Logica & Linguaggio: Calcolo di Lambda

RAFFAELLA BERNARDI

UNIVERSITÀ DI TRENTO

P.ZZA VENEZIA, ROOM: 2.05, E-MAIL: BERNARDI@DISI.UNITN.IT

Contents

1. Lambda Calculus

FOL augmented with Lambda calculus can capture the "how" and accomplish tasks 2 and 3.

- 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.

1.1. 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)`.

1.2. 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. $(\lambda x.love(x, mary))(john) \Rightarrow love(john, mary)$

1.3. 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. $love(john, mary)$

1.4. Exercise

Give the lambda term representing a transitive verb.

(a) Build the meaning representation of "John saw Mary" starting from:

- John: $\texttt{j}$
- Mary: $\texttt{m}$
- saw: $\lambda x.\lambda y.\texttt{saw}(y,x)$

(b) Build the parse tree of the sentence.

(c) Compare what you have done to assembly the meaning representation with the way you have built the tree.

1.5. α-conversion

Warning: Accidental bounds, e.g. $\lambda x.\lambda y.\texttt{Love}(y,x)(y)$ gives $\lambda y.\texttt{Love}(y,y)$. We need to rename variables before performing β-conversion.

α-conversion is the process used in the λ-calculus to rename bound variables. For instance, we obtain

$\lambda x.\lambda y.\texttt{Love}(y,x)$ from $\lambda z.\lambda y.\texttt{Love}(y,z)$.

When working with lambda calculus we always α-covert before carrying out β-conversion. In particular, we always rename all the bound variables in the functor so they are distinct from all the variables in the argument. This prevents accidental binding.

2. Lambda-Terms Interpretations

In the first part of the course you've seen that a Model is a pair consisting of a domain ($\mathcal{D}$) and an interpretation function (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.

2.1. 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 $\lambda x_e(\texttt{walk}(x))_t$

2.2. 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 $(a \rightarrow b)$ is a type.

3. Determiners

Which is the lambda term representing quantifiers like “nobody”, “everybody”, “a man”or “every student” or a determiners like “a”, “every” or “no” ?

We know how to represent in FOL the following sentences

- “Nobody left”
  $\neg\exists x.\texttt{left}(x)$
- “Everybody left”
  $\forall x.\texttt{left}(x)$
- “Every student left”
  $\forall x.\texttt{Student}(x) \rightarrow \texttt{left}(x)$
- “A student left”
  $\exists x.\texttt{Student}(x) \wedge \texttt{left}(x)$
- “No student left”
  $\neg\exists x.\texttt{Student}(x) \wedge \texttt{left}(x)$

But how do we reach these meaning representations starting from the lexicon?

3.1. Determiners (cont'd)

Let's start representing "a man" as $\exists x.man(x)$. Applying the rules we have seen so far, we obtain that the representation of "A man loves Mary" is:

$$love(\exists x.man(x), mary)$$

which is clearly wrong.

Notice that $\exists x.man(x)$ just isn't the meaning of "a man". If anything, it translates the complete sentence "There is a man".

We will look at this next time.

4. The Three Tasks Revised

Task 1 Specify a reasonable *syntax* for the natural language fragment of interest. *We can do this using CG.*

Task 2 Specify semantic representations for the *lexical items*. *We know what this involves*

Task 3 Specify the *translation* of an item $\mathcal{R}$ whose parts are $\mathcal{F}$ and $\mathcal{A}$ with the help of functional application. That is, we need to specify which part is to be thought of as functor (here it's $\mathcal{F}$), which as argument (here it's $\mathcal{A}$) and then let the resultant translation $\mathcal{R}'$ be $\mathcal{F}'(\mathcal{A}')$. *We know that* β*-conversion (with the help of* α*-conversion), gives us the tools needed to actually construct the representation built by this process.*

5. Summing up: Constituents and Assembly

Let's go back to the points where FOL fails, i.e. constituent representation and assembly. The λ-calculus succeeds in both:

Constituents: each constituent is represented by a lambda term.

John: $\mathtt{j}$ knows: $\lambda xy.(\mathtt{know}(x))(y)$ read john: $\lambda y.\mathtt{know}(y, \mathtt{j})$

Assembly: function application ($\alpha(\beta)$) and abstraction ($\lambda x.\alpha[x]$) capture composition and decomposition of meaning representations.