Computational Linguistics: Formal Semantics and Compositionality

Raffaella Bernardi

University of Trento

Contents

1. Seen to be done

We have seen:

- how lexical meaning is represented by sets.
- how to go from a set-theoretical representation to a functional one.

Today we are going to see that:

- lambda terms represent functions
- lambda calculus accounts for the composition of the meaning representations of a sentence

2. Frege: From words to sentences

Complete vs. Incomplete Expressions Frege made the following distinction:

- A sentence is a **complete** expression, it's reference is the truth value.
- A proper name stands for an object and is represented by a constant. It's a **complete** expression.
- A predicate is an **incomplete** expression, it needs an object to become complete. It is represented by a function. Eg. "left" needs to be completed by "Raj" to become the complete expression "Raj left".

Principle of Compositionality: The meaning of a sentence is given by the meaning of its parts and by the compositionality rules.

3. Lambda Calculus

- It has a **variable binding operators** λ. Occurrences of variables bound by λ should be thought of as place-holders for missing information: they explicitly mark where we should substitute the various bits and pieces obtained in the course of semantic construction.
- An **operation** called β-conversion performs the required substitutions.

3.1. Lambda Calculus: Function and lambda terms

Function $f : X \to Y$. And $f(x) = y$ e.g. $SUM(x, 2)$ if $x = 5$, $SUM(5, 2) = 7$.

- $\lambda x.x$
- $\lambda x.(x + 2)$ [SUM(x,2)]
- $\underbrace{(\lambda x.(x + 2))}_{function}\ \underbrace{5}_{argument}$
- $\underbrace{(\lambda x.(x + 2))}_{function}\ \underbrace{5}_{argument} = 5 + 2$
- $((\lambda y.\lambda x.(x + y))\ \underbrace{5}_{argument2}\ )\ \underbrace{2}_{argument1} = (\lambda x.(x + 5))\ 2 = 2 + 5$

3.2. Lambda-terms: Examples

Here is an example of lambda terms:

$$\lambda x.\texttt{left}(x)$$

The prefix $\lambda x.$ binds the occurrence of x in $\texttt{student}(x)$. We say it **abstracts** over the variable x. The purpose of abstracting over variables is to mark the slots where we want the substitutions to be made.

To glue `vincent` with "left" we need to apply the lambda-term representing "left" to the one representing "Vincent":

$$\lambda x.\texttt{left}(x)(\texttt{vincent})$$

Such expressions are called **functional applications**, the left-hand expression is called the **functor** and the right-hand expression is called the **argument**. The functor is applied to the argument. Intuitively it says: fill all the placeholders in the functor by occurrences of the term `vincent`.

The substitution is performed by means of β-conversion, obtaining `left(vincent)`.

3.3. Functional Application

Summing up:

- FA has the form: Functor(Argument). E.g. $(\lambda x.love(x, mary))(john)$
- FA triggers a very simple operation: Replace the λ-bound variable by the argument. E.g. $(\lambda x.love(x, mary))(john) \Rightarrow love(john, mary)$

3.4. β-conversion

Summing up:

1. Strip off the λ-prefix,
2. Remove the argument,
3. Replace all occurences of the λ-bound variable by the argument.

For instance,

1. $(\lambda x.love(x, mary))(john)$
2. $love(x, mary)(john)$
3. $love(x, mary)$
4. $love(john, mary)$

3.5. Exercise

Give the lambda term representing "saw".

(a) Build the meaning representation of "John saw Mary" starting from: John: `j`, Mary: `m` saw: $\lambda x.\lambda y.\texttt{saw}(y, x)$

4. Lambda-Terms Interpretations

We've seen that a Model is a pair consisting of a domain ($\mathcal{D}$) and an interpretation function ($\mathcal{I}$).

- In the case of FOL we had only one domain, namely the one of the objects/entities we were reasoning about. Similarly, we only had one type of variables. Moreover, we were only able to speak of propositions/clauses.
- λ-terms speak of functions and we've used also **variables standing for functions**. Therefore, we need a more complex concept of interpretation, or better a more **complex concept of domain** to provide the fine-grained distinction among the objects we are interested in: truth values, entities and functions.
- For this reason, the λ-calculus is of Higher Order.

4.1. Models, Domains, Interpretation

In order to interpret meaning representations expressed in FOL augmented with λ, the following facts are essential:

- **Sentences**: Sentences can be thought of as referring to their truth value, hence they denote in the the domain $D_t = \{1, 0\}$.
- **Entities**: Entities can be represented as constants denoting in the domain D_e, e.g. $D_e = \{\texttt{john}, \texttt{vincent}, \texttt{mary}\}$
- **Functions**: The other natural language expressions can be seen as incomplete sentences and can be interpreted as **boolean functions** (i.e. functions yielding a truth value). They denote on functional domains $D_b^{D_a}$ and are represented by functional terms of type $(a \to b)$.

  For instance "walks" misses the subject (of type e) to yield a sentence (t).

  - denotes in $D_t^{D_e}$
  - is of type $(e \to t)$,
  - is represented by the term $\lambda x_e(\texttt{walk}(x))_t$

4.2. Lambda-calculus: some remarks

The pure lambda calculus is a theory of functions as rules invented around 1930 by Church. It has more recently been applied in Computer Science for instance in "Semantics of Programming Languages".

In Formal Linguistics we are mostly interested in lambda conversion and abstraction. Moreover, we work only with typed-lambda calculus and even more, only with a fragment of it.

The types are the ones we have seen above labeling the domains, namely:

- e and t are types.
- If a and b are types, then $(a \to b)$ is a type.

5. Lambda Terms represent functions

For instance "walk" is a set of entities (those entities which walk), hence it's a function:

- denotes in $D_t^{D_e}$
- is of type $(e \to t)$,
- is represented by the term $\lambda x_e(\texttt{walk}(x))_t$

5.1. Exercises: Lambda terms

1. Build a model for a situation of your choice. Specify the domains of interpretation, the set-theoretical representation of words, and their corresponding typed lambda terms.

2. If an intransitive verb (e.g. "walk") is represented by a unary-function, a transitive verb (e.g. "know") by a binary-function, what is the function representing a ditransitive verb (e.g. "gave")?

3. What could be the meaning representation of an adjective (e.g. "smar")?

$$\lambda X.\lambda x.X(x) \wedge Smart(x)$$

it's the intersection between the set X entities and the set of Smart entities.