Last time

- Discussed properties we want from a meaning representation:
  - compositional
  - verifiable
  - canonical form
  - unambiguous
  - expressive
  - allowing inference

- Argued that first-order logic has all of these except compositionality, and is a good fit for natural language.

- Adding $\lambda$-expressions to FOL allows us to compute meaning representations compositionally.

Today

- We’ll see how to use $\lambda$-expressions in computing meanings for sentences: syntax-driven semantic analysis.

- But first: a final improvement to event representations

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:

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

As in the sentence:

\[
\text{John gave Mary a book.}
\]

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

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

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

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

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

Problem with event representations

- Predicates in First-order Logic have fixed arity
- Requires separate \textit{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 \textit{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 \textit{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 \textit{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: \textit{Giving}(e); remaining conjuncts represent the participants (semantic roles).
Entailment relations

• This representation automatically gives us logical entailment relations between events. ("A entails B" means "A ⇒ B").

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

Sharon Goldwater Semantic analysis 8

Entailment relations

• This representation automatically gives us logical entailment relations between events. ("A entails B" means "A ⇒ B").

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

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

entails

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

• Can add as many semantic roles as needed for the event.

Sharon Goldwater Semantic analysis 9

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 \( \lambda \)-expressions.

Sharon Goldwater Semantic analysis 10

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 order and syntactic 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:

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

• \(A.sem\) (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:

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

• 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:

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

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

• We can do the same with reified events:

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

Building larger constituents

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

Building larger constituents

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

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

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

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

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

- Final rule is:
  \[ S \rightarrow NP \; VP \; \{ VP.sem(NP.sem) \} \]

- 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 =
  AyCaramba

- So, S.sem =
  \[ \lambda x. \exists e. \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, \text{Meat}) \]

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

- As next slides show, our existing rules can’t handle this example, or quantifiers (from NPs with determiners) in general.

- We’ll show the problem, then the solution.

Breaking it down

- What is the meaning of Every child anyway?

  - Every child ...
    - ...sleeps \[ \forall x. \text{Child}(x) \Rightarrow \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, x) \]
    - ...cries \[ \forall x. \text{Child}(x) \Rightarrow \exists e. \text{Crying}(e) \land \text{Crier}(e, x) \]
    - ...talks \[ \forall x. \text{Child}(x) \Rightarrow \exists e. \text{Talking}(e) \land \text{Talker}(e, x) \]
    - ...likes pizza \[ \forall x. \text{Child}(x) \Rightarrow \exists e. \text{Liking}(e) \land \text{Liker}(e, x) \land \text{Likee}(e, \text{pizza}) \]

  - So it looks like the meaning is something like
    \[ \forall x. \text{Child}(x) \Rightarrow Q(x) \]

  - 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

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

yields

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

• 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 $\lambda$:

$\lambda Q \forall x. Child(x) \Rightarrow Q(x)$

• Then, apply it to $VP.sem$:

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

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

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

But, how can we get the right NP.sem?

• We will need a new set of noun rules:

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>Child</td>
<td>${\lambda x. Child(x)}$</td>
</tr>
<tr>
<td>Det</td>
<td>Every</td>
<td>${\lambda P. \lambda Q. \forall x. P(x) \Rightarrow Q(x)}$</td>
</tr>
<tr>
<td>NP</td>
<td>Det Noun</td>
<td>${Det.sem(Noun.sem)}$</td>
</tr>
</tbody>
</table>

Sharon Goldwater Semantic analysis 24

Sharon Goldwater Semantic analysis 25

Sharon Goldwater Semantic analysis 26

Sharon Goldwater Semantic analysis 27
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
  
  \[
  \lambda P. \lambda Q. \forall x. P(x) \Rightarrow Q(x) \ (\lambda x. \text{Child}(x))
  \]

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

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

\[\lambda \text{to the rescue again}\]

- Assign a different MR to proper nouns, allowing them to take VPs as arguments:
  
  \[
  \begin{align*}
  \text{ProperNoun} & \rightarrow \text{Kate} \{ \lambda P. P(\text{Kate}) \} \\
  \text{For Kate sleeps, this gives us}
  \end{align*}
  \]

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

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

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

One last problem

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

\[
\begin{array}{c}
\text{S} \\
\text{NP} \\
\text{VP} \\
\text{ProperNoun} \\
\text{Verb} \\
\text{Kate} \\
\text{sleeps}
\end{array}
\]

- Not valid!

\[\lambda \text{to the rescue again}\]

- Assign a different MR to proper nouns, allowing them to take VPs as arguments:
  
  \[
  \begin{align*}
  \text{ProperNoun} & \rightarrow \text{Kate} \{ \lambda P. P(\text{Kate}) \} \\
  \text{For Kate sleeps, this gives us}
  \end{align*}
  \]

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

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

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

- Terminology: we **type-raised** the 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. \ \text{Child}(x)\}
ProperNoun \rightarrow Kate \ \{\lambda P. \ P(Kate)\}
Verb \rightarrow sleeps \ \{\lambda x. \ \exists e. \ \text{Sleeping}(e) \land \text{Sleeper}(e, x)\}
Verb \rightarrow serves \ \{\lambda y. \ \lambda x. \ \exists e. \ \text{Serving}(e) \land \text{Server}(e, x) \land \text{Served}(e, y)\}

Complications

\begin{itemize}
  \item This grammar still applies Verbs to NPs when \textit{inside} the VP.
  \item Try doing this with our new type-raised NPs and you will see it doesn't work.
  \item 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”.
\end{itemize}

What we did achieve

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

\begin{itemize}
  \item existentially quantified variables represent events
  \item lexical items have function-like \(\lambda\)-expressions as MRs
  \item non-branching rules copy semantics from child to parent
  \item branching rules apply semantics of one child to the other(s) using \(\lambda\)-reduction.
\end{itemize}

Semantic parsing algorithms

\begin{itemize}
  \item Given a CFG with semantic attachments, how do we obtain the semantic analysis of a sentence?
    \item One option (integrated): Modify syntactic parser to apply semantic attachments at the time syntactic constituents are constructed.
    \item Second option (pipelined): Complete the syntactic parse, then walk the tree bottom-up to apply semantic attachments.
\end{itemize}
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 \( \forall x. \mathrm{Child}(x) \Rightarrow \exists e. \mathrm{Sleeping}(e) \land \mathrm{Sleeper}(e, x) \)
  AyCaramba serves meat \( \exists e. \mathrm{Serving}(e) \land \mathrm{Server}(e, \text{AyCaramba}) \land \mathrm{Served}(e, \text{Meat}) \)

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

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

See, e.g., Zettlemoyer and Collins (2005); Kwiatkowski et al. (2010); Reddy et al. (2014); Choi et al. (2015)

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.

References


