Course: Modeltheoretic Semantics and the Syntax-Semantics Interface
Eddy Ruys and Yoad Winter

Lecture 2: Types and Model Structure

Topics: types, domains, characteristic functions, function application, the category to type matching

In the first lecture we have talked rather sloppily about models and various sets, functions and truth-values they contain. In this lecture we will go somewhat deeper into the *structure* of the model: what objects are there in the model, how can we describe them, and does this give us more insight into the relations between syntax and semantics?

As we have said, natural language expressions are taken to denote objects in the model. However, different expressions denote different kinds of things. For instance, we let *Tina* denote a member of the set of entities E, whereas *tall* denotes a *subset* of E and a sentence like *Tina is tall* denotes a truth value. We would like to have a systematic way to describe these distinctions. For this reason we divide the model into different *domains*, where each domain has a *type*: a label describing its internal structure. The three expressions mentioned above denote members of domains with different types.

The simplest types and the simplest domains are the *primitive* ones. There are two primitive types, each with its corresponding domain. One primitive type is called e (for $e$ntities) and its corresponding domain is the non-empty set of entities E, which we from now on denote by D_e (the e Domain). The other primitive type is called t (for $t$ruth-values) and its corresponding domain is the set $D_t = \{0, 1\}$. The proper name *Tina* takes its denotation from the D_e domain. In short, we say that proper names are of type e. A sentence like *Tina is tall* denotes a truth value in D_t.

All the non-primitive types are recursively defined on top of these two primitive types. To see how, let us start with an example. We have said that predicates like *tall* denote subsets of D_e. For instance, consider a model where the set of entities D_e is $\{\mathbf{tina'}, \mathbf{john'}, \mathbf{mary'}\}$. Let us indicate that only Tina and John are tall in this model. We can do that by letting *tall* refer to the set $T = \{\mathbf{tina'}, \mathbf{john'}\}$, a subset of D_e. Alternatively, we can say the same thing in a slightly different way: let *tall*, instead of denoting the set T, denote a corresponding *function* f_T from entities to truth-values, i.e. from D_e to D_t. For each member of the set of entities, f_T assigns it *true* if it is in T and *false* if it is not in T. Thus, we have:

$$f_T: \quad \mathbf{tina'} \mapsto 1 \quad \mathbf{john'} \mapsto 1 \quad \mathbf{mary'} \mapsto 0$$

f_T is called the *characteristic function* of the set $T \subset D_e$. For every element x of D_e, the truth value $f_T(x)$ indicates whether x is in T or not. In general, we may view the denotation of a predicate like *tall* either as a subset of the set of entities or as the characteristic function of such a set. We can move freely back and forth between these two perspectives, as they encode precisely the same information concerning which entities are tall. For the sake of the discussion in this lecture let us adopt the latter, functional perspective on predicates. The advantages of this step will become clear as we go along.

We obviously want predicates like *tall* to be able to denote *any* subset of D_e, or, as we now say, any function from D_e to D_t. Thus, we want the domain from which *tall* takes its denotation to be the set of *all functions from D_e to D_t*. In set theory this set of functions is denoted by $D_t^{D_e}$. This domain will be of type (et), for which we will often use the notation et as an abbreviation, erasing parentheses whenever this does not cause confusion.[1] Here are the eight members in D_{et} for the tripleton domain D_e we are dealing with:

$$
\begin{array}{llll}
f_1: & \mathbf{tina}' \mapsto 0 & \mathbf{john}' \mapsto 0 & \mathbf{mary}' \mapsto 0 \\
f_2: & \mathbf{tina}' \mapsto 0 & \mathbf{john}' \mapsto 0 & \mathbf{mary}' \mapsto 1 \\
f_3: & \mathbf{tina}' \mapsto 0 & \mathbf{john}' \mapsto 1 & \mathbf{mary}' \mapsto 0 \\
f_4: & \mathbf{tina}' \mapsto 0 & \mathbf{john}' \mapsto 1 & \mathbf{mary}' \mapsto 1 \\
f_5: & \mathbf{tina}' \mapsto 1 & \mathbf{john}' \mapsto 0 & \mathbf{mary}' \mapsto 0 \\
f_6: & \mathbf{tina}' \mapsto 1 & \mathbf{john}' \mapsto 0 & \mathbf{mary}' \mapsto 1 \\
f_7: & \mathbf{tina}' \mapsto 1 & \mathbf{john}' \mapsto 1 & \mathbf{mary}' \mapsto 0 \\
f_8: & \mathbf{tina}' \mapsto 1 & \mathbf{john}' \mapsto 1 & \mathbf{mary}' \mapsto 1
\end{array}
$$

In this way, a subset of D_e is described by a function in D_{et}, which must correspond to one of the rows in this array. The predicate *tall* can denote any of these eight functions when the set of entities D_e is the one specified. For instance, the function f_T above is function f_7 in this array.

To summarize, a proper name like *Tina* denotes an element of D_e, a sentence denotes an element of D_t and a predicate like *tall* denotes a function from the first domain to the latter, i.e. an element of D_{et}. Of course, when we consider more expressions, we are likely to need more types than these ones. How do we define them?

What we obviously want is a general strategy for defining types. The *tall* example has illustrated the way to do that: we defined a new type, et, and its corresponding domain on the basis of the given types e and t and their domains. Generally, if we are given the types τ and σ and their corresponding domains D_τ and D_σ, we can define the new type $\tau\sigma$ of a domain $D_{\tau\sigma}$ that consists of all the functions from D_τ to D_σ. In this way we can go on and use the new type and domain we get to create more new types and domains. For instance, we can use et and e to create the type $(et)e$ and its domain $D_{(et)e}$ of all functions from D_{et} to D_e. There are infinitely many types we can define using this inductive procedure. This is precisely what we do now, using the following two definitions.

Definition 1 (types) *e and t are types. If τ and σ are types then $(\tau\sigma)$ is a type.*

[1] In many places you will find the same type denoted by $\langle e, t\rangle$

Definition 2 (typed domains) *D_e is an arbitrary non-empty set. $D_t = \{0,1\}$. If τ and σ are types then $D_{\tau\sigma} = D_\sigma^{D_\tau}$, the set of functions from D_τ to D_σ.*

These two simple definitions summarize all we have discussed above. A model[2] is the infinite collection of all typed domains. Note that the primitive domain D_t is given, so in order to determine a model, all we need to do is to determine the other primitive domain: D_e. After we have done this, all the compound domains are inductively fixed by definitions 1 and 2.

Working with functions in this way as the building blocks of compound domains gives us an easy way to describe what is going on compositionally in analyzing a sentence. Consider a maximally simple sentence like *Tina smiles.* The sentence denotes a truth value simply because the subject denotes an entity of type e and the predicate denotes a function of type et: from e entities to truth-values. Once this function applies to the subject denotation we get a truth value. This operation of *function application* is crucial: it is the main operation that "glues" denotations together into new denotations. Let us exemplify the notation we use in more detail for the simple sentence *Tina smiles*:

- The word *Tina* denotes an element $\mathbf{tina}'$ of D_e, or in short: $\mathbf{tina}'_e$.
- The word *smiles* denotes a function $\mathbf{smile}'_{et}$.
- The sentence *Tina smiles* denotes the truth value in D_t which is the result of applying the function $\mathbf{smile}'$ to $\mathbf{tina}'$. This value is denoted $\mathbf{smile}'(\mathbf{tina}')$.

Note how easy it is to know from two types if function application is allowed between objects in the corresponding domains. When function application is allowed also the outcome type is obvious. For instance:

- e and et give t (as we have seen above)
- $(e(et))(et)$ and $e(et)$ give et
- $(e(et))(et)$ and et don't allow function application: neither of these types is a prefix of the other.

In general, thus, function application gives us from two denotations of types $\tau\sigma$ (the function type) and τ (the argument type) a new denotation of type σ: the result of applying the function to the argument.

The postulation of function application also facilitates the typing (=the determination of the type) of different expressions. For instance, consider the auxiliary verb *is* in the sentence *Tina is tall.* We have decided to let *Tina* get the type e and *tall* get the type et. The constituent *is tall* has to denote a function that applies to $\mathbf{tina}'$ and derives a truth-value. Hence, it must be, like *tall*, of type et. To get this, the denotation of *is* will be a function of type $(et)(et)$: when it applies to an et object like $\mathbf{tall}'$ it will give us an et object.

[2] what is more accurately called a *frame*

In fact, *is* will denote the *identity* function on predicates: the function that for each *et* predicate assigns the same predicate itself. Thus, the denotation of *is tall* is the same as the denotation of *tall*.

These typing considerations are summarized in figure 1. Note that now it is natural to use a more realistic structure than the one that was tentatively used in the first lecture for expository purposes.

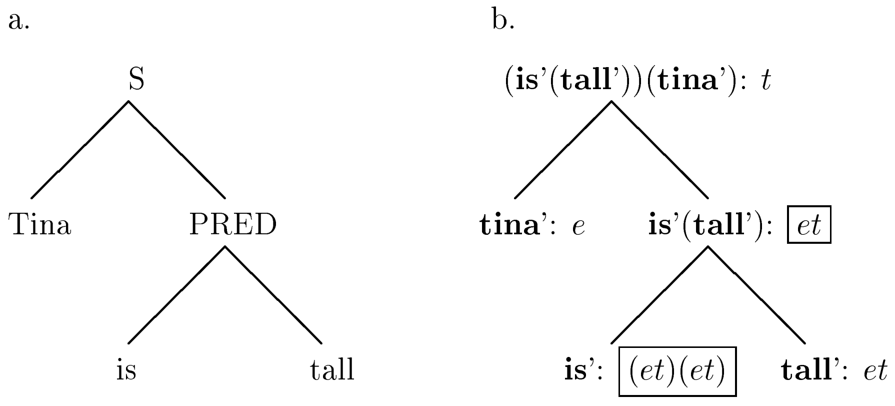

Figure 1: compositionality and typing

What we have done above is a kind of puzzle solving. We were given parts of the puzzle: the types for the denotations of the words *Tina* and *tall* and the type for the whole sentence. We had the structure of the sentence and sought types for the word *is* and the category PRED that would match the other types by allowing function application to work compositionally. Because we knew that the denotation of PRED would have to apply to an e denotation to produce a t denotation, we concluded that this category should be of type et. To generate this type, the denotation $\mathbf{is}'$ has to apply to the et type denotation of *tall* and give another et function. Thus, it has to be of type $(et)(et)$.

We should stress the relations between *typing* and *sentence structure*. For instance, the $(et)(et)$ typing of *is* is motivated by the structure in figure 1, but not necessarily by different structures that might have been postulated. Suppose tentatively that we gave the sentence the implausible structure of figure 2, in which *Tina* and *is* are assumed to form a constituent. In this structure the typing of *is* could not have remained $(et)(et)$: functions of this type apply to et type objects and not to e type entities. A possibility would be to change the typing of *is* to ee: a function from entities to entities. This would allow the denotation $\mathbf{is}'$ to apply to $\mathbf{tina}'_e$ and produce an e type denotation for the category FUNNY, to which the predicate $\mathbf{tall}'$ could apply and produce a truth value.

This simple example shows that in assigning a type to a particular expression, structural considerations play a major role: the syntactic configurations in which an expression appears often affect the semantic type we choose to assign to it.

So far, types have just put some order in things that were already done in

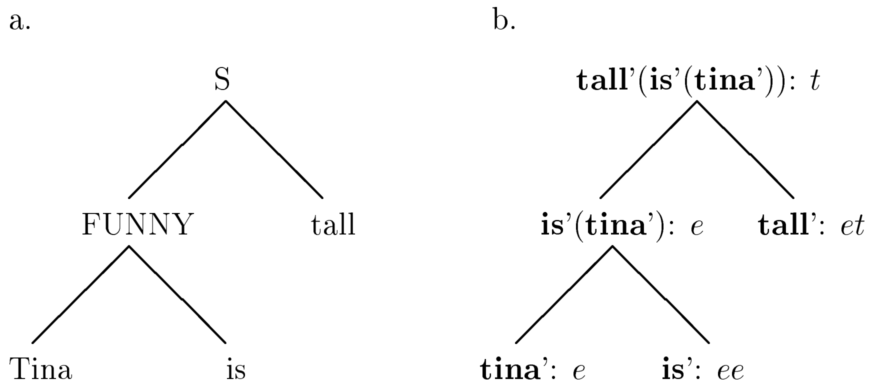

Figure 2: a funny structure

the previous lecture. But the idea to let expressions denote typed objects is very helpful when we want to define denotations of new expressions. For instance, what does the transitive verb *see* denote? Consider a sentence like *Tina saw Mary*. Compositionally, the constituent *saw Mary* is going to have the type et: like intransitive predicates (e.g. *tall*) its denotation applies to entities to produce a truth value. Since we have decided to let *Mary* be of type e there is no question as for what type the verb *see* has to get: the type of functions from e objects to et objects. In short: $e(et)$. Similarly, a ditransitive predicate like *show*, as in *Tina showed Mary John* denotes objects of type $e(e(et))$: functions from entities to functions from entities to functions from entities to truth values. It is surely convenient that we have so many types at our disposal!

Note that in the discussion above we have made two implicit assumptions:

1. *Each syntactic category has a unique type.*
   For instance: since *Tina* and *Mary* are of the same syntactic category (proper name/noun phrase), they must have the same type: e.

2. *Function application is the only way to combine denotations.*
   For instance: we assumed that the only way to let the constituent *is tall* have the type et is by assigning the type $(et)(et)$ to the copula *is*. We reached this conclusion because $(et)(et)$ is the only type with which we can get an et object after application to an et object. Of course, there might be other ways to combine denotations besides function application, in which case the conclusion would not necessarily follow.

Adopting these two assumptions puts a very strong constraint on the syntax-semantics interface. The first assumption means that once you determine the type of an expression, *all* expressions of the same category must have the same type. In other words: whenever we want two expressions to denote objects of a different type we must take care that the syntax can assign the two expressions different categories. Also the second principle is highly restrictive: once you determine the type (say $\tau_1\tau_2$) of one of two sisters, the other sister must either be a function over this type (i.e. $(\tau_1\tau_2)\tau_3$) or an argument of this function (i.e.

τ_1). Furthermore, the mother must be either of type τ_3 (in the first case) or of type τ_2 (in the latter case). These two possibilities are illustrated in figure 3.

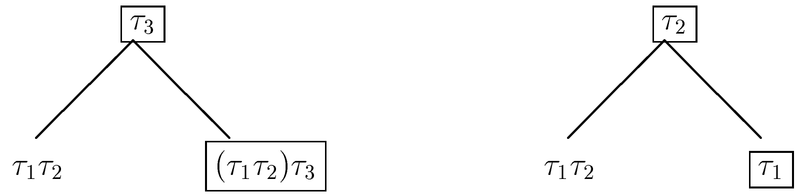

Figure 3: possibilities with function application

Such a strict organization of the grammar is challenging, and therefore theoretically desired: it leaves us with only a small amount of possibilities to develop syntactic theories with suitable typings of lexical items. At the present stage of semantic theory, however, it is debatable whether such a strict configuration can succeed as a linguistic program. These restrictions may be too strict for natural language, and attempts are being made to relax them in principled ways. Nevertheless; since these two constraints are simple and methodologically convenient, we will adopt them throughout the first part of this course as an illustration of the power of modeltheoretic techniques.

A note on characteristic functions: We have shown above that we can freely switch between a set denotation of a predicate like *tall* and the corresponding characteristic function of type *et*. The situation is in fact more general. Consider the transitive predicate *see*. Intuitively, such a predicate can be viewed as a *two-place relation* between entities: a subset of the cartesian product $E \times E$. The relation will specify which entities see which entities. For instance, in case $E = \{\mathbf{tina}', \mathbf{john}', \mathbf{mary}\}$, we can describe a situation like the one given in (1) by letting *see* denote the relation S in (2).

(1) Tina saw only Mary, John didn't see anyone and Mary saw everyone (including herself).

(2) $S = \{\langle \mathbf{tina}', \mathbf{mary}' \rangle, \langle \mathbf{mary}', \mathbf{tina}' \rangle, \langle \mathbf{mary}', \mathbf{john}' \rangle, \langle \mathbf{mary}', \mathbf{mary}' \rangle\}$

We have proposed that *see* gets the type $e(et)$. Every function like this uniquely characterizes a subset of $E \times E$, and vice versa. For instance, the $e(et)$ function corresponding to S is:

$$
\begin{array}{llll}
f_S: & \mathbf{tina}' & \mapsto & [\mathbf{tina}' \mapsto 0 \quad \mathbf{john}' \mapsto 0 \quad \mathbf{mary}' \mapsto 1] \\
 & \mathbf{john}' & \mapsto & [\mathbf{tina}' \mapsto 0 \quad \mathbf{john}' \mapsto 0 \quad \mathbf{mary}' \mapsto 1] \\
 & \mathbf{mary}' & \mapsto & [\mathbf{tina}' \mapsto 1 \quad \mathbf{john}' \mapsto 0 \quad \mathbf{mary}' \mapsto 1]
\end{array}
$$

The function f_S assigns each entity x in D_e a function in D_{et} that characterizes the set of entities that saw x. For instance, the VP *saw Mary* denotes the characteristic function of the set of entities that saw Mary, namely: $\{\mathbf{tina}', \mathbf{mary}'\}$. The reasoning behind this choice is that the denotation of *see* compositionally applies to the denotation of the *object* in the sentence. The resulting denotation of the VP applies to the subject denotation. Hence, the first argument of the denotation of the verb should be the object denotation. Generally, the characteristic function f_S of a relation S is defined by:

$$(f_S(x))(y) = 1 \text{ iff } \langle y, x\rangle \in S$$

Note that $f_S(x)$ is itself a characteristic function. In our case, the one characterizing the set of entities that saw x. The successive application $(f_S(x))(y)$ is sometimes written in short $f_S(x)(y)$. This switch in the intuitive order of the arguments is confusing upon first encounter. However, it is necessary if we want to be able to speak of the denotation of constituents like *saw Mary*: it is much more natural then to think of *Mary* as the first argument of the verb's denotation and not as in the intuitive relation S. After some exercising you will surely get used to this and realize that the method is completely innocuous. Moreover, the procedure we have used for moving back and forth between the intuitively convenient sets/relations and their corresponding characteristic functions is easily extended to many other useful types.

Exercises

1. Give types corresponding to the following descriptions:

   - functions from truth-values to functions from truth-values to truth-values
   - functions from functions from entities to truth-values to truth-values
   - functions from functions from functions from entities to truth-values to entities to truth-values

   Describe the following types:

   - $e(et)$
   - $(ee)t$
   - $(e(et))(et)$

2. Let $D_e = \{a, b\}$. Write down all the members of D_{ee}, $D_{e(et)}$ and $D_{(ee)t}$. Notice the difference between the last two domains.

3. Write down all the members of D_{tt}. Why don't you need to know D_e in order to do that?

4. Write down the characteristic functions for the following sets/relations. Assume that $D_e = \{a, b, c, d\}$.

   - The set $R = \{b, d\}$ describing a denotation of the predicate *run*.
   - The two-place relation $K = \{\langle a, a\rangle, \langle b, c\rangle, \langle b, d\rangle, \langle d, a\rangle\}$ describing a denotation of the verb *kiss*.
   - The three-place relation $S = \{\langle a, a, a\rangle, \langle b, d, d\rangle, \langle d, a, c\rangle\}$ describing a denotation of the verb *show*. Read $\langle x, y, z\rangle \in S$ as "*x showed y z*".

5. For each of the following pairs of types say whether function application can operate. If it can, give the resulting type.

   - $(ee)(tt) \quad tt$
   - $(et)t \quad ((et)t)(e(tt))$
   - $(et)t \quad (((et)t)e)((et)t)$
   - $e(tt) \quad et$
   - $e \quad e((e(e(et))))$

6. Assume (unrealistically) that the expression "*it is not the case that*" in sentence (i) below is a constituent. What type would you give it? What would be its denotation? Give two structures as in figure 1 in the lecture notes. Show an equivalent sentence to (i) in order to support your proposal (hint: get back to lecture 1).

   (i) It is not the case that Tina is tall.

7. Give a binary tree structure for sentence (ii) below.
   Deduce from this structure a type for the adverb *quickly*.
   Describe in words the type of functions denoted by this word (as in exercise 1 above) and fill in two structures (as you were requested in exercise 6).
   What restriction on the set of functions *quickly* can denote can you infer from the entailment (ii)$\Rightarrow$(iii)?

   (ii) Tina ran quickly.
   (iii) Tina ran.

8. Give a type for the word *very* in sentence (iv) below. Give the same word a type also in (v), in consideration of your answer to exercise 7.
   You have reached a challenge for the principle of "one type per category". Can you think of a way to maintain the idea despite this fact? Alternatively, can you relax the restriction in a principled way?

   (iv) Tina is very tall.
   (v) Tina ran very quickly.

9. In the figure below we have left two types in the structure unspecified as X and Y. Give two possibilities to substitute types instead of X and Y so that function application can go on safely. Can you think of the relevance of this puzzle?