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.

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.

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.

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.

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.

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**.

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.

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.

whereas we will look at Formal Semantics.

3. Formal Semantics: Main questions

The main questions are:

3. Formal Semantics: Main questions

The main questions are:

1. What does a given sentence mean?

3. Formal Semantics: Main questions

The main questions are:

1. What does a given sentence mean?
2. How is its meaning built?

3. Formal Semantics: Main questions

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?

3. Formal Semantics: Main questions

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.

3. Formal Semantics: Main questions

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**.

4. Logical Approach

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

4. 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.

4. 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.

4. 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**.

4.1. Model: individual constants

The interpretation of a formal language has to include a specification of what constants in the language refers to. This is done by means of the **interpretation function** $\mathcal{I}$ which assigns an appropriate **denotation** in the model $\mathcal{M}$ to each individual and n-place predicate constant .

4.1. Model: individual constants

The interpretation of a formal language has to include a specification of what constants in the language refers to. This is done by means of the **interpretation function** $\mathcal{I}$ which assigns an appropriate **denotation** in the model $\mathcal{M}$ to each individual and n-place predicate constant .

If α is an individual constant, $\mathcal{I}$ maps α onto one of the entities of the universe of discourse $\mathcal{U}$ of the model $\mathcal{M}$: $\mathcal{I}(\alpha) \in \mathcal{U}$.

4.1. Model: individual constants

The interpretation of a formal language has to include a specification of what constants in the language refers to. This is done by means of the **interpretation function** $\mathcal{I}$ which assigns an appropriate **denotation** in the model $\mathcal{M}$ to each individual and n-place predicate constant .

If α is an individual constant, $\mathcal{I}$ maps α onto one of the entities of the universe of discourse $\mathcal{U}$ of the model $\mathcal{M}$: $\mathcal{I}(\alpha) \in \mathcal{U}$.

4.2. Model: one-place predicates

One-place properties are seen as sets of individuals: the property of being orange describes the **set of individuals** that are orange. Formally, for P a one-place predicate, the interpretation function $\mathcal{I}$ maps P onto a subset of the universe of discourse $\mathcal{U} : \mathcal{I}(P) \subseteq \mathcal{U}$. For instance,

4.2. Model: one-place predicates

One-place properties are seen as sets of individuals: the property of being orange describes the **set of individuals** that are orange. Formally, for P a one-place predicate, the interpretation function $\mathcal{I}$ maps P onto a subset of the universe of discourse $\mathcal{U} : \mathcal{I}(P) \subseteq \mathcal{U}$. For instance,

4.3. Model: two-place predicates

Two-place predicates such as “love”, “eat”, “mother-of” do not denote sets of individuals, but **sets of ordered pairs of individuals**, namely all those pairs which stand in the “loving”, “eating”, “mother-of” relations. We form ordered pairs from two sets A and B by taking an element of A as first member of the pair and an element of B as the second member. Given the relation R, the interpretation function $\mathcal{I}$ maps R onto a set of ordered pairs of elements of $\mathcal{U} : \mathcal{I}(R) \subseteq \mathcal{U} \times \mathcal{U}$

4.3. Model: two-place predicates

Two-place predicates such as “love”, “eat”, “mother-of” do not denote sets of individuals, but **sets of ordered pairs of individuals**, namely all those pairs which stand in the “loving”, “eating”, “mother-of” relations. We form ordered pairs from two sets A and B by taking an element of A as first member of the pair and an element of B as the second member. Given the relation R, the interpretation function $\mathcal{I}$ maps R onto a set of ordered pairs of elements of $\mathcal{U} : \mathcal{I}(R) \subseteq \mathcal{U} \times \mathcal{U}$

4.4. Example

Let our model be based on the set of entities $E = \{\texttt{lori}, \texttt{ale}, \texttt{sara}, \texttt{pim}\}$ which represent **Lori, Ale, Sara** and **Pim**, respectively. Assume that they all know themselves, plus **Ale** and **Lori** know each other, but they do not know **Sara** or **Pim**; **Sara** does know **Lori** but not **Ale** or **Pim**. The first three are students whereas **Pim** is a professor, and both **Lori** and **Pim** are tall. This is easily expressed set theoretically. Let $[\![\texttt{w}]\!]$ indicate the interpretation of **w**:

4.4. Example

Let our model be based on the set of entities $E = \{\texttt{lori}, \texttt{ale}, \texttt{sara}, \texttt{pim}\}$ which represent **Lori, Ale, Sara** and **Pim**, respectively. Assume that they all know themselves, plus **Ale** and **Lori** know each other, but they do not know **Sara** or **Pim**; **Sara** does know **Lori** but not **Ale** or **Pim**. The first three are students whereas **Pim** is a professor, and both **Lori** and **Pim** are tall. This is easily expressed set theoretically. Let $[\![\texttt{w}]\!]$ indicate the interpretation of **w**:

$$
\begin{array}{lcl}
[\![\texttt{sara}]\!] & = & \text{sara;} \\
[\![\texttt{pim}]\!] & = & \text{pim;} \\
[\![\texttt{lori}]\!] & = & \text{lori;} \\
[\![\texttt{know}]\!] & = & \{\langle \text{lori, ale} \rangle, \langle \text{ale,lori} \rangle, \langle \text{sara, lori} \rangle, \\
 & & \langle \text{lori, lori} \rangle, \langle \text{ale, ale} \rangle, \langle \text{sara, sara} \rangle, \langle \text{pim, pim} \rangle\}; \\
[\![\texttt{student}]\!] & = & \{\text{lori, ale, sara}\}; \\
[\![\texttt{professor}]\!] & = & \{\text{pim}\}; \\
[\![\texttt{tall}]\!] & = & \{\text{lori, pim}\}.
\end{array}
$$

which is nothing else to say that, for example, the relation **know** is the **set of pairs** $\langle \alpha, \beta \rangle$ where α knows β; or that 'student' is the set of all those elements which are a student.

4.5. Relational and Functional Perspectives

Alternatively, one can assume a functional perspective and interpret, for example, **student** as a function from individual (entities) to truth values, $student(marco) = 1$, $student(raffaella) = 0$.

4.5. Relational and Functional Perspectives

Alternatively, one can assume a functional perspective and interpret, for example, **student** as a function from individual (entities) to truth values, $student(marco) = 1$, $student(raffaella) = 0$.

The shift from the relational to the functional perspective is made possible by the fact that the **sets and their characteristic functions amount to the same thing**:

4.5. Relational and Functional Perspectives

Alternatively, one can assume a functional perspective and interpret, for example, **student** as a function from individual (entities) to truth values, $student(marco) = 1$, $student(raffaella) = 0$.

The shift from the relational to the functional perspective is made possible by the fact that the **sets and their characteristic functions amount to the same thing**:

> if f_X is a function from Y to $\{0, 1\}$, then $X = \{y \mid f_X(y) = 1\}$. In other words, the assertion '$y \in X$' and '$f_X(y) = 1$' are equivalent.

4.5. Relational and Functional Perspectives

Alternatively, one can assume a functional perspective and interpret, for example, **student** as a function from individual (entities) to truth values, $student(marco) = 1$, $student(raffaella) = 0$.

The shift from the relational to the functional perspective is made possible by the fact that the **sets and their characteristic functions amount to the same thing**:

> if f_X is a function from Y to $\{0, 1\}$, then $X = \{y \mid f_X(y) = 1\}$. In other words, the assertion '$y \in X$' and '$f_X(y) = 1$' are equivalent.

Therefore, the two notations $y(z)(u)$ and $y(u, z)$ are equivalent.

4.5. Relational and Functional Perspectives

Alternatively, one can assume a functional perspective and interpret, for example, **student** as a function from individual (entities) to truth values, $student(marco) = 1$, $student(raffaella) = 0$.

The shift from the relational to the functional perspective is made possible by the fact that the **sets and their characteristic functions amount to the same thing**:

> if f_X is a function from Y to $\{0, 1\}$, then $X = \{y \mid f_X(y) = 1\}$. In other words, the assertion '$y \in X$' and '$f_X(y) = 1$' are equivalent.

Therefore, the two notations $y(z)(u)$ and $y(u, z)$ are equivalent.

4.6. Exercises: Relations vs. Functions

Think of which function you can assign to the words in the model considered before and repeated here:

Sara, Pim, Lori, know, student, professor, tall, every man, every Mexican student, no Mexican student, some man.

4.7. Summing up

Summarizing, when trying to formalize natural language semantics, at least two sorts of objects are needed to start with: the set of **truth values** t, and the one of **entities** e.

4.7. Summing up

Summarizing, when trying to formalize natural language semantics, at least two sorts of objects are needed to start with: the set of **truth values** t, and the one of **entities** e.

Moreover, we spoke of more complex objects as well, namely functions. More specifically, we saw that the kind of functions we need are **truth-valued functions** (or boolean functions).

4.7. Summing up

Summarizing, when trying to formalize natural language semantics, at least two sorts of objects are needed to start with: the set of **truth values** t, and the one of **entities** e.

Moreover, we spoke of more complex objects as well, namely functions. More specifically, we saw that the kind of functions we need are **truth-valued functions** (or boolean functions).

Furthermore, we have illustrated how one can move back and forwards between a **set/relational and a functional perspective**. The former can be more handy and intuitive when reasoning about entailment relations among expressions; the latter is more useful when looking for lexicon assignments.

References: Keenen 85.

5. Formal Semantics: What

What does a given sentence mean?

5. 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?”

5. 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.

5. 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 `loves`(`vicent`, `mia`)

5. 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 $\exists x.\texttt{student}(x) \wedge \texttt{loves}(x, \texttt{mia})$.

5. 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.

5. 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.

6. Formal Semantics: How

How is the meaning of a sentence built?

6. 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:

6. 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`

6. 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`

6. 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.

6. 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**.

6.1. Formal Semantics: How (cont'd)

But,

6.1. Formal Semantics: How (cont'd)

But,

1. Why the meaning representation of "Vincent loves Mia" is not `love(mia,vincent)`?

6.1. Formal Semantics: How (cont'd)

But,

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

6.1. Formal Semantics: How (cont'd)

But,

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

As for 1., the missing ingredient is the **syntactic structure**! [Vincent [loves$_v$ Mia$_{np}$]$_{vp}$]$_s$.

6.1. Formal Semantics: How (cont'd)

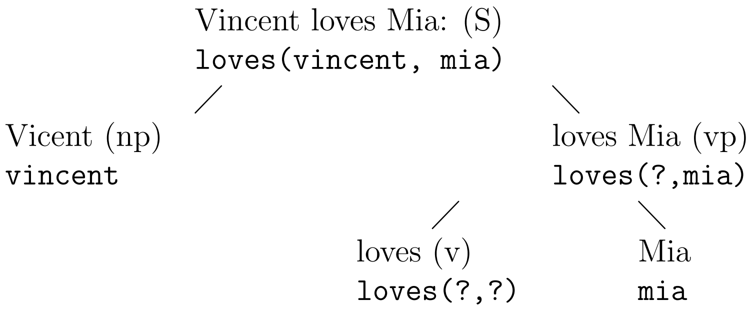
But,

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

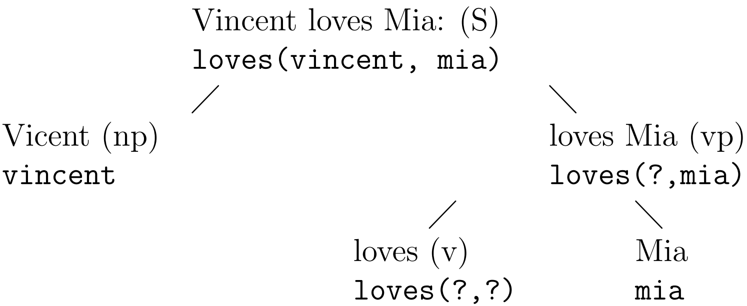
As for 1., the missing ingredient is the **syntactic structure**! [Vincent [loves$_v$ Mia$_{np}$]$_{vp}$]$_s$.

We will come back to 2. next time.

6.2. Formal Semantics: How (Cont'd)



6.2. Formal Semantics: How (Cont'd)



Briefly, **syntactic structure guiding gluing**.

6.3. Compositionality

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

6.3. 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.

6.3. 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.

6.3. 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.

6.4. Ambiguity

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

6.4. Ambiguity

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

For instance, “John saw a man with the telescope”

6.4. 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]$_{pp}$]$_{np}$]$_{vp}$

6.4. 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]$_{pp}$]$_{np}$]$_{vp}$ $\exists x.\texttt{Man}(x) \wedge \texttt{Saw}(j, x) \wedge \texttt{Has}(x, t)$

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

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

Different parse trees result into different meaning representations!

7. How far can we go with FOL?

FOL can capture the **what** (partially) and cannot capture the **how**, i.e.

Problems with the "what":

order

- Swimming is healthy. $Healthy(Swim)$: wrong!
  (property of property)

7. How far can we go with FOL?

FOL can capture the **what** (partially) and cannot capture the **how**, i.e.

Problems with the "what":

order

- Swimming is healthy. $Healthy(Swim)$: wrong!
  (property of property)
- John has all the properties of Santa Clause $\forall P(P(s) \rightarrow P(j))$: wrong!
  (quantification over properties)

7. How far can we go with FOL?

FOL can capture the **what** (partially) and cannot capture the **how**, i.e.

Problems with the "what":

order

- Swimming is healthy. $Healthy(Swim)$: wrong!
  (property of property)
- John has all the properties of Santa Clause $\forall P(P(s) \rightarrow P(j))$: wrong!
  (quantification over properties)
- Red has something in common with green. $\exists P(P(red) \wedge P(green)$: wrong!
  (quant. over properties of properties)

7. How far can we go with FOL?

FOL can capture the **what** (partially) and cannot capture the **how**, i.e.

Problems with the "what":

order
- Swimming is healthy. $Healthy(Swim)$: wrong! (property of property)
- John has all the properties of Santa Clause $\forall P(P(s) \rightarrow P(j))$: wrong! (quantification over properties)
- Red has something in common with green. $\exists P(P(red) \wedge P(green)$: wrong! (quant. over properties of properties)

adj.
- There was a red book on the table. $\exists x(Book(x) \wedge Red(x) \wedge On\_the\_table(x))$.
- There was a small elephant in the zoo. $\exists x(Elephant(x) \wedge Small(x) \wedge In\_the\_zoo(x))$.: wrong!

7. How far can we go with FOL?

FOL can capture the **what** (partially) and cannot capture the **how**, i.e.

Problems with the "what":

order

- Swimming is healthy. $Healthy(Swim)$: wrong! (property of property)
- John has all the properties of Santa Clause $\forall P(P(s) \rightarrow P(j))$: wrong! (quantification over properties)
- Red has something in common with green. $\exists P(P(red) \wedge P(green)$: wrong! (quant. over properties of properties)

adj.

- There was a red book on the table. $\exists x(Book(x) \wedge Red(x) \wedge On\_the\_table(x))$.
- There was a small elephant in the zoo. $\exists x(Elephant(x) \wedge Small(x) \wedge In\_the\_zoo(x))$.: wrong!

adv.

- Milly swam slowly. (modifier of the verb rather than of individuals!)

7. How far can we go with FOL?

FOL can capture the **what** (partially) and cannot capture the **how**, i.e.

Problems with the "what":

order

- Swimming is healthy. $Healthy(Swim)$: wrong! (property of property)
- John has all the properties of Santa Clause $\forall P(P(s) \rightarrow P(j))$: wrong! (quantification over properties)
- Red has something in common with green. $\exists P(P(red) \wedge P(green)$: wrong! (quant. over properties of properties)

adj.

- There was a red book on the table. $\exists x(Book(x) \wedge Red(x) \wedge On\_the\_table(x))$.
- There was a small elephant in the zoo. $\exists x(Elephant(x) \wedge Small(x) \wedge In\_the\_zoo(x))$.: wrong!

adv.

- Milly swam slowly. (modifier of the verb rather than of individuals!)
- Milly swam terribly slow (modifier of a modifier).

7. How far can we go with FOL?

FOL can capture the **what** (partially) and cannot capture the **how**, i.e.

Problems with the "what":

order
- Swimming is healthy. $Healthy(Swim)$: wrong! (property of property)
- John has all the properties of Santa Clause $\forall P(P(s) \rightarrow P(j))$: wrong! (quantification over properties)
- Red has something in common with green. $\exists P(P(red) \wedge P(green)$: wrong! (quant. over properties of properties)

adj.
- There was a red book on the table. $\exists x(Book(x) \wedge Red(x) \wedge On\_the\_table(x))$.
- There was a small elephant in the zoo. $\exists x(Elephant(x) \wedge Small(x) \wedge In\_the\_zoo(x))$.: wrong!

adv.
- Milly swam slowly. (modifier of the verb rather than of individuals!)
- Milly swam terribly slow (modifier of a modifier).

7.1. FOL: How?

Problems with the how:

7.1. FOL: How?

Problems with the how:

Constituents: it cannot capture the meanings of constituents.

7.1. FOL: How?

Problems with the how:

Constituents: it cannot capture the meanings of constituents.

Assembly: it cannot account for meaning representation assembly.

7.1. FOL: How?

Problems with the how:

Constituents: it cannot capture the meanings of constituents.

Assembly: it cannot account for meaning representation assembly.

8. Building Meaning Representations

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

8. 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.

8. 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**.

8. 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.

8. 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.

8. 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 start looking at the other two tasks.

9. Lambda Calculus

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

9. 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.

9.1. Lambda-terms: Examples

Here is an example of lambda terms:

9.1. Lambda-terms: Examples

Here is an example of lambda terms:

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

9.1. Lambda-terms: Examples

Here is an example of lambda terms:

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

The prefix λx. 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.

9.1. 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":

9.1. 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”:

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

9.1. Lambda-terms: Examples

Here is an example of lambda terms:

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

The prefix λx. 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`.

9.1. 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)`.

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

9.3. β-conversion

Summing up:

1. Strip off the λ-prefix,

9.3. β-conversion

Summing up:

1. Strip off the λ-prefix,
2. Remove the argument,

9.3. 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.

9.3. 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. $(\lambda x.love(x, mary))(john)$

9.3. 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. $love(x, mary)(john)$

9.3. 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. $love(x, mary)$

9.4. Exercise

Give the lambda term representing a transitive verb.

9.4. Exercise

Give the lambda term representing a transitive verb.

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

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

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

9.4. 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: $\lambda x.\lambda y.\mathtt{saw}(y, x)$

(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.

9.5. 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.

9.5. 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

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

9.5. 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.

10. 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 ($\mathcal{I}$).

10. 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 were only able to speak of propositions/clauses.

10. 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 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.

10.1. Models, Domains, Interpretation

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

10.1. 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. $D_e = \{\texttt{john}, \texttt{vincent}, \texttt{mary}\}$

10.1. 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 $(a \rightarrow b)$.

10.1. 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).

10.1. 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 $(a \rightarrow b)$.

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

  - denotes in $D_t^{D_e}$

10.1. 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 $(e \to t)$,

10.1. 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 $(a \rightarrow b)$.

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

10.2. 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”.

10.2. 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.

10.2. 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:

10.2. 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.

10.2. 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.

11. The Three Tasks Revised

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

11. 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**

11. 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.**

11. 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 the Lab we have seen that the solution proposed for task 1 leads itself to computational implementation naturally. Next week we will see that this holds for task 2 and 3 too (though we won't go into the detail of it. If you are interested in it and you know Prolog already: good topic for a project!)