Computational Linguistics: Semantics

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

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

Contents

1. CL: what have we achieved?

Main challenge of CL: to deal with ambiguity at the different levels.

Which kind of ambiguities are the following ones?

- La vecchia porta sbatte.
- La vecchia porta la sbarra.
- John saw the man with the telescope.
- John saw the woman in the park with the telescope. He was at home.

We have seen how to recognize/parse these sentences so to obtain different parse trees whenever necessary.

2. Semantics

Semantics: it's the study of the meaning of words and the reference of linguistic inputs. The former is studied in **Lexical Semantics** and the latter in **Formal Semantics**.

Lexical Semantics Words are seen as having a systematic structure that governs what they mean, how they **relate to actual entities** and how they can be used. Studies on this topic result into e.g. Dictionary or Ontologies like WordNET.

February: Workshop in TN http://tcc.itc.it/events/meaning2005/

whereas we will look at Formal Semantics.

3. Formal Semantics

The main questions are:

1. What does a given sentence mean?
2. How is its meaning built?
3. How do we infer some piece of information out of another?

The first and last question are closely connected.

In fact, since we are ultimately interested in understanding, explaining and accounting for the entailment relation holding among sentences, we can think of **the meaning of a sentence as its truth value**.

3.1. Logical Approach

To tackle these questions we will use Logic, since using Logic helps us answering the above questions at once.

1. Logics have a precise semantics in terms of **models** —so if we can translate/represent a natural language sentence S into a logical formula ϕ, then we have a precise grasp on at least part of the meaning of S.

2. Important **inference problems** have been studied for the best known logics, and often good **computational implementations** exists. So translating into a logic gives us a handle on inference.

When we look at these problems from a computational perspective, i.e. we bring in the implementation aspect too, we move from Formal Semantics to **Computational Semantics**.

3.2. Formal Semantics: What

What does a given sentence mean?

The meaning of a sentence is its truth value. Hence, this question can be rephrased in "Which is the meaning representation of a given sentence to be evaluated as true or false?"

- **Meaning Representations**: Predicate-Argument Structures are a suitable meaning representation for natural language sentences. E.g.

  the meaning representation of "Vincent loves Mia" is $\texttt{loves}(\texttt{vicent}, \texttt{mia})$

  whereas the meaning representation of "A student loves Mia" is $\exists x.\texttt{student}(x) \wedge \texttt{loves}(x, \texttt{mia})$.

- **Interpretation**: a sentence is taken to be a proposition and its meaning is the truth value of its meaning representations. E.g.

  $[\![\exists x.\texttt{student}(x) \wedge \texttt{left}(x)]\!] = 1$ iff standard FOL (First Order Logic) definitions are satisfied.

3.3. Formal Semantics: How

How is the meaning of a sentence built?

To answer this question, we can look back at the example of "Vincent loves Mia". We see that:

- "Vincent" contributes the constant `vincent`
- "Mia" contributes the constant `mia`
- "loves" contributes the relation symbol `loves`

This observation can bring us to conclude that the **words** making up a sentence contribute all the bits and pieces needed to build the sentence's meaning representation.

In brief, **meaning flows from the lexicon**.

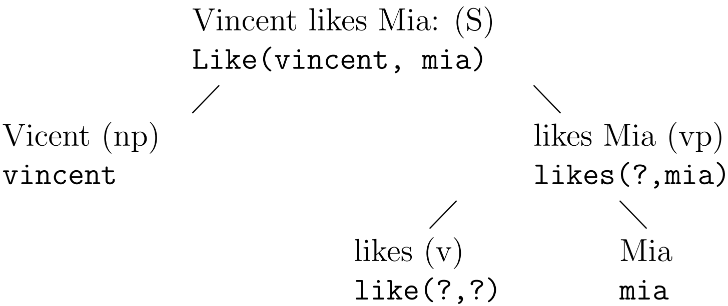
3.4. Formal Semantics: How (cont'd)

But,

- What does "a" contribute to in "A student loves Mia"?
- Why the meaning representation of "Vincent loves Mia" is not `love(mia,vincent)`?

The missing ingredient is the **syntactic structure**! [Vincent [loves$_v$ Mia$_{np}$]$_{vp}$]$_s$.

3.5. Formal Semantics: How (Cont'd)



Briefly, **syntactic structure guiding gluing**.

3.6. Compositionality

The question to answer is: “How can we specify in which way the bit and pieces combine?”

1. Meaning (representation) ultimately flows from the lexicon.
2. Meaning (representation) are combined by making use of syntactic information.
3. The meaning of the whole is function of the meaning of its parts, where “parts” refer to substructures given us by the syntax.

3.7. Ambiguity

A single linguistic sentence can legitimately have different meaning representations assigned to it.

For instance, "John saw a man with the telescope"

a.	John [saw [a man [with the telescope]$_{pp}$]$_{np}$]$_{vp}$	$\exists x.\mathtt{Man}(x) \wedge \mathtt{Saw}(j,x) \wedge \mathtt{Has}(x,t)$
b.	John [[saw [a man]$_{np}$]$_{vp}$ [with the telescope]$_{pp}$]$_{vp}$	$\exists x.\mathtt{Man}(x) \wedge \mathtt{Saw}(j,x) \wedge \mathtt{Has}(j,t)$

Different parse trees result into different meaning representations!

3.8. How far can we go with FOL?

FOL can capture the **what** but cannot capture the **how**, i.e.

Constituents: it cannot capture the meanings of constituents.

Assembly: it cannot account for meaning representation assembly.

4. Building Meaning Representations

To build a meaning representation we need to fulfill three tasks:

Task 1 Specify a reasonable **syntax** for the natural language fragment of interest.

Task 2 Specify semantic representations for the **lexical items**.

Task 3 Specify the **translation** of constituents **compositionally**. That is, we need to specify the translation of such expressions in terms of the translation of their parts, parts here referring to the substructure given to us by the syntax.

Moreover, when interested in Computational Semantics, all three tasks need to be carried out in a way that leads to computational implementation naturally.

We have looked at Task 1 in lecture 2 and 3 (formal grammars) and at their computational side (Implementation in Prolog, Recognition and Parsing) during the Labs.

Today we will look at the other two tasks.

5. 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.

5.1. 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 the placeholders in the functor by occurrences of the term `vincent`.

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

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

5.4. Exercise

Give the lambda term representing a transitive verb.

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

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

(b) Build the parse tree of the sentence by means the bottom-up method.

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

5.5. 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.

- “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?

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

6.3. Determiners (Cont'd)

Let's start from what we have, namely "man' and "loves Mary":

$\lambda y.man(y)$, $\lambda x.love(x, mary)$.

Hence, the term representing "a" is:

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

Try to obtain the meaning representation for "a man", and the "a man loves Mary".

By β-conversion twice we obtain that "a man" is $\lambda Y.\exists z.\mathtt{Man}(z) \wedge Y(z)$, and then $\exists z.\mathtt{Man}(z) \wedge love(z, mary)$

7.1. Scope Ambiguities

Can you think of other expressions that may cause scope ambiguity?

John **think** a student left

Does the student exist or not?

a. $\exists x.think(j, left(x))$

b. $think(j, \exists x.left(x))$

We will see more about this tomorrow.

7.2. Dependencies

While studying the syntax of natural language, we have seen that important concepts to account for are local and long-distance dependencies.

The λ-operator gives us (more or less) a way to represent this link semantically.

For instance, in $\lambda x.\lambda y.like(y, x)$ we express that the dependency of the subject and object from the verb.

Local dependencies But the calculus gives us also a natural way to handle long-distance dependencies: eg. relative pronouns.

7.3. Relative Pronouns

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

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

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

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.

7.4. Relative Pronoun (Cont'd)

Recall,

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

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: $\lambda Y.\lambda z.\texttt{read}(\texttt{j}, z) \land Y(z)$

- 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.

8. 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.

9. 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.

9.1. 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 $(a \rightarrow b)$.

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

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

9.2. 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.

10. The Three Tasks Revised

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

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.**

In Lab 1 and 2 we have seen that the solution proposed for task 1 leads itself to computational implementation naturally. In the next lab we will see that this holds for task 2 and 3 too (though we want go into the detail of it. If you are interested in it and you know Prolog already: good topic for a project!)

11. 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.

12. Inference

We've said that an important ultimate question in Formal Semantics is "How do we infer some piece of information out of other?"

We have seen how to use logic to represent natural language input.

Therefore the question reduces to the question of "Which are the inference tools for the logic we used?".

For instance, a tool you already know is the "Tableaux Method" (**Logic** Course by Franconi).

In the **Computational Logic** course (by Filottrani) you will study the computational properties of this tool and get hands on experience implementing it in PROLOG.

13. Inference (Cont'd)

Inference play a crucial role also at the discourse level, which is the topic of tomorrow class. For instance,

"Jon has a rabbit. The tail is white and fluffy".

"The tail" of whom?

I know that most rabbits have a tail, hence Jon's rabbit has a tail. Therefore, I can interpret "the tail" to be of Jon's rabbit.

14. The need for semantic representation

We shall also assume a framework in which we are attempting to communicate in natural language with some computer system which has its own internal representation of "knowledge" in some symbolic form.

The natural language front end (NLFE) has to formulate queries, commands and statements in a way that allow appropriate responses to be produced by the "back-end" system.

This can be done in two ways, using:

- Syntactic Representation
- Meaning Representation

14.1. Syntactic Representation

In this framework, it would be possible in principle to have the syntactic form of an input sentence (i.e. the syntax tree) act directly upon the stored information within the back-end knowledge based system, without any intermediate representations being constructed. That is, the response of the system, whether it be the moving of a robot arm, or the printing out of information, would be computed directly from the syntax tree.

This approach is **rarely followed**, as it leads to unnecessary complications. There are certain regularities about how each natural language expresses information, and there may be several different ways of expressing essentially the same request. If the syntactic tree is directly interpreted, then all the (linguistic) knowledge about these **grammatical variations** (which are irrelevant to the work of the back-end) has to be built-in to the interpreter which controls the back-end responses. Minor alterations to the grammatical coverage of the front-end then necessitate changes to this response-generator.

14.2. Meaning Representation

It is therefore normal to have the NLFE formulate its query (or command, or whatever) in some specialised notation for semantic representation, and then pass this on to further levels of the system for processing.

The superficial details of how the request was expressed in English can then be handled by separate linguistic rules, without the back-end having to include such information – the semantic representation of the input is a "canonical form". (This is a very similar methodological argument to that which supports splitting the earlier phase of processing into two stages, syntax and semantic interpretation).

Much of the research that goes on in NL semantics is concerned with the design of suitable formalisms for this intermediate language.

14.3. Meaning Representation (Cont'd)

It is important to note that this intermediate semantic language need not be the same as whatever representation the actual back-end uses to store its knowledge. The semantic representation language is suitable for depicting the content of a sentence (or a sequence of sentences), and for extracting a response from the back-end; the back-end's own formalism, used for storing information, might be completely different. This point is sometimes glossed over in systems which use some very general notation (e.g. logic, Prolog) for both purposes.

14.4. Remark

Notice that a system of semantic representation is not just some diagrams or symbolic expressions thrown together in a pleasing or intuitively attractive way. Very simple examples such as representing “John loves Mary” by $love(john, mary)$ can sometimes tempt beginners into thinking that a semantic expression is just the words rearranged, with a bit of punctuation thrown in. This is not a viable approach. A semantic representation formalism should have at the very least the following properties:

1. It should be **clearly defined**. That is, there should be an explicit statement of what would qualify as a legal (well-formed) expression in this notation.
2. Its operations/behaviour should be clearly defined. That is, there should be proper definitions of what **inference** or other manipulations can be carried out on these expressions.
3. To justify the term “semantic”, it should be reasonable to argue that the behaviour of these expressions in some way captures the meaning of the corresponding linguistic expressions.

Anything that lacks these fundamental properties cannot be a serious semantic representation, although it might be handy as an informal notation for sketching out one's thoughts, or a visual aid in conveying ideas to other workers.

15. Semantics vs. Knowledge Representation

To a large extent, the issues involved in semantics are the same as the more general issues of knowledge representation.

But the two endeavours (semantics and knowledge representation) are not wholly identical, since knowledge representation could well be less constrained in what is acceptable as a solution.

- For natural language **semantics**, the formalisations chosen should make those **distinctions which are reflected in the natural language involved**, whereas general knowledge representation need make only those distinctions relevant to inference and other processes.
- The representations used in general **knowledge representation need not be expressible in natural language at all**, but those in semantics have expressibility in language as their entire justification.
- More particularly, NL semantic representations (and their rules) must have a well-defined interface to actual words, phrases and sentences (the concrete syntax), which is not a requirement for more general knowledge representation.

16. Conclusions

Recall: LCT seminar this afternoon at 16:00 (Paolo Dongilli).

Tomorrow, we will look at Discourse Structure.