Agreement, Types and Natural Language Semantics

Informatics 2A: Lecture 25

Shay Cohen

13 November 2017

Last Class

- How do we extract a grammar from a treebank?
- How do we estimate the rule probabilities of a grammar?
- How can we overcome the limitations of plain PCFGs? (more on that today)

Agreement phenomena

In PLs, typing rules enforce type agreement between different (often separated) constituents of a program:

```
int i=0; ...; if (i>2) ...
```

There are somewhat similar phenomena in NL: constituents of a sentence (often separated) may be constrained to agree on an attribute such as person, number, gender.

- You, I imagine, are unable to attend.
- The hills are looking lovely today, aren't they?
- He came very close to injuring himself.

Agreement in various languages

These examples illustrate that in English:

- Verbs agree in person and number with their subjects;
- Tag questions agree in person, number, tense and mode with their main statement, and have the opposite polarity.
- Reflexive pronouns follow suit in person, number and *gender*.

French has much more by way of agreement phenomena:

- Adjectives agree with their head noun in gender and number.

  Le petit chien,   La petite souris,   Les petites mouches

- Participles of *être* verbs agree with their subject:

  Il est arrivé,   Elles sont arrivées

- Participles of other verbs agree with preceding direct objects:

  Il a vu la femme,   Il l'a vue

How can we capture these kinds of constraints in a grammar?

Agreement rules: why bother?

Modelling agreement is obviously important if we want to generate grammatically correct NL text.

But even for understanding input text, agreement can be useful for resolving ambiguity.
E.g. the following sentence is ambiguous . . .

The boy who eats flies ducks.

. . . whilst the following are less so:

The boys who eat fly ducks.
The boys who eat flies duck.

Node-splitting via attributes

One solution is to refine our grammar by splitting certain non-terminals according to various *attributes*. Examples of attributes and their associated values are:

- Person: 1st, 2nd, 3rd
- Number: singular, plural
- Gender: masculine, feminine, neuter
- Case: nominative, accusative, dative, . . .
- Tense: present, past, future, . . .

In principle these are language-specific, though certain common patterns recur in many languages.

We can then split phrase categories as the language demands, e.g.

- Split NP on person, number, case (e.g. NP[3,sg,nom]),
- Split VP on person, number, tense (e.g. VP[3,sg,fut]).

Parameterized CFG productions

We can often use such attributes to enforce agreement constraints. This works because of the head phrase structure typical of NLs. E.g. we may write parameterized rules such as:

$$\begin{array}{rcl} \text{S} & \rightarrow & \text{NP[p,n,nom] VP[p,n]} \\ \text{NP[3,n,c]} & \rightarrow & \text{Det[n] Nom[n]} \end{array}$$

Each of these really abbreviates a finite number of rules obtained by specializing the attribute variables. (Still a CFG!)
When specializing, each variable must take the same value everywhere, e.g.

$$\begin{array}{rcl} \text{S} & \rightarrow & \text{NP[3,sg,nom] VP[3,sg]} \\ \text{S} & \rightarrow & \text{NP[1,pl,nom] VP[1,pl]} \\ \text{NP[3,pl,acc]} & \rightarrow & \text{Det[pl] Nom[pl]} \end{array}$$

Parsing algorithms can be adapted to work with this machinery: don't have to 'build' all the specialized rules individually.

Example: subject-verb agreement in English

$$\begin{array}{rcl}
\text{S} & \rightarrow & \text{NP[p,n,nom] VP[p,n]} \\
\text{NP[p,n,c]} & \rightarrow & \text{Pro[p,n,c]} \\
\text{Pro[1,sg,nom]} & \rightarrow & \textit{I}\text{, etc.} \\
\text{Pro[1,sg,acc]} & \rightarrow & \textit{me}\text{, etc.} \\
\text{NP[3,n,c]} & \rightarrow & \text{Det[n] Nom[n] RelOpt[n]} \\
\text{Nom[n]} & \rightarrow & \text{N[n]} \mid \text{Adj Nom[n]} \\
\text{N[sg]} & \rightarrow & \textit{person}\text{, etc.} \\
\text{N[pl]} & \rightarrow & \textit{people}\text{, etc.} \\
\text{RelOpt[n]} & \rightarrow & \epsilon \mid \textit{who}\text{ VP[3,n]} \\
\text{VP[p,n]} & \rightarrow & \text{VV[p,n] NP[p',n',acc]} \\
\text{VV[p,n]} & \rightarrow & \text{V[p,n]} \mid \text{BE[p,n] VG} \\
\text{V[3,sg]} & \rightarrow & \textit{teaches}\text{, etc.} \\
\text{BE[p,n]} & \rightarrow & \textit{is}\text{, etc.} \\
\text{VG} & \rightarrow & \textit{teaching}\text{, etc.}
\end{array}$$

(Other rules omitted.)

Semantics

- We mentioned in previous lectures that grammars need to be **revealing**
- We also mentioned that language is **compositional**: the meaning of a sentence is constructed from the meaning of its constituents
- Is syntax enough to represent the meaning?
- Want to be able to map from syntax to a logical statement (semantics)

Syntax and Semantics

Syntax is concerned with which expressions in a language are well-formed or grammatically correct. This can largely be described by rules that make no reference to *meaning*.
Semantics is concerned with the meaning of expressions: i.e. how they relate to 'the world'. This includes both their

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

When we say a sentence is ambiguous, we usually mean it has more than one 'meaning'. (So what exactly are meanings?)

We've already encountered word sense ambiguity and structural ambiguity. We'll also meet another kind of semantic ambiguity, called scope ambiguity. (This already shows that the meaning of a sentence can't be equated with its parse tree.)

Formal and natural language semantics

Providing a semantics for a language (natural or formal) involves giving a systematic mapping from the structure underlying a string (e.g. syntax tree) to its ‘meaning’.

Whilst the kinds of meaning conveyed by NL are much more complex than those conveyed by FLs, they both broadly adhere to a principle called compositionality.

Compositionality

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 formal languages are designed for compositionality, the meaning of NL utterances can often (not always) be derived compositionally as well.

Compare:

purple armadillo　　　hot dog

Other desiderata for Meaning Representation

Verifiability: One must be able to use the meaning representation of a sentence to determine whether the sentence is true with respect to some given model of the world.

Example: given an exhaustive table of ‘who loves whom’ relations (a world model), the meaning of a sentence like *everybody loves Mary* can be established by checking it against this model.

Desiderata for Meaning Representation

Unambiguity: 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.

Example: each interpretation of *I made her duck* or *time flies like an arrow* should have a distinct meaning representation.

Desiderata for Meaning Representation

Canonical form: the meaning representations for sentences with the same meaning should (ideally) both be convertible into the same canonical form, that shows their equivalence.

Example: the sentence *I filled the room with balloons* should ideally have the same canonical form with *I put enough balloons in the room to fill it from floor to ceiling*.

(The kind of formal semantics we discuss won't achieve this particularly well!)

Desiderata for Meaning Representation

Logical inference: A good meaning representation should come with a set of rules for logical inference or deduction, showing which truths imply which other truths.

E.g. from

Zoot is an armadillo.
Zoot is purple.
Every purple armadillo sneezes.

we should be able to deduce

Zoot sneezes.

Propositional Logic

Propositional logic is a very simple system for meaning 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, implies).

Propositional Logic

Why not use propositional logic as a meaning representation system for NL? E.g.

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

- We're unable to represent the internal structure of the proposition 'Fred ate lentils' (e.g. how its meaning is derived from that of 'Fred', 'ate', 'lentils').
- We're unable to express e.g.

Everyone ate lentils.
Someone ate lentils.

Predicate Logic

First-order predicate logic (FOPL) let us do a lot more (though still only accounts for a tiny part of NL).

Sentences in FOPL are built up from terms made from:

- constant and variable symbols that represent entities;
- predicate symbols that represent properties of entities and relations that hold between entities;
- function symbols (won't bother with these here).

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 ($=$)

Predicate Logic Example

Constants

Constant symbols:

- **Each constant symbol denotes one and only one entity:** Scotland, Aviemore, EU, Barack Obama, 2007
- **Not all entities have a constant that denotes them:** Barack Obama's right knee, this piece of chalk
- **Several constant symbols may denote the same entity:** The Morning Star $\equiv$ The Evening Star $\equiv$ Venus
  National Insurance number, Student ID, your name

Predicates

Predicate symbols:

- Every predicate has a specific arity. E.g. brown/1, country/1, live_in/2, give/3.
- A predicate symbol of arity n is interpreted as a set of n-tuples of entities that satisfy it.
- Predicates of arity 1 denote properties: brown/1.
- Predicates of arity > 1 denote relations: live_in/2, give/3.

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.

Universal Quantifier ($\forall$)

Universal quantifiers can be used to express general truths:

- *Cats are mammals*
- $\forall$x.Cat(x) $\Rightarrow$ Mammal(x)

Intuitively, a universally quantified sentence corresponds to a (possibly infinite) conjunction of sentences:

Cat(sam) $\Rightarrow$ Mammal(sam) $\wedge$ Cat(zoot) $\Rightarrow$ Mammal(zoot) $\wedge$ Cat(fritz) $\Rightarrow$ Mammal(fritz) $\wedge \ldots$

A quantifier has a scope, analogous to scope of PL variables.

Existential Quantifier ($\exists$)

Existential quantifiers are used to express the existence of an entity with a given property, without specifying which entity:

- *I have a cat*
- $\exists$x.Cat(x) $\wedge$ Own(i, x)

An existentially quantified sentence corresponds intuitively to a disjunction of sentences:

(Cat(Josephine) $\wedge$ Own(I, Josephine)) $\vee$
(Cat(Zoot) $\wedge$ Own(I, Zoot)) $\vee$
(Cat(Malcolm) $\wedge$ Own(I, Malcolm)) $\vee$
(Cat(John) $\wedge$ Own(I, John)) $\vee$ . . .

Existential Quantifier ($\exists$)

Why do we use "$\wedge$" rather than "$\Rightarrow$" with the existential quantifier? What would the following correspond to?

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

(a) I own a cat

(b) There's something such that if it's a cat, I own it.

What if that something isn't 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 e.g. a laptop. But "I own a cat" shouldn't be true simply for this reason.

Abstract syntax of FOPL

The language of first-order predicate logic can be defined by the following CFG (think of it as a grammar for abstract syntax trees). We write F for formulae, AF for atomic formulae, t for terms, v for variables, c for constants.

$$\begin{array}{rcl} \mathsf{F} & \rightarrow & \mathsf{AF} \mid \mathsf{F} \wedge \mathsf{F} \mid \mathsf{F} \vee \mathsf{F} \mid \mathsf{F} \Rightarrow \mathsf{F} \mid \neg\, \mathsf{F} \\ & & \mid \forall\, \mathsf{v.F} \mid \exists\, \mathsf{v.F} \\ \mathsf{AF} & \rightarrow & \mathsf{t{=}t} \mid \mathsf{UnaryPred(t)} \mid \mathsf{BinaryPred(t,t)} \mid \ldots \\ \mathsf{t} & \rightarrow & \mathsf{v} \mid \mathsf{c} \end{array}$$

Question

Which captures the meaning of *Every dog has a bone*?

1. $\forall x.\exists y.(\text{dog}(x) \wedge \text{bone}(y) \wedge \text{has}(x,y))$
2. $\forall x.(\text{dog}(x) \Rightarrow \exists y.(\text{bone}(y) \wedge \text{has}(x,y)))$
3. $\forall x.\exists y.\text{bone}(y) \wedge (\text{dog}(x) \Rightarrow \text{has}(x,y))$
4. $\exists y.\forall x.(\text{dog}(x) \Rightarrow (\text{bone}(y) \wedge \text{has}(x,y)))$

(N.B. The logical form looks structurally quite different from the parse tree for the original sentence. So there's some real work to be done!)

Types in Natural Language Semantics

Types are very useful if we wish to describe the semantics (i.e., meaning) of natural languages. For example, we can use types employed in logic to model the meanings of various phrase types.

Basic Types

1. e — the type of real-world *entities* such as Inf2a, Stuart, John.
2. t — the type of *facts with truth value* like ‘Inf2a is amusing’.

From these two basic types, we may construct more complex types via the function type constructor.

From basic to complex formal types

We use the notation $< \sigma, \tau >$ to denote functions of the form $\sigma \rightarrow \tau$. E.g.:

- *<e,t>*: **unary predicates** – functions from entities to facts.
- *<e, <e,t>>*: **binary predicates** – functions from entities to unary predicates.
- *<<e,t>, t>*: **type-raised entities** – functions from unary predicates to truth values.

- Inf2a, Stuart **:** *e*
- enjoys **:** *<e, <e,t>>*
- enjoys Inf2a, is amusing **:** *<e,t>*
- Inf2a is amusing, Stuart enjoys Inf2a **:** *t*
- every student **:** *<<e,t>, t>*

This simple system of types will be enough to start with. But for more precise semantic modelling, a much richer type system is desirable.

Compositionality

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?

- The meaning of a complete sentence will hopefully be a FOPL formula, which we consider as having type t (truth values).
- But the meaning of smaller fragments of the sentence will have other types. E.g.

$$\begin{array}{ll} \textit{has a bone} & <e,t> \\ \textit{every dog} & <<e,t>,t> \end{array}$$

- The idea is to show how to associate a meaning with such fragments, and how these meanings combine.
- To do this, we need to extend our language of FOPL with λ expressions (λ = lambda; written as \ in Haskell).

Lambda (λ) Expressions

λ-**expressions** are an extension to FOPL that allows us to work with 'partially constructed' formulae. A λ-expression consists of:

- the Greek letter λ, followed by a variable (formal parameter);
- a FOPL expression that may involve that variable.

$\lambda x.sleep(x) \quad : \quad < e, t >$
'The function that takes an entity x to the statement sleep(x)'

$$\underbrace{(\lambda x.sleep(x))}_{function}\underbrace{(Mary)}_{argument} \quad : \quad t$$

A λ-expression can be applied to a term.
(The above has the same truth value as $sleep(Mary)$.)

Lambda expressions can be nested. We can use nesting to create functions of several arguments that accept their arguments one at a time.

$\lambda y.\lambda x.$ love(x,y) : $< e, < e, t >>$
'The function that takes y to
(the function that takes x to the statement love(x,y))'

$\lambda z.\lambda y.\lambda x.$ give(x,y,z) : $< e, < e, < e, t >>>$
'The function that takes z to
(the function that takes y to
(the function that takes x to the statement 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. In general:

$$(\lambda x.M)N \ \Rightarrow_\beta \ M[x \mapsto N] \quad (M \text{ with } N \text{ substituted for } x)$$

$$\underbrace{(\lambda x.sleep(x))}_{function}\underbrace{(Mary)}_{argument} \Rightarrow_\beta sleep(Mary)$$

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

$$\underbrace{(\lambda x.love(x,crabapples))}_{function}\underbrace{(Mary)}_{argument} \Rightarrow_\beta love(Mary, crabapples)$$

Summary

- Agreement in language can be modeled in a grammar by splitting the nonterminals.
- First-order predicate logic can be used to model the meaning of language.
- Each constituent in a sentence is given a type, such as an "entity" or a "statement."
- The types can become quite complex and need to make use of λ-expressions to retain compositionality

Next Lecture: more about computing semantics using syntax