Computational Linguistics: Lambda Calculus and NL

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

P.zza Domenicani, Room: 2.28, e-mail: bernardi@inf.unibz.it

Contents

1. Historical Remark: Montague

Montague ('74) was the first to seriously propose and defend the thesis that the relation between syntax and semantics in a natural language such as English could be viewed as not essentially different from the relation between syntax and semantics in formal language such as the language of FOL. The framework he developed is known as "Montague's Universal Grammar" and its main components are:

- Model-Theory
- The principle of "Compositionality" which is due to Frege (1879).
- The λ-calculus which was invented by Church in the 1930's.
- Categorial Grammar which is due to Ajdukiewicz ('35) and Bar-Hill ('53). It's a (context free) Grammar based on functional application.

The novelty of Montagues' work was to apply them to natural language in a uniform framework.

We will study Categorial Grammar after the winter break.

2. Lambda-Terms Interpretations

In the Logic course you've seen that an Interpretation 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 where 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. Domains

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: formally

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 interested in **typed-lambda calculus**.

Types The types are the ones we have seen above labeling the domains

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

The two important operations are β **conversion** and **abstraction**.

conversion $(\lambda x_a.M)(N_a)$ is equivalent to M with every occurrence of x bound by λ replaced by N.

abstraction If α is expression of type a and v is a variable of type b, then $\lambda v\alpha$ is an expression of type $(a \rightarrow b)$.

2.3. Relative Pronouns

For instance, "which John read [. . .]":

We know how to represent the noun phrase "John" and the verb "read", namely, as `john` and $\lambda x.y.\texttt{read}(y, x)$.

What is the role of "which" in e.g. "the book which John read is new"?

The term representing "which" has to express the fact that it is replacing the role of a noun phrase in subject (or object position) within a subordinate sentence while being the subject (object) of the main sentence:

$$\lambda X.\lambda Y.\lambda z.X(z) \wedge Y(z)$$

The double role of "which" is expressed by the double occurrence of z.

2.4. Relative Pronoun (Cont'd)

Recall,

$$\lambda X.\lambda Y.\lambda z.X(z) \wedge Y(z)$$

i. read u: $\lambda y(\texttt{read}(y, u))$
ii. John read u: $\texttt{read}(\texttt{j}, u)$
iii. John read: $\lambda u.\texttt{read}(\texttt{j}, u)$
iv. which John read: $\lambda Y.\lambda z.\texttt{read}(\texttt{j}, z) \wedge Y(z)$

- at the syntactic level we said that the relative pronoun "which" plays the role of the verb's object and it leaves a gap in the object position.
- Semantically, the gap is represented by the u on which the relative pronoun forces the abstraction [iii.] before taking its place.

3. Ambiguities

Starting from:

john: $\mathtt{j}$	book: $\lambda x(\mathtt{book}(x))$
read: $\lambda x.\lambda y.(\mathtt{read}(y,x))$	didn't: $\lambda X.\neg X$
a: $\lambda X.\lambda Y(\exists x.X(x) \wedge Y(x))$	

Build the meaning representation for "John didn't read a book".

a. $\exists x.\mathtt{book}(x) \wedge \neg\mathtt{read}(\mathtt{j},x)$ [A > NOT]

b. $\neg\exists x.\mathtt{B}(x) \wedge \mathtt{read}(\mathtt{j},x)$ [NOT > A]

After Christmas we will put together syntax and semantics (syntax-semantic interface). Now we will practice on the λ-calculus.

4. Exercise 1: Well formed formula

Let j be a constant of type e; M of type $e \to t$; S of type $((e \to t) \to (e \to t))$, and P of type $(e \to t) \to t$. Furthermore, x is a variable of type e, and Y a variable of type $(e \to t)$.

Determine which of the following is well-formed, given its type.

1. $(\lambda x.M(x))(P)$.
2. $(\lambda x.M(x))(j)$.
3. $\lambda x.M(j)$.
4. $S(\lambda x.M(x))$.
5. $(\lambda Y.Y(j))(M)$
6. $\lambda x.(M(x) \wedge M(j))$
7. $(\lambda x.M(x)) \wedge M(j)$

5. Exercise 2: λ-conversion

Let j be a constant of type e; M of type $(e \rightarrow t)$, and A of type $e \rightarrow (e \rightarrow t)$. Furthermore, x and y are variables of type e, and Y is a variable of type $e \rightarrow t$. Reduce the following expression as much as possible by means of λ-conversion.

1. $\lambda x(M(x))(j)$
2. $\lambda Y(Y(j))(M)$
3. $\lambda x \lambda Y(Y(x))(j)(M)$
4. $\lambda x \forall y(A(x)(y))(j)$
5. $\lambda x \forall y(A(x)(y))(y)$
6. $\lambda Y(Y(j)) \lambda x(M(x))$
7. $\lambda Y \forall x(Y(x)) \lambda y(A(x)(y))$

6. Exercise 3: λ-calculus and NL

Given,

- new $\lambda X_{(e \to t)}.\lambda x_e(X(x) \wedge new(x))$
- book $\lambda x_e.(book(x))_t$
- student $\lambda x_e.(student(x))_t$
- a $\lambda X_{(e \to t)}\lambda Y_{(e \to t)}(\exists x_e.X(x) \wedge Y(x))$
- john j
- read $\lambda x_e.\lambda y_e.read(y, x)$
- left $\lambda y_e.left(y)$

build the meaning representation for

1. John read a book

2. A new student left
3. John read a new book
4. A student read a book

7. Exercise 4: λ-calculus and NL

You know that e.g.

"Every student left" can be represented as $\forall x.Student(x) \rightarrow Left(x)$; "No student left" as $\neg\exists x.Student(x) \wedge Left(x)$, "John dind't leave" as $\neg leave(j)$; "John or mary left" as $left(j) \vee left(m)$; "Mary left and John stayed" as $left(m) \wedge stay(j)$.

Use them to give the lambda terms for the words below.

1. every
2. everybody
3. no
4. didn't
5. did
6. and
7. or

8. Grading

Final Exam : When in February? (it must be on Saturday!)

Critiques : On LCT seminars or Papers http://www.inf.unibz.it/~bernardi/Courses/ComLing04/#critiques (next week you will find more!)

Projects : http://www.inf.unibz.it/~bernardi/Courses/ComLing04/#projects