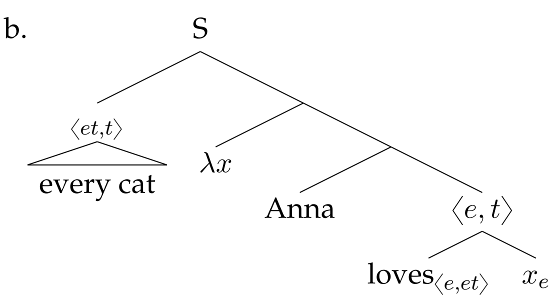
  Solution in (26b): A covert **movement** of the generalized quantifier, called **Quantifier Raising** (QR).

(26) Anna loves every cat.

a.



b.



  - At LF, the generalized quantifier *every cat* is moved to the left edge of the sentence, leaving a trace.
  - We interpret this trace as a variable of a matching type (i.e., x_e), and then abstract over this variable by inserting λx_e immediately below *every cat*. This abstraction operation is called **Predicate Abstraction**.

- In generative grammar, phrasal movement (overt or covert) is uniformly formalized as follows:

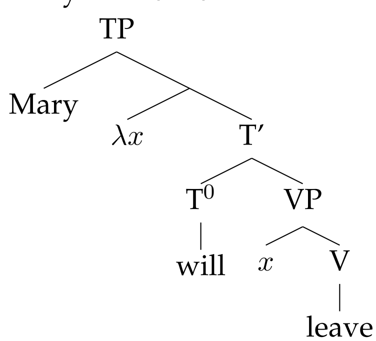
(27) **Movement of a phrase α from position A to position B:**
  a. α is moved to position B;
  b. α in A is replaced with a trace, interpreted as a variable;
  c. we abstract over this trace variable by inserting a matching lambda node immediately below B, forming a λ-abstract.

The sister node of the λ-abstract is the moved phrase. The variable bound by the λ-operator is the trace.

- Examples: what elements are moved in the following structures?

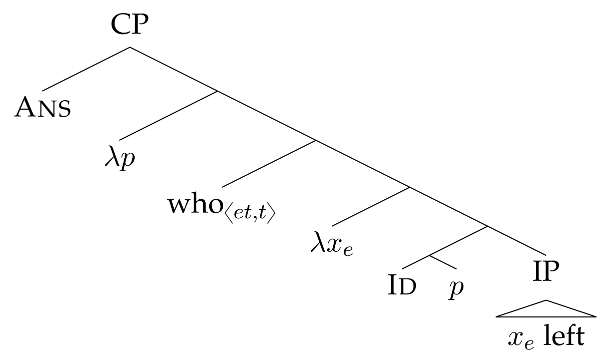
– Subject movement:

(28) Mary will leave.



– *Wh*-movement (Heim 1995)

(29) Who left?



2. Intensional Semantics

2.1. Extension

- The **extension** of an expression is dependent on the evaluation world. We add an *evaluation world parameter* $[\![\bullet]\!]^w$ to the notations of extensions:

(30) General notation: $[\![X]\!]^w$ ('the extension of X in w')

  - $[\![\bullet]\!]$ is called **Interpretation function**; it maps syntactic expressions to their denotation/meaning.
  - The w in $[\![\bullet]\!]$ is called the evaluation world.

Examples:

(31) a. $[\![\text{Mary lives in Cambridge}]\!] = 1$ iff Mary lives in Cambridge in w.
  b. $[\![\text{old}]\!]^w = \{x : x \text{ is old in } w\}$ (As a set)
  $[\![\text{old}]\!]^w = f : D_e \to \{0, 1\}$ s.t. for all $x \in D$, $f(x) = 1$ iff x is old in w. (As a characteristic function)
  $[\![\text{old}]\!]^w = \lambda x_e.\mathsf{old}_w(x)$. (Using λ-notation)

2.2. Why is extensional semantics insufficient?

- So far, we've been using extensional semantics: the meaning of a complex expression is composed from the extensions of the components of a complex expression. For example, *met* is a relation between entities:

(32) $[\![\text{Andy met Betty}]\!]^w = [\![\text{met}]\!]^w([\![\text{Andy}]\!]^w, [\![\text{Betty}]\!]^w) = [\![\text{met}]\!]^w(a, b)$

- Nevertheless, the following example shows that extensional semantics is insufficient:

In the actual world w, Gennaro smokes and likes Belgian chocolate. Thus:

(33) $[\![\text{Gennaro smokes}]\!]^w = [\![\text{Gennaro likes Belgian chocolate}]\!]^w = 1$

If *believe* expresses a relation between the extension of the belief holder and the the extension of the embedded sentence, we have:

(34) $[\![\text{Kate believes S}]\!]^w = [\![\text{believe}]\!]^w([\![\text{Kate}]\!]^w, [\![\text{S}]\!]^w)$

Due to the equation in (33), we expect the following equation to hold in the actual world:

(35) $[\![\text{Kate believes Gennaro smokes}]\!]^w = [\![\text{Kate believes Gennaro likes Belgian chocolate}]\!]^w$
   a. $[\![\text{believe}]\!]^w([\![\text{Kate}]\!]^w, [\![\text{Gennaro smokes}]\!]^w) = [\![\text{believe}]\!]^w(k, 1)$
   b. $[\![\text{believe}]\!]^w([\![\text{Kate}]\!]^w, [\![\text{Gennaro likes Belgian chocolate}]\!]^w) = [\![\text{believe}]\!]^w(k, 1)$

However, equation (35) doesn't hold. In fact, Kate knows Gennaro smokes, but she doesn't know that he likes Belgian chocolate.

The fact that equation (33) doesn't ensure equation (35) shows that the meaning of *believe* cannot be defined purely based on the extension of its embedded clause. In other words, *believe* is not type of type $\langle t, \langle e, t\rangle\rangle$. It does not express a relation between an entity (Kate) and a truth value (the extension of "Gennaro smokes").

2.3. Defining intension

- The **intension** of an expression X is a function which (i) takes a possible world as an argument, and (ii) returns the extension of X in that world.

  (36) General notation: $\lambda w_s.[\![X]\!]^w$ ('the intension of X')

  – The intension of a sentence is a function from worlds to truth values, called *proposition*.
  – The intension of a one-place predicate (IV/VP/NP/Pred Adj/..) of type $\langle e, t\rangle$ is a function from worlds to $\langle e, t\rangle$ functions, called *property*.
  – The intension of a definite NP is a function from worlds to entities, called *individual concept*.

Examples: (the descriptions of each example are all equivalent)

(37) The intension of "Gennaro smokes" :
   a. λw_s. Gennaro smokes in w
   b. $\lambda w_s[\text{smokes}_w(g) = 1]$
   c. $\lambda w_s.\text{smokes}_w(g)$

(38) The intension of "composer":
   a. $\lambda w_s \lambda x_e.x$ is a composer in w
   b. $\lambda w_s \lambda x_e[\text{composer}_w(x) = 1]$
   c. $\lambda w_s \lambda x_e.\text{composer}_w(x)$

- A proposition can also be viewed as the set of possible worlds where this proposition is true.

  (39) The intension of "Gennaro smokes": $\{w : [\![\text{Gennaro smokes}]\!]^w = 1\}$

- Hence, we can use set-theoretical operations to represent the following relations and operations:

Relations and operations	Set-theoretical notations
p entails q	$p \subseteq q$
p contradicts q	$p \cap q = \varnothing$
p and q	$p \cap q$
p or q	$p \cup q$
p is possible	$p \neq \varnothing$
p is necessary	$p = W$

- **Discussions:** the following formulas are problematic. Identify and correct the problems.

  (40) a. $\forall q \in C[q \rightarrow q \subseteq p]$ (Every true proposition in C entails p.)
  b. $(p \subseteq q) \rightarrow \neg q$ (If p entails q, then q is false.)

2.4. Intensional-izing the theory of types and compositions

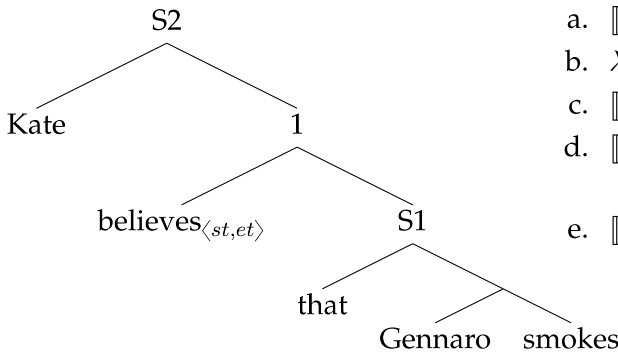
- Intensional-izing the theory of types[1]

(41) a. **Basic types**: e (individuals/entities) and t (truth values).
b. **Functional types**: If σ and τ are types, then $\langle \sigma, \tau \rangle$ is a type.
c. **Intensional types**: If σ is a type, then $\langle s, \sigma \rangle$ is an intensional type.

- Intensional-izing the theory of semantic composition

(42) **Intensional Functional Application**
If $\{\beta, \gamma\}$ is the set of α's daughters, $[\![\beta]\!] \in D_{\langle\langle s,\sigma\rangle,\tau\rangle}$, and $[\![\gamma]\!] \in D_\sigma$, then $[\![\alpha]\!] = [\![\beta]\!](\lambda w.[\![\gamma]\!]^w)$

(43) Kate believes that Gennaro smokes.



a. $[\![\text{S1}]\!]^w = \mathsf{smoke}_w(g)$
b. $\lambda w.[\![\text{S1}]\!]^w = \lambda w.\mathsf{smoke}_w(g)$
c. $[\![\text{believe}]\!]^w = \lambda p_{\langle s,t\rangle}\lambda x_e.\mathsf{believe}_w(x, p)$
d. $[\![1]\!]^w = [\![\text{believe}]\!]^w(\lambda w.[\![\text{S1}]\!]^w)$ By IFA
$= \lambda x_e.\mathsf{believe}_w(x, \lambda w.\mathsf{smoke}_w(m))$
e. $[\![\text{S2}]\!]^w = [\![1]\!]^w([\![\text{Kate}]\!]^w)$ by FA
$= (\lambda x_e.\mathsf{believe}_w(x, \lambda w.\mathsf{smoke}_w(g)))(k)$
$= \mathsf{believe}_w(k, \lambda w.\mathsf{smoke}_w(g))$

- Solving the problem in (35):
  - First, consider the following four worlds:

    a. In $w1$, Gennaro smokes, and he like Belgian chocolate.
    b. In $w2$, Gennaro smokes, but he doesn't like Belgian chocolate.
    c. In $w3$, Gennaro doesn't smoke, but he likes Belgian chocolate.
    d. In $w4$, Gennaro doesn't smoke, and he doesn't like Belgian chocolate.

    Then: The intension of "Gennaro smokes" is $\{w1, w2\}$
    The intension of "Gennaro likes Belgian chocolate" is $\{w1, w3\}$

  - Second, "x believes S" is true in w iff S is true in every world that is compatible with x's belief in w. Assume that in the actual world w, Kate believes that Gennaro smokes, and she has no idea whether he likes Belgian chocolate. Then the worlds that are compatible with Kate's belief in w are $\{w1, w2\}$. Then "Kate believes S" is true in w iff S is true in both $w1$ and $w2$.
  - Third, compute the extension of the two *believe*-sentences:

(44) a. $[\![\text{Kate believes Gennaro smokes}]\!]^w$
$= [\![\text{believe}]\!]^w([\![\text{Kate}]\!]^w, \lambda w.[\![\text{Gennaro smokes}]\!]^w)$
$= 1$
(because for every world in $\{w1, w2\}$, Gennaro smokes is true.)
b. $[\![\text{Kate believes Gennaro likes Belgian chocolate}]\!]^w$
$= [\![\text{believe}]\!]^w([\![\text{Kate}]\!]^w, \lambda w.[\![\text{Gennaro likes Belgian chocolate}]\!]^w)$
$= 0$
(because there is a world in $\{w1, w2\}$ where Gennaro likes Belgian chocolate isn't true.)

[1]Note that we are not actually adding s for possible worlds to our type theory. This is because (as far as we've seen) there are no expressions of natural language that have specific possible worlds as their values.