Introduction to model-theoretic semantics

(Lecture notes for COGS 543)

Umut Özge

October 25, 2017
Draft 1.0

1 Introduction

Model-theoretic semantics is the art of defining correspondences between the expressions of a language and objects that are external to that language. To give a very simple example: assume you and I have a silly code between us, according to which, during the normal course of my speaking whenever I utter the definite article *the* you blink your left eye and whenever I utter the indefinite article *a(n)* you blink your right eye. Of course my utterances have a semantics associated with the language I speak – some idiolect of English; but besides that, there is another correspondence that maps the expressions *the* and *a(n)* to some objects which has no place in the semantics of my English. This second semantics is defined over a microscopic subset of English, namely $\{the, a, an\}$. What would a semantics for this subset look like? Here, one needs to decide on the basis of the particular aim one has in developing a semantics. If we need a semantics to instruct a robot, then it would be logical to construct our correspondence in such a way that expressions are mapped to a set of robot control routines. If we are more on the side of cognitive neuroscience, we would be likely to choose neurological triggering mechanisms as the other side of the correspondence. If the observable behavior would suffice for our purposes, then mapping *the* to "blink left eye", and *a* and *an* to "blink right eye". You may even associate *the* with the probability of observing a left-eye blink within the limits of a particular time frame, so on and so forth. The technical term for what you take to be on the non-language side of the correspondence is "model".

In natural language semantics, the general practice – or where people usually begin – is to take the world as the model. This approach conceives the world as consisting of entities (humans, cats, chairs, hopes, fears...) and properties of and relations among these entities. Again to give a very simple example: the expression "the largest planet in the solar system as of October 15, 2017, 14:14:19 CET" is mapped to Jupiter.

Although model-theoretic semanticists intend to associate language with the world outside, they use a certain formal representation of the world instead of the world itself, when they construct their semantic theories. In the case of physical

objects like Jupiter or the chair I sit on at the moment, the reason why they do so is that the physical objects are impractical (or impossible in the case of Jupiter) to put on paper or computer screen, where you describe your semantics. There are also abstract objects that language refers to, but for which it is not easy, if not impossible, to pick corresponding objects from the outer world.

In natural language semantics, we are interested in natural languages, which are highly ambiguous. Given this and some other concerns that we will come to later, what semanticists do is to first map natural languages to a disambiguated formal language (e.g. first order logic), and then interpret the expressions of this formal language with respect to a given model. We will call such intermediate representation that serve a way point in mapping natural languages to model-theoretic objects "logical forms". In what follows, we will have a deeper look at this process. We will return to the topic of how to map natural language expressions to logical forms later on, where we delve into the notion of grammar that governs that mapping.

2 Characteristic functions and Currying

We will assume that natural language expressions somehow get transformed into a formal disambiguated language. The formal language we will use is a slight variant of first-order logic with equality and lambda abstraction. We will develop it gradually. There are certain notational conventions and conceptual points we diverge from first-order logic. We will first discuss those.

We need to get used to seeing everything as functions. Relations differ from functions in associating an object possibly with more than one objects. In predicate logic, it is straightforward to think of predicates as relations.

(1) $Loves(John, Mary)$

is a relation that holds between a person and any person that s/he is in the loving relation. As for a given individual, it is not guaranteed that there is exactly one individual that s/he loves, the predicate $Loves$ is not a function.

It is also mathematically sound to think of $Loves$ as the set of ordered pairs of individuals, where the person appearing as the first coordinate loves the one appearing as the second coordinate. Once we start to treat relations as sets, then it is immediate that every relation is equivalent to a function, because every set is equivalent to its characteristic function. Here comes the definition:

Definition 2.1
For every set A, there exists a unique function f_A, called the characteristic function of A, defined as follows:

$$f_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{otherwise} \end{cases}$$

Therefore we will focus not on $Loves$' being a relation from I, the set of individuals, to I, but its being a function $f : I \times I \mapsto \{0, 1\}$. These types of functions will be called Boolean functions; the connection to truth and falsity should be obvious.

Besides Boolean functions, we will also have functions that map individuals to individuals, sets of individuals and various other objects. For instance, assuming that geometric objects and real numbers are also individuals, the function $Diameter$ will map the set of circles to the set of real numbers. For a given circle as input to $Diameter$, there will be one and only one real number associated with the input.

The second concept related to functions that we will need is Currying. Take the following simple function that multiplies two numbers:

```
def multiply(x,y):
    return x*y
```

Whenever you need to use `multiply()`, you have to provide both arguments at the same time. It is impossible to delay the saturation of one of the arguments. Compare the following Curried form with the above:

```
def curried_multiply(x):
    def retval(y):
        return x*y
    return retval
```

Now you can incrementally saturate the function: `curried_multiply(8)` will give you a function that multiplies its argument with 8 and returns the result; to get 72 you need to have `curried_multiply(8)(9)`, rather than `curried_multiply(8,9)`.

The functions in our language will all be Curried as in the above example. Therefore a function like $Loves$ will get its arguments one by one. By convention we will agree that the arguments are fed in the reverse order. For instance, to express what is expressed in (1), we will first feed the second argument to the function, resulting in,

(2) $Loves(Mary)$

This form stands for a function that returns 1 for individuals who love Mary and 0 otherwise. The full form is,

(3) $Loves(Mary)(John)$

We will adopt two other notational conventions: 1. The constants[1] like $Loves$, $John$, and so on, will be written in lower case and terminated by a prime as in

[1] See below for what "constant" means.

$loves'$ and $john'$ (or an abbreviated form j'); 2. Given a function f and an argument a, we will depict function application as $(f\,a)$, rather than like $f(a)$. Under these conventions (3) becomes,

(4) $((loves'\,m')\,j')$

The next topic we need to cover deserves a section on its own, it is one of the most important concepts in natural language semantics.

3 Semantic types

In programming, the error you get when you attempt to take the square root of a string is a type error. The square root function is a function from floats to floats, it neither accepts nor returns any other type of object. Our functions will be similarly typed, so that they work with only objects of types designated beforehand.

Thanks to the fact that our functions are Curried, we will be able to elegantly characterize the set of possible types our functions can get on the basis of a small set of basic types.

For now we will have only two basic types. It is remarkable how far one can go with just two basic types in semantics. One of our basic types is the type of expressions that have a truth value; a sentence like *Mary smiled*, for instance. We will designate this type with t. The second basic type is the type of individuals (or entities), and it is designated by e. What about functions? The type of a function is designated by having its domain's (input) type and range's (output) type separated by a comma and enclosed in angle brackets – like in html tags. For instance, it looks reasonable to take $smiled'$ to be a function that returns 1 or 0 according to whether or not it takes an individual who had a smile on his/her face some time before the expression is uttered. Therefore it has the type $\langle e, t\rangle$, a function from individuals to truth values. What about $loves'$? Given that it takes an individual and returns a function, its result type should be more complex than its input type. The type of $loves'$ after it is fed by an individual is a function that takes an individual as an input and gives 1 if that individual loves whatever was the first argument and 0 otherwise. Therefore $loves'$ has the type $\langle e, \langle e, t\rangle\rangle$: a function from individuals to functions from individuals to truth values.

Here is a definition that provides all the possible types that can be built on the basis of e and t:

Definition 3.1 (Semantic types)

i. e and t are types;

ii. if α and β are types, then so is $\langle \alpha, \beta\rangle$;

iii. nothing else is a type.

In defining our formal language for semantic representations, every item in our vocabulary will belong to one of the types defined in Definition 3.1.

4 A language for logical forms

The vocabulary of our language – everything except parentheses – comes from the three sets below; type of each item is subscripted to its name, except quantifiers which do not have types.

Logical constants come in two separate sets, connectives and quantifiers:

$$K = \{\wedge_{\langle t,\langle t,t\rangle\rangle}, \vee_{\langle t,\langle t,t\rangle\rangle}, \rightarrow_{\langle t,\langle t,t\rangle\rangle}, \neg_{\langle t,t\rangle}\}$$
$$Q = \{\forall, \exists\}$$

Non-logical constants:

$$C = \{loves'_{\langle e,\langle e,t\rangle\rangle}, diameter'_{\langle e,e\rangle}, slowly'_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}, blue'_{\langle e,t\rangle}, j'_e, \ldots\}$$

Variables:

$$V = \{p_{\langle e,t\rangle}, q_{\langle e,t\rangle}, r_{\langle e,t\rangle}, \ldots, x_e, y_e, z_e, \ldots\}$$

Think of logical constants as function words in natural language; non-logical constants as content-words; variables as pronouns. Variables are distinguished from non-logical constants by not having a prime. They are also always single character. When the type of an expression is not at issue or obvious, we do not write it.

We are ready to define the well-formed expressions of our language.

Let L be the set of well-formed expressions of the language of logical forms.

Definition 4.1 (Syntax of L)

i. $C \cup K \cup V \subseteq L$

ii. If $\pi_{\langle\alpha,\beta\rangle} \in L$ and $\sigma_\alpha \in L$, then $(\pi\sigma)_\beta \in L$

iii. If $\kappa \in \{\forall, \exists\}$, $\chi \in V$ and $\tau_t \in L$, then $(\kappa\chi\tau)_t \in L$

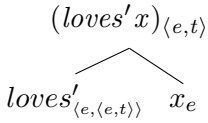
iv. Nothing else is in L.

□

We have an elegant, but cryptic, definition of the syntax of L. Now we clarify.

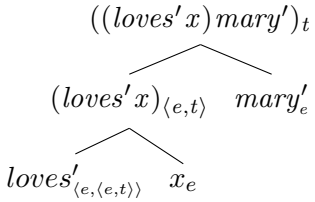
Clause (i) says that all items in our vocabulary except the quantifiers are well-formed expressions. Clause (ii) builds up complex expressions through function application. A simple example:

(5)



In (5), the leaves of the tree, $loves'_{\langle e,\langle e,t\rangle\rangle}$ and x_e are validated by Clause (i), while application of the former to the latter to form the root of the tree, $(loves'\, x)_{\langle e,t\rangle}$, is validated by Clause (ii). Now we take a further step and apply the function we obtained in (5) to another item from our vocabulary, this time a constant rather than a variable:

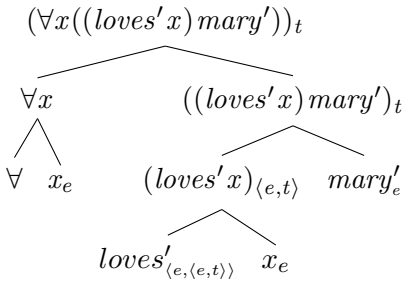
(6)



Our type system says that the expression at the root of (6) is of type t, which means it must stand for something that can be true or false. An approximation to this formal expression from natural language would be *Mary loves it, him or her*. The variable acts like a pronoun as mentioned above. And as in the case of natural language pronouns, we cannot decide on the truth value of the expression unless we know who or what the variable (or pronoun) stands for.

So far we made use of Clauses (i) and (ii) of Definition 4.1. Now let us see Clause (iii) in action.

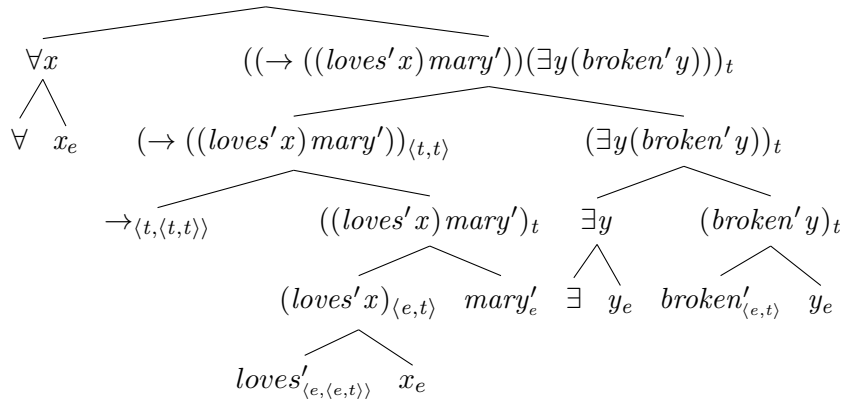
(7)



What we have here is an approximation to the sentence *Mary loves everything*.

In a similar fashion, we can invite our connectives into the business.

(8) $(\forall x((\rightarrow ((loves' x)\, mary'))(\exists y(broken' y))))_t$



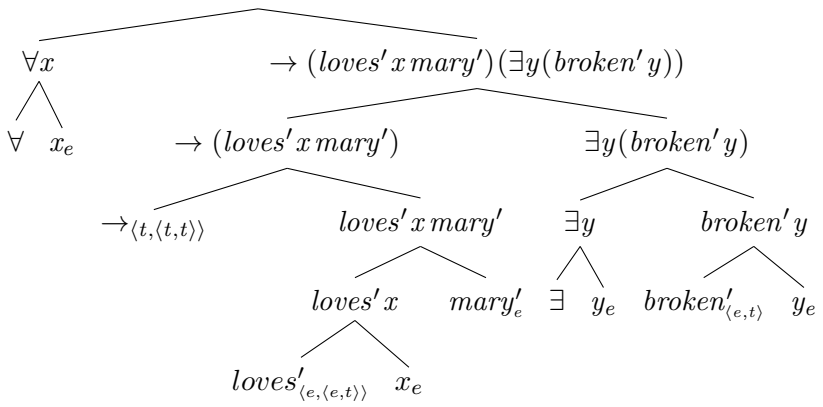
What we just built in (8) is an approximation to *For any individual, if Mary loves him or her, then there is someone who gets broken*.

The formulas in their current forms are quite hard to read. One way to make them more readable is to agree on a convention to eliminate some parentheses. From now on we agree that in function application (i) outermost parentheses are omitted, and (ii) application associates to left. The first is easy to grasp: $((fx)y)$ becomes $(fx)y$. The second needs some time get used to. Assume you are given an expression with some of its parenthesis are removed. Left association means that in restoring parentheses back you start from the two left most expressions, *without looking into parentheses you encounter when counting the left-most two*. Here are a couple of examples:

(9) a. $x\, y\, z \equiv ((xy)z)$
b. $x(yz) \equiv (x(yz))$
c. $x(yz)w \equiv ((x(yz))w)$

Let us now apply these parentheses elimination conventions to (8); observe that the type decorations goes away with the outer parenthesis, you need to mentally keep track of them:

(10) $\forall x(\rightarrow (loves' x\, mary')(\exists y(broken' y)))$



Now it is time to give a semantics to our formal language L. First we define what we understand from a model. A model $\mathcal{M}$ is a tuple $\langle D_e, D_t, I\rangle$, where D_e is the domain of individuals (=a set of e type entities), D_t is the domain of truth values (usually $\{0,1\}$), and I is the interpretation function. We need some more notation to understand the mechanics if I.

Definition 4.2 (Set of functions)
Given two sets X and Y, X^Y denotes the set of all functions defined from Y to X. □

Given a model and the definition (4.2), it is straightforward to denote the domain for any type. For instance, the domain of functions from entities to truth values is $D_t^{D_e}$ or $\{0,1\}^{D_e}$. If you want to domain of functions of type $\langle e, \langle e, t\rangle\rangle$ you will have $(\{0,1\}^{D_e})^{D_e}$, or for $\langle\langle e, t\rangle, t\rangle$ you will have $\{0,1\}^{(\{0,1\}^{D_e})}$, and finally the domain of functions of type $\langle\alpha, \beta\rangle$ is $D_\beta^{D_\alpha}$. Here there is an unfortunate confusion in terminology: when we say "domain", we do not mean the usual sense giving the set of input values for the function; rather we use "domain" in the sense of the set of functions that has the same type as the function under consideration.

Now back to models. For a model $\mathcal{M} = \langle D_e, D - t, I\rangle$ to be a model for a language L, the interpretation function I should be able to return a value for all the constants (logical and non-logical) of L from the domain corresponding to the type of the constant. In other words, I needs to be an effective dictionary for the vocabulary of L.

The interpretation function that comes with the model takes care of constants of our language; how about the variables? The variables – pronouns of L – are handled via *the environment for evaluation*, a function that maps variables of the language to corresponding objects in the model. Like the interpretation function I, the environment, denoted by g maps each variable to an object with the same type. The fundamental difference between I and g is that the former is fixed by the model and does not change during the course of evaluation; the environment on the other hand is dynamic, it gets *extended* during evaluation. Given an environment g, we call the environment $g_{[x\mapsto u]}$ obtained from g by mapping x to u if it is not

already so, an extension of g.

It is time to state the semantics of L with respect to a model $\mathcal{M}$ and an environment g. You can think of evaluation as a higher order function denoted by double pipes: $\|\psi\|_{\mathcal{M},g}$ standing for the semantic evaluation of the expression ψ with respect to $\mathcal{M}$ and g.

Definition 4.3 (Semantics of L)

i. $\|\alpha\|_{\mathcal{M},g} = g(\alpha)$, if $\alpha \in V$

ii. $\|\alpha\|_{\mathcal{M},g} = I(\alpha)$, if $\alpha \in C \cup K$

iii. $\|(\alpha\beta)\|_{\mathcal{M},g} = \|\alpha\|_{\mathcal{M},g}(\|\beta\|_{\mathcal{M},g})$

iv. $\|(\forall\alpha_\tau\beta)\|_{\mathcal{M},g} = 1$ iff for all $d \in D_\tau$, $\|\beta\|_{\mathcal{M},g_{[x\mapsto d]}} = 1$

v. $\|(\exists\alpha_\tau\beta)\|_{\mathcal{M},g} = 1$ iff there is at least one $d \in D_\tau$, $\|\beta\|_{\mathcal{M},g_{[x\mapsto d]}} = 1$

□

An example.

$$\|\forall x(\rightarrow (loves'\, x\, mary')(\exists y(jealous'\, x\, y)))\|_{\mathcal{M},g}$$

Specify the model $\mathcal{M}$ and the environment g:

$$\begin{aligned}
D_e =& \{\textit{mary}, \textit{pedro}, \textit{rosinante}\}\\
D_t =& \{1, 0\}\\
I =& \{(mary', \textit{mary}), (pedro', \textit{pedro}),\\
&(loves', \{(\textit{mary}, \{(\textit{rosinante}, 1), (\textit{pedro}, 1), (\textit{mary}, 0)\}),\\
&\qquad (\textit{pedro}, \{(\textit{rosinante}, 0), (\textit{pedro}, 1), (\textit{mary}, 0)\}),\\
&\qquad (\textit{rosinante}, \{(\textit{rosinante}, 0), (\textit{pedro}, 0), (\textit{mary}, 1)\})\}),\\
&(jealous', \{(\textit{mary}, \{(\textit{rosinante}, 0), (\textit{pedro}, 1), (\textit{mary}, 0)\}),\\
&\qquad (\textit{pedro}, \{(\textit{rosinante}, 1), (\textit{pedro}, 0), (\textit{mary}, 0)\}),\\
&\qquad (\textit{rosinante}, \{(\textit{rosinante}, 0), (\textit{pedro}, 0), (\textit{mary}, 1)\})\})\}\\
g =& \{(x, \textit{rosinante}), (y, \textit{rosinante}\}
\end{aligned}$$

We do not provide the interpretations of logical constants, which are taken to be the standard logical connectives.

At the top-level the formula invokes (iv):

$$\|\forall x(\rightarrow (loves'\, x\, mary')(\exists y(jealous'\, x\, y)))\|_{\mathcal{M},g} = 1 \text{ iff }, \quad (1)$$

$$\text{for all } d \in D_e, \quad \|(\rightarrow (loves'\, x\, mary')(\exists y(jealous'\, x\, y)))\|_{\mathcal{M},g_{[x\mapsto d]}} = 1 \quad (2)$$

We start iterating over D_e, replacing x with a model-theoretic object in D_e and checking wether we obtain 1 as result. First,

$$\|(\rightarrow (loves' x mary')(\exists y(jealous' x y)))\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}} \tag{3}$$

This invokes clause (iii):

$$\|(\rightarrow (loves' x mary')\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}(\|(\exists y(jealous' x y))\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}) \tag{4}$$

To handle this function argument form, let us first compute the function part $\|(\rightarrow (loves' x mary')\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}$, which is itself a function application, invoking clause (iii) again:

$$\| \rightarrow \|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}(\|(loves' x mary')\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}) \tag{5}$$

Now evaluate $\|(loves' x mary')\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}$, invoking clause (iii):

$$\|(loves' x)\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}(\|mary'\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}) \tag{6}$$

Evaluate $\|(loves' x)\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}$, once again invoking clause (iii):

$$\|loves'\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}(\|x\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}) \tag{7}$$

Evaluate $\|x\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}}$, invoking clause (i):

$$\|x\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}} = g_{[x\mapsto \textit{mary}]}(x) = \textit{mary} \tag{8}$$

We also have the following by clause (ii),

$$\|loves'\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}} = I(loves') \tag{9}$$

the evaluation step in (7) yields a function from e type objects to t type objects:

$$I(loves')(\textit{mary}) \tag{10}$$

$$= \{(\textit{rosinante}, 1), (\textit{pedro}, 1), (\textit{mary}, 0)\} \tag{11}$$

To compute the value of (6), we need,

$$\|mary'\|_{\mathcal{M},g_{[x\mapsto \textit{mary}]}} = I(mary') = \textit{mary} \tag{12}$$

feeding this into (10), we get 0 as the value of (6). Feeding 0 to the interpretation of $\rightarrow$ would yield a function $\{(0,1),(1,1)\}$. The next task is to compute the value for the consequent of the conditional with the extended environment and completing the iteration for $x \mapsto \textit{mary}$. The iteration has to go on until we are out of objects in D_e. The completion of the evaluation is left as an exercise.