ANLP Lecture 23
Syntax/Semantics interface
(Semantic analysis)

Sharon Goldwater
(based on slides by James Martin and Johanna Moore)

6 November 2014
Today

• Closer look at meaning representations

• Syntax-Driven Semantic Analysis
Verbal (event) MRs: the story so far

Syntax:

\[ \text{NP} \text{ give } \text{NP}_1 \text{ NP}_2 \]

Semantics:

\[ \lambda z. \lambda y. \lambda x. \text{Giving}_1(x,y,z) \]

Applied to arguments:

\[ \lambda z. \lambda y. \lambda x. \text{Giving}_1(x,y,z) (\text{book})(\text{Mary})(\text{John}) \]

As in the sentence:

John gave Mary a book.

\[ \text{Giving}_1(\text{John}, \text{Mary}, \text{book}) \]
But what about these?

John gave Mary a book for Susan.
\( \text{Giving}_2(\text{John}, \text{Mary}, \text{Book}, \text{Susan}) \)

John gave Mary a book for Susan on Wednesday.
\( \text{Giving}_3(\text{John}, \text{Mary}, \text{Book}, \text{Susan}, \text{Wednesday}) \)

John gave Mary a book for Susan on Wednesday in class.
\( \text{Giving}_4(\text{John}, \text{Mary}, \text{Book}, \text{Susan}, \text{Wednesday}, \text{InClass}) \)

John gave Mary a book with trepidation.
\( \text{Giving}_5(\text{John}, \text{Mary}, \text{Book}, \text{Susan}, \text{Trepidation}) \)
Problem with event representations

- Predicates in First-order Logic have fixed arity

- Requires separate *Giving* predicate for each syntactic *subcategorisation frame* (number/type/position of arguments).

- Separate predicates have no logical relation, but they ought to.
  
  - Ex. if \( \text{Giving}_3(a, b, c, d, e) \) is true, then so are \( \text{Giving}_2(a, b, c, d) \) and \( \text{Giving}_1(a, b, c) \).

- See J&M for various unsuccessful ways to solve this problem; we’ll go straight to a more useful way.
Reification of events

• We can solve these problems by reifying events.
  – Reify: to “make real” or concrete, i.e., give events the same status as entities.
  – In practice, introduce variables for events, which we can quantify over.
Reification of events

• We can solve these problems by **reifying** events.
  – Reify: to “make real” or concrete, i.e., give events the same status as entities.
  – In practice, introduce variables for events, which we can quantify over.

• MR for **John gave Mary a book** is now

\[ \exists e, z. \text{Giving}(e) \land \text{Giver}(e, \text{John}) \land \text{Givee}(e, \text{Mary}) \land \text{Given}(e,z) \land \text{Book}(z) \]

• The giving event is now a single predicate of arity 1: **Giving(e)**; remaining conjuncts represent the participants (semantic roles).
Entailment relations

- Our new representation automatically gives us logical entailment relations between events.

- John gave Mary a book on Tuesday entails John gave Mary a book.
Entailment relations

• Our new representation automatically gives us logical entailment relations between events.

• John gave Mary a book on Tuesday entails John gave Mary a book. Similarly,

\[ \exists e, z. \ Giving(e) \land Giver(e, John) \land Givee(e, Mary) \land Given(e, z) \land Book(z) \land Time(e, Tuesday) \]

entails

\[ \exists e, z. \ Giving(e) \land Giver(e, John) \land Givee(e, Mary) \land Given(e, z) \land Book(z) \]

• Can add as many semantic roles as needed for the event.
At last: Semantic Analysis

- Given this way of representing meanings, how do we compute meaning representations from sentences?

- The task of **semantic analysis** or **semantic parsing**.

- Most methods rely on a (prior or concurrent) syntactic parse.

- Here: a compositional **rule-to-rule** approach based on FOL augmented with λ-expressions.
Syntax Driven Semantic Analysis

• Based on the **principle of compositionality**.
  – meaning of the whole built up from the meaning of the parts
  – more specifically, in a way that is guided by word ordering and syntactic constituents/relations.

• Build up the MR by augmenting CFG rules with semantic composition rules.

• Representation produced is *literal meaning*: context independent and free of inference

Note: other syntax-driven semantic parsing formalisms exist, e.g. Combinatory Categorial Grammar (Steedman, 2000) has seen a surge in popularity recently.
Example of final analysis

- What we’re hoping to build
CFG Rules with Semantic Attachments

• To compute the final MR, we add **semantic attachments** to our CFG rules.

• These specify how to compute the MR of the parent from those of its children.

• Rules will look like:

\[ A \rightarrow \alpha_1 \ldots \alpha_n \begin{cases} f(\alpha_j.sem, \ldots, \alpha_k.sem) \end{cases} \]

• \( A.sem \) (the MR for \( A \)) is computed by applying the function \( f \) to the MRs of some subset of \( A \)'s children.
Proposed rules

- Ex: **AyCaramba serves meat** (with parse tree)

- Rules with semantic attachments for nouns and NPs:

  \[
  \begin{align*}
  \text{ProperNoun} & \rightarrow \text{AyCaramba} & \{\text{AyCaramba}\} \\
  \text{MassNoun} & \rightarrow \text{meat} & \{\text{Meat}\} \\
  \text{NP} & \rightarrow \text{ProperNoun} & \{\text{ProperNoun.sem}\} \\
  \text{NP} & \rightarrow \text{MassNoun} & \{\text{MassNoun.sem}\}
  \end{align*}
  \]

- Unary rules normally just copy the semantics of the child to the parents (as in NP rules here).
What about verbs?

- Before event reification, we had verbs with meanings like:

  \[ \lambda y. \lambda x. Serving(x, y) \]

- \( \lambda s \) allowed us to compose arguments with predicate.

- We can do the same with reified events:

  \[ \lambda y. \lambda x. \exists e. Serving(e) \land Server(e, x) \land Served(e, y) \]
What about verbs?

• Before event reification, we had verbs with meanings like:

\[ \lambda y. \lambda x. \text{Serving}(x,y) \]

• \( \lambda \)s allowed us to compose arguments with predicate.

• We can do the same with reified events:

\[ \lambda y. \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, y) \]

• This MR is the semantic attachment of the verb:

\[
\text{Verb} \rightarrow \text{serves} \\
\{ \lambda y. \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, y) \} \]
Building larger constituents

• The remaining rules specify how to apply $\lambda$-expressions to their arguments. So, VP rule is:

$$\text{VP} \rightarrow \text{Verb} \text{ NP} \quad \{\text{Verb.sem(NP.sem)}\}$$
The remaining rules specify how to apply $\lambda$-expressions to their arguments. So, VP rule is:

$$\text{VP} \rightarrow \text{Verb} \hspace{1em} \text{NP} \hspace{1em} \{\text{Verb.sem(NP.sem)}\}$$

where $\text{Verb.sem} = \lambda y. \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, y)$

and $\text{NP.sem} = \text{Meat}$
Building larger constituents

• The remaining rules specify how to apply \(\lambda\)-expressions to their arguments. So, VP rule is:

\[
\text{VP} \rightarrow \text{Verb} \ \text{NP} \quad \{\text{Verb.sem(NP.sem)}\}
\]

\[
\begin{array}{c}
\text{Verb} \\
\text{serves} \\
\text{meat}
\end{array}
\]

\[
\begin{array}{c}
\text{NP} \\
\text{Mass-Noun}
\end{array}
\]

where  

\[
\text{Verb.sem} = \lambda y. \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, y)
\]

and  

\[
\text{NP.sem} = \text{Meat}
\]

• So,  

\[
\text{VP.sem} = \lambda y. \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, y) \ (\text{Meat}) = \\
\lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, \text{Meat})
\]
Finishing the analysis

• Final rule is:

\[ S \rightarrow NP \quad VP \quad \{ VP.sem(NP.sem) \} \]

• now with \( VP.sem = \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, \text{Meat}) \)

and \( NP.sem = \text{AyCaramba} \)

• So, \( S.sem = \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, \text{Meat}) \) (AyCa.) = \( \exists e. \text{Serving}(e) \land \text{Server}(e, \text{AyCaramba}) \land \text{Served}(e, \text{Meat}) \)
Problem with these rules

• Consider the sentence Every child sleeps.

\[ \forall x. \text{Child}(x) \Rightarrow \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, x) \]

• Meaning of Every child (involving \( x \)) is interleaved with meaning of sleeps

• Our existing rules can’t handle this example, or quantifiers (from NPs with determiners) in general.

• How can we fix things?
Breaking it down

• What is the meaning of Every child anyway?

• Every child ...

...sleeps \quad \forall x. \ Child(x) \Rightarrow \exists e. \ Sleeping(e) \land Sleeper(e, x)

...cries \quad \forall x. \ Child(x) \Rightarrow \exists e. \ Crying(e) \land Crier(e, x)

...talks \quad \forall x. \ Child(x) \Rightarrow \exists e. \ Talking(e) \land Talker(e, x)

...likes pizza \quad \forall x. \ Child(x) \Rightarrow \exists e. \ Liking(e) \land Liker(e, x) \land Likee(e, pizza)
Breaking it down

• What is the meaning of Every child anyway?

• Every child ...

  ...sleeps \( \forall x. \ Child(x) \implies \exists e. \ Sleeping(e) \land Sleeper(e, x) \)

  ...cries \( \forall x. \ Child(x) \implies \exists e. \ Crying(e) \land Crier(e, x) \)

  ...talks \( \forall x. \ Child(x) \implies \exists e. \ Talking(e) \land Talker(e, x) \)

  ...likes pizza \( \forall x. \ Child(x) \implies \exists e. \ Liking(e) \land Liker(e, x) \land Likee(e, pizza) \)

• So it looks like the meaning is something like

\[ \forall x. \ Child(x) \implies Q(x) \]

• where \( Q(x) \) is some (potentially quite complex) expression with a predicate-like meaning
Could this work with our rules?

- We said $S.sem$ should be $VP.sem(NP.sem)$

- but

  $\lambda y. \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, y) \ (\forall x. \text{Child}(x) \ \Rightarrow \ Q(x))$

  yields

  $\exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, \forall x. \text{Child}(x) \ \Rightarrow \ Q(x))$

- This isn’t a valid FOL: complex expressions cannot be arguments to predicates.
Switching things around

• But if we define $S.sem$ as $NP.sem(VP.sem)$ it works!

• First, must make $NP.sem$ into a functor by adding $\lambda$:

$$\lambda Q \forall x. \ Child(x) \Rightarrow Q(x)$$
Switching things around

• But if we define $S.sem$ as $NP.sem(VP.sem)$ it works!

• First, must make $NP.sem$ into a functor by adding $\lambda$:

$$\lambda Q \forall x. \text{Child}(x) \Rightarrow Q(x)$$

• Then, apply it to $VP.sem$:

$$\lambda Q \forall x. \text{Child}(x) \Rightarrow Q(x) \ (\lambda y. \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, y))$$

$$\forall x. \text{Child}(x) \Rightarrow (\lambda y. \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, y)) (x)$$

$$\forall x. \text{Child}(x) \Rightarrow \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, x)$$
But, how can we get our NP.sem?

- We will need a new set of noun rules:

  Noun $\rightarrow$ Child $\{\lambda x. \text{Child}(x)\}$
  Det $\rightarrow$ Every $\{\lambda P. \lambda Q. \forall x. P(x) \Rightarrow Q(x)\}$
  NP $\rightarrow$ Det Noun $\{\text{Det.sem}(\text{Noun.sem})\}$
But, how can we get our NP.sem?

- We will need a new set of noun rules:
  
  \[
  \begin{align*}
  \text{Noun} & \rightarrow \text{Child} & \{ \lambda x. \text{Child}(x) \} \\
  \text{Det} & \rightarrow \text{Every} & \{ \lambda P. \lambda Q. \forall x. P(x) \Rightarrow Q(x) \} \\
  \text{NP} & \rightarrow \text{Det Noun} & \{ \text{Det.sem(Noun.sem)} \}
  \end{align*}
  \]

- So, Every child is derived as

  \[
  \lambda P. \lambda Q. \forall x. P(x) \Rightarrow Q(x) (\lambda x. \text{Child}(x))
  \]

  \[
  \lambda Q \forall x. (\lambda x. \text{Child}(x))(x) \Rightarrow Q(x)
  \]

  \[
  \lambda Q \forall x. \text{Child}(x) \Rightarrow Q(x)
  \]
One last problem

- Our previous MRs for proper nouns were not functors, so don't work with our new rule $S \rightarrow NP \; VP \; \{NP.sem(VP.sem)\}$.

$\exists e. Sleeping(e) \wedge Sleeper(e, y)) \Rightarrow $ Not valid!

Kate ($\lambda y. \exists e. Sleeping(e) \wedge Sleeper(e, y))$
\( \lambda \) to the rescue again

- Assign a different MR to proper nouns, allowing them to take VPs as arguments:

  \[
  \text{ProperNoun} \rightarrow \text{Kate} \quad \{\lambda P. \ P(Kate)\}
  \]

- For \text{Kate sleeps}, this gives us

  \[
  \lambda P. \ P(Kate) \ (\lambda y. \ \exists \ e. \ Sleeping(e) \wedge Sleeper(e, y))
  \\
  (\lambda y. \ \exists \ e. \ Sleeping(e) \wedge Sleeper(e, y))(\text{Kate})
  \\
  \exists \ e. \ Sleeping(e) \wedge Sleeper(e, \text{Kate})
  \]
\( \lambda \) to the rescue again

- Assign a different MR to proper nouns, allowing them to take VPs as arguments:

  \[
  \text{ProperNoun} \rightarrow \text{Kate} \quad \{ \lambda P. P(Kate) \}
  \]

- For Kate sleeps, this gives us

  \[
  \lambda P. P(Kate) (\lambda y. \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, y)) \\
  (\lambda y. \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, y))(\text{Kate}) \\
  \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, \text{Kate})
  \]

- Terminology: we type-raised the argument \( a \) of a function \( f \), turning it into a function \( g \) that takes \( f \) as argument. (!)

  - The final returned value is the same in either case.
Final grammar?

\[
S \rightarrow NP\ VP\ \{NP.sem(VP.sem)\} \\
VP \rightarrow Verb\ \{Verb.sem\} \\
VP \rightarrow Verb\ NP\ \{Verb.sem(NP.sem)\} \\
NP \rightarrow Det\ Noun\ \{Det.sem(Noun.sem)\} \\
NP \rightarrow ProperNoun\ \{ProperNoun.sem\} \\
Det \rightarrow Every\ \{\lambda P. \lambda Q. \forall x. P(x) \Rightarrow Q(x)\} \\
Noun \rightarrow Child\ \{\lambda x. Child(x)\} \\
ProperNoun \rightarrow Kate\ \{\lambda P. P(Kate)\} \\
Verb \rightarrow sleeps\ \{\lambda x. \exists e. Sleeping(e) \land Sleeper(e, x)\} \\
Verb \rightarrow serves\ \{\lambda y. \lambda x. \exists e. Serving(e) \land Server(e, x) \land Served(e, y)\}
\]
Complications

• This grammar still applies Verbs to NPs when *inside* the VP.

• Try doing this with our new type-raised NPs and you will see it doesn’t work.

• In practice, we need automatic type-raising rules that can be used exactly when needed, otherwise we keep the base type.
  
  – e.g., “base type” of proper noun is “entity”, not “function from (functions from entities to truth values) to truth values”.
What we did achieve

Developed a grammar with semantic attachments using many ideas currently in use:

- existentially quantified variables represent events
- lexical items have function-like $\lambda$-expressions as MRs
- non-branching rules copy semantics from child to parent
- branching rules apply semantics of one child to the other(s) using $\lambda$-reduction.
Compositional semantics: bigger picture

Incorporating semantics into a grammar involves two separate but intertwined processes:

1. Figuring out the right representation for a single constituent based on the parts of that constituent.

2. Figuring out the right representation for a category of constituents based on the grammar rules that use that category.
Semantic parsing algorithms

• Given a CFG with semantic attachments, how do we obtain the semantic analysis of a sentence?

• One option (integrated): Modify syntactic parser to apply semantic attachments at the time syntactic constituents are constructed.

• Second option (pipelined): Complete the syntactic parse, then walk the tree bottom-up to apply semantic attachments.
Learning a semantic parser

• Much current research focuses on *learning* semantic grammars rather than *hand-engineering* them.

• Given sentences paired with meaning representations, e.g.,

Every child sleeps  \( \forall x. \ Child(x) \Rightarrow \exists e. \ Sleeping(e) \land Sleeper(e, x) \)
AyCaramba serves meat  \( \exists e. \ Serving(e) \land Server(e, AyCaramba) \land Served(e, Meat) \)

• Can we automatically learn
  – Which words are associated with which bits of MR?
  – How do those bits combine (in parallel with the syntax) to yield the final MR?

• And, can we do this with less well-specified semantic representations?
Summary

• Semantic analysis/semantic parsing: the process of deriving a meaning representation from a sentence.

• Uses the grammar and lexicon (augmented with semantic information) to create context-independent literal meanings.

• $\lambda$-expressions handle compositionality, building semantics of larger forms from smaller ones.

• Final meaning representations are expressions in first-order logic.