

Semantics for Languages (Natural, Formal and Programming)

Informatics 2A: Lecture 21

Bonnie Webber

School of Informatics
University of Edinburgh
bonnie@inf.ed.ac.uk

12 November 2009

- 1 Recipe for a Semantic Definition
 - Syntax and Semantics
 - Compositionality
 - Exceptions
 - Meaning Representations
 - Desiderata for Meaning Representation
- 2 Logical Representations
 - Propositional Logic
 - Predicate Logic
- 3 Commonsense Mapping
- 4 Semantic Composition for NL
 - Compositionality
 - Lambda Expressions

Reading:

*J&M 2nd ed., Chapter 17 (Intro → Section 17.3),
Chapter 18 (Intro → Section 18.1)*

Syntax and Semantics

Semantics is concerned with how expressions in a language map to a world – both their

- **denotation** (literal meaning)
- **connotation** (other associations)

Why consider **semantics** only **after** the lexicon and syntax?

When we say a sentence is **ambiguous**, it is really because it has more than one (literal) meaning.

Some ambiguity comes from words having more than one sense, **some** from sentences having >1 parse tree (syntactic analysis) with respect to a grammar, and **some** from a property called **scope**.

Syntax and Semantics

Computing the meaning of a sentence should reflect both the intended senses of its words and its intended syntactic analysis. We saw this in our canonical example:

I made her duck

- I caused her to drop and avert her head. (*duck* as **action**)
- I created the duck that she owns. (*duck* as **individual**)
- I cooked a/some duck for her. (*duck* as **mass**)

Syntax and Semantics

- Providing a **semantics** for a language (natural, programming or formal) involves giving a **systematic mapping from** the structure underlying a string **to** its meaning.
- While the kinds of meaning conveyed by NLS are generally more complex than those conveyed by PLs, NL and PL semantics share two things in common:
 - **compositionality**
 - **exceptions**

Exceptions: Referential Failure

NLS and PLs share a concept of **referents**, **referring expressions** and **referential failure**.

the Director of Teaching in Informatics
the previous slide

Referential failure occurs when a referring expression fails to have a referent in what the hearer takes to be its intended model.

In such cases, an **exception handling** mechanism is needed:

- (1) To assess a truth value: *The present king of England is bald.*
- (2) To decide what to do: *Turn right at the McDonald Building.*

Compositionality

Compositionality provides a **systematic mapping** between structures of the language and (literal) meaning.

Principle of Compositionality: *The meaning of a complex expression is a function of the meaning of its parts and of the rules by which they are combined.*

While PLs are designed for **compositionality**, it is surprising how often the literal meaning of NL utterances can be derived compositionally as well.

We will look at both strictly compositional and less compositional aspects of the mapping between NL utterances and their literal meaning.

Exceptions: Presupposition Failure

Many sentences we use every day convey additional assumptions about the world, besides the existence of referents for its referring expressions. These are called **presuppositions**.

Unlock the door. \Rightarrow the door is locked.
Do you regret getting drunk Friday? \Rightarrow you got drunk Friday.
John stopped drinking milk. \Rightarrow John used to drink milk.

A **presupposition failure** occurs when a sentence conveys a false assumption about the world.

An exception handling mechanism is needed in order to assess the **truth value** of a sentence containing a false assumption.

- (3) Do you regret getting drunk on Friday?

Meaning Representations

In PL semantics (Lectures 23 & 24), the elements involved in mapping between states of a machine can be as simple as numbers or sets, or more complex, as in **XPath** (the XML Path language) used to specify nodes and their relationships in an XML document.

In NL semantics, the elements of meanings are more complex:

- things in the world;
- actions in the world involving those things;
- abstract mental constructs: democracy, status, agreement, etc.

These are not things we can manipulate or calculate on themselves: Instead, we work with **representations** of meaning.

Unambiguous

2. A meaning representation should be **unambiguous**, with one and only one interpretation. If a sentence is ambiguous, there should be a different meaning representation for each sense.

(5) I made her duck.

Each interpretation of (5) should have a distinct representation.

Verifiability

A meaning representation requires four properties for effective use:

1. It should be **verifiable**: One must be able to use the meaning representation of a sentence to determine the relationship between its meaning and a model of the world.

(4) "One must be able to use the meaning representation of a sentence to determine the relationship between its meaning and a model of the world."

The representation of (4) should account (in part) for what you take (4) to mean.

Verifiability implies that we can assess the truth with respect to a world of the meaning representation assigned to a sentence.

Canonical Form

3. A meaning representation should have a **canonical form**: The meaning representations for sentences with the same meaning should both be convertible into the same canonical form, that shows their equivalence.

[I filled the room with balloons] \equiv [I put enough balloons in the room to fill it from floor to ceiling]

Relationships other than identity should be derivable by **entailment** and other forms of **inference**.

[I filled the room with balloons] \Rightarrow [I put at least one balloon in the room]

Expressivity: Predicate-Argument Structure

4. A meaning representation should be **expressive**, allowing a wide range of meanings to be expressed in a natural and revealing way, including **relationships** between the words in a sentence – eg,

- **Restrictions** on the concept denoted by the head of a phrase.
 - *brown cow* (**How is brown related to cow?**)
 - *man who came to dinner* (**or man related to came to dinner?**)
 - *walk briskly* (**or walk related to briskly?**)
- **Predicate-argument relations** specifying the participants in an event or situation associated with the head of a phrase:
 - *Fred eats lentils* (NP V NP): an *eating* event, with Fred doing the eating (*agent*), and lentils being eaten (*theme*);
 - *Fred eats lentils with a fork* (NP V NP with NP): the same, but with a fork as the *instrument* used for eating the lentils.

Propositional Logic

Propositional logic is one system for representation and reasoning in which expressions comprise:

- **atomic sentences** (P, Q, etc.);
- **complex sentences** built up from atomic sentences and logical connectives (and, or, not).

Inf1 notation: not $P \rightarrow (Q \text{ or } R)$

J&M notation: $\neg P \Rightarrow (Q \vee R)$

Propositional Logic

Question: Why not use propositional logic as a meaning representation system for NL?

Fred ate lentils or he ate rice. ($P \vee Q$)
 Fred ate lentils or John ate lentils ($P \vee R$)

We lose any obvious relationship between the clauses that make up these sentences.

Everyone ate lentils. ($P1 \wedge P2 \wedge P3 \wedge P4 \dots$)
 Someone ate lentils. ($P1 \vee P2 \vee P3 \vee P4 \dots$)

We can't really express either sentence.

Predicate Logic

Inf1 also introduced **first-order predicate logic** (FOPL), which is closer to being expressive enough for NL semantics.

Sentences in FOPL are built up from **terms** made from:

- **constant and variable symbols** that represent entities;
- **function symbols** that allow us to indirectly specify entities;
- **predicate symbols** that represent properties of entities and relations that hold between entities;

which are combined into simple sentences (**predicate-argument** structures) and complex sentences through:

quantifiers (\forall, \exists) disjunction (\vee)
 negation (\neg) implication (\Rightarrow)
 conjunction (\wedge) equality ($=$)

Constants

Constant symbols: Scotland, Perth, EU, John:

- Each constant symbol denotes one and only one entity.

George W. Bush, Scotland, 2007

- Not all entities have a constant symbol that denotes them.

George W. Bush's right knee, this pen

- Several constant symbols may denote the same entity.

The Morning Star \equiv The Evening Star \equiv Venus
 National Insurance number, Student ID, your name

Variables

Variable symbols: x, y, z:

- Variable symbols range over entities.
- An atomic sentence with a variable among its arguments, e.g., Part_of(x, EU), only has a truth value if that variable is **bound** by a quantifier.

Predicates

Predicate symbols:

- Every predicate has a specific arity: Brown/1, Country/1, Live_in/2, Give/3.
- Each predicate symbol of arity N is interpreted as a set of N -tuples of entities that satisfy it.
- Predicates of arity 1 denote properties.
- Predicates of arity > 1 denote relations.

Universal Quantifier (\forall)

Universal quantifiers can be used to express **general truths**:

Cats are mammals is a **general fact** that any entity having the property of being a cat also has the property of being a mammal.

$\forall x. \text{Cat}(x) \Rightarrow \text{Mammal}(x)$

Universally quantified sentence corresponds to a **conjunction** of sentences in which a constant **substitutes** for a variable.

$\text{Cat}(\text{sam}) \Rightarrow \text{Mammal}(\text{sam}) \wedge \text{Cat}(\text{zoot}) \Rightarrow \text{Mammal}(\text{zoot})$
 $\wedge \text{Cat}(\text{fritz}) \Rightarrow \text{Mammal}(\text{fritz}) \wedge \dots$

A quantifier has a **scope**, defined as what depends on it.

Existential Quantifier (\exists)

Existential quantifier can be used to express **indefinite truths** that a property/relation holds of some entity, without specifying which one, e.g., *I have a cat*:

$$\exists x. \text{Cat}(x) \wedge \text{Own}(i, x)$$

An existentially quantified sentence corresponds to **disjunction** of sentences in which a constant **substitutes** for a variable.

$$\begin{aligned} &(\text{Cat}(\text{Josephine}) \wedge \text{Own}(i, \text{Josephine})) \vee \\ &(\text{Cat}(\text{Zoot}) \wedge \text{Own}(i, \text{Zoot})) \vee \\ &(\text{Cat}(\text{Malcolm}) \wedge \text{Own}(i, \text{Malcolm})) \vee \\ &(\text{Cat}(\text{John}) \wedge \text{Own}(i, \text{John})) \vee \dots \end{aligned}$$

Existential Quantifier (\exists)

Why do we use “ \wedge ” rather than “ \Rightarrow ” with the existential quantifier? What would the following proposition correspond to?

$$\exists x. \text{Cat}(x) \Rightarrow \text{Own}(i, x)$$

(a) *I own a cat*

(b) *There is something that if it's a cat, I own it*

What if that something is not a cat?

The proposition formed by connecting two propositions with \Rightarrow is true if the antecedent (the left of the \Rightarrow) is false.

So this proposition is true if there is something that's a laptop, for example: “I own a cat” shouldn't be true simply for this reason.

Commonsense Mapping

We can easily map simple NL statements into FOPL:

Fred showed “Born to Run” to Mary.
Fred showed Mary “Born to Run”.
Show(Fred, Born_to_Run, Mary)

We need to specify what position corresponds to what argument (*agent* = Fred, *object* = Born_to_Run, *beneficiary* = Mary): Differences in syntax shouldn't lead to differences in representation.

We can also easily map NL statements involving a single existential:

Fred sang Mary a song.
Fred sang a song to Mary.
 $\exists x. \text{Song}(x) \wedge \text{Sing}(\text{Fred}, x, \text{Mary})$

Commonsense Mapping

Also easy to map are NL statements that involve a single universal, provided one recognizes that general rules can look like existentials:

Cats are mammals.
A cat is a mammal.
 $\forall x. \text{Cat}(x) \Rightarrow \text{Mammal}(x)$

Fred sang to Mary each song in “Born to Run”.
 $\forall x. \text{Song}(x) \wedge \text{In}(x, \text{Born_to_Run}) \Rightarrow \text{Sing}(\text{Fred}, x, \text{Mary})$

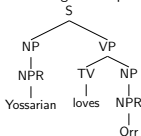
With more than one quantifier, issues of **scope** arise, which we'll discuss in Lecture 22.

Compositionality

Now we return to the **Principle of compositionality**:

The meaning of a complex expression is a function of the meaning of its parts and of the rules by which they are combined.

Do we have sufficient tools to **systematically compute** meaning representations according to this principle?



o If the interpretation of **loves** is represented as the binary predicate $\text{love}(x,y)$ and that of **Orr**, as orr , how do we combine them to produce an interpretation for the VP **loves Orr**?

o To compute NL interpretations compositionally, we need **lambda expressions** (λ -expressions)

Lambda (λ) Expressions

o λ -expressions are a notational extension to FOPL used to construct new **unary predicates**. A λ -expression consists of:

- the Greek letter λ ;
- a variable that it binds as its formal parameter;
- a FOPL expression that uses that variable

Example

$\lambda x. \text{sleep}(x)$

o A λ -expression can be **applied** to a **term**:

$\lambda x. \text{sleep}(x)$ (orr) has the same truth value as $\text{sleep}(\text{orr})$
functor argument

o Lambda expressions can be used to express a **binary predicate** as a pair of **unary predicates**:

$\text{love}(x,y)$

Unary predicate: $\lambda y. \text{love}(x, y)$

Unary predicate: $\lambda y. \lambda x. \text{love}(x, y)$

or a **ternary predicate** as three (nested) **unary predicates**.

$\text{give}(x,y,z)$

Unary predicate: $\lambda z. \text{give}(x, y, z)$

Unary predicate: $\lambda z. \lambda y. \text{give}(x, y, z)$

Unary predicate: $\lambda z. \lambda y. \lambda x. \text{give}(x, y, z)$

Beta Reduction

When a lambda expression **applies** to a term, a reduction operation (**beta (β) reduction**) can be used to replace its formal parameter with the term and simplify the result.

$$\underbrace{\lambda x.sleep(x)}_{\text{functor}} \underbrace{(orr)}_{\text{argument}} \text{ simplifies to } \Rightarrow_{\beta} \text{sleep}(orr)$$

$$\underbrace{\lambda y.\lambda x.love(x, y)}_{\text{functor}} \underbrace{(crabapples)}_{\text{argument}} \Rightarrow_{\beta} \lambda x.love(x, crabapples)$$

$$\underbrace{\lambda x.love(x, crabapples)}_{\text{functor}} \underbrace{(orr)}_{\text{argument}} \Rightarrow_{\beta} \text{love}(orr, crabapples)$$

λ -expressions can also **apply** to other λ -expressions:

$$\underbrace{\lambda P.P(orr)}_{\text{functor}} \underbrace{(\lambda x.sleep(x))}_{\text{argument}} \Rightarrow_{\beta} \underbrace{(\lambda x.sleep(x))}_{\text{functor}} \underbrace{(orr)}_{\text{argument}} \Rightarrow_{\beta} \text{sleep}(orr)$$

Implied bracketting

λ expressions are left-associative:

$$\lambda x.expression(arg1, arg2) = ((\lambda x.expression)arg1)arg2$$

$$\underbrace{\lambda y.\lambda x.love(x, y)}_{\text{functor}} \underbrace{(orr, crabapples)}_{\text{arguments}} \Rightarrow_{\beta} \underbrace{\lambda x.love(x, orr)}_{\text{functor}} \underbrace{(crabapples)}_{\text{argument}} \Rightarrow_{\beta} \text{love}(crabapples, orr)$$

$$\underbrace{\lambda x.\lambda y.love(x, y)}_{\text{functor}} \underbrace{(orr, crabapples)}_{\text{arguments}} \Rightarrow_{\beta} \underbrace{\lambda y.love(orr, y)}_{\text{functor}} \underbrace{(crabapples)}_{\text{argument}} \Rightarrow_{\beta} \text{love}(orr, crabapples)$$

A problem

We have applied λ -expressions to terms and to other λ -expressions. Can we apply a λ -expression to a variable that already appears in the functor?

Example

$$\underbrace{\lambda x.\lambda y.love(x, y)}_{\text{functor}} \underbrace{(y)}_{\text{argument}}$$

No! This expression should **not** simplify to

$$\Rightarrow_{\beta} \lambda y.love(y, y)$$

because we have **accidentally bound** the unbound variable y .

We must rename the variable in the functor to avoid this.

Alpha Conversion

Alpha conversion (α -conversion) involves renaming functor bound variables so that they differ from the variables in the argument.

This operation is applied before beta-conversion.

Example

$$\lambda x.\lambda y.\textit{love}(x, y)(y) \Rightarrow_{\alpha} \lambda x.\lambda z.\textit{love}(x, z)(y) \Rightarrow_{\beta} \lambda z.\textit{love}(y, z)$$

Tomorrow we look at automatically computing these representations.