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:
NP give NP$_1$ 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.
\textit{Giving}_2(\textit{John, Mary, Book, Susan})

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

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

John gave Mary a book with trepidation.
\textit{Giving}_5(\textit{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 \textit{subcategorisation frame} (number/type/position of arguments).
- Separate predicates have no logical relation, but they ought to.
  - Ex. if \textit{Giving}_3(a, b, c, d, e) is true, then so are \textit{Giving}_2(a, b, c, d) and \textit{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 \textbf{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 \textbf{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
  \begin{align*}
  \exists e, z. \textit{Giving}(e) \land \textit{Giver}(e, \textit{John}) \land \textit{Givee}(e, \textit{Mary}) \\
  \land \textit{Given}(e, z) \land \textit{Book}(z)
  \end{align*}

- 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 \( \Rightarrow \) B”).

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

Sharon Goldwater Semantic analysis 8

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.

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:

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

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

Building larger constituents

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

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

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

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

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

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

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

Problem with these rules

• Consider the sentence \textit{Every child sleeps}.
  \[ \forall x. \text{Child}(x) \Rightarrow \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, x) \]

• Meaning of \textit{Every child} (involving \( x \)) is interleaved with meaning of \textit{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 \textit{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
  \[
  \lambda y. \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, y) \land \forall x. \text{Child}(x) \Rightarrow Q(x)
  \]
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 \( \lambda \):
  \[
  \lambda Q \forall x. \text{Child}(x) \Rightarrow Q(x)
  \]
- Then, apply it to \( VP.sem \):
  \[
  \forall x. \text{Child}(x) \Rightarrow (\lambda y. \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, y)) (x)
  \]
  \[
  \forall x. \text{Child}(x) \Rightarrow \exists e. \text{Sleeping}(e) \land \text{Sleeper}(e, x)
  \]

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

- We will need a new set of noun rules:
  - Noun → Child \( \{ \lambda x. \text{Child}(x) \} \)
  - Det → Every \( \{ \lambda P. \lambda Q. \forall x. P(x) \Rightarrow Q(x) \} \)
  - NP → Det Noun \( \{ \text{Det.sem(Noun.sem)} \} \)
But, how can we get our NP.sem?

• We will need a new set of noun rules:
  
  Noun → Child \{λx. \text{Child}(x)\}
  Det → Every \{λP. λQ. ∀x. P(x) ⇒ Q(x)\}
  NP → Det Noun \{Det.sem(Noun.sem)\}

• So, Every child is derived as
  
  λP. λQ. ∀x. P(x) ⇒ Q(x) (λx. \text{Child}(x))

λQ ∀x. (λx. \text{Child}(x))(x) ⇒ Q(x)
λQ ∀x. \text{Child}(x) ⇒ Q(x)

λ to the rescue again

• Assign a different MR to proper nouns, allowing them to take VPs as arguments:
  
  ProperNoun → Kate \{λP. P(Kate)\}

• For Kate sleeps, this gives us
  
  λP. P(Kate) (λy. ∃e. \text{Sleeping}(e) ∧ \text{Sleeper}(e, y))
  (λy. ∃e. \text{Sleeping}(e) ∧ \text{Sleeper}(e, y))(Kate)
  ∃e. \text{Sleeping}(e) ∧ \text{Sleeper}(e, Kate))

λ to the rescue again

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

\[ S \rightarrow \text{NP} \rightarrow \text{VP} \rightarrow \text{det} \rightarrow \text{every} \rightarrow \text{proper noun} \rightarrow \text{verb} \rightarrow \text{Kate} (\lambda y. \exists e. \text{Sleeping}(e) ∧ \text{Sleeper}(e, y)) \Rightarrow \text{Not valid!} \]

One last problem

• Assign a different MR to proper nouns, allowing them to take VPs as arguments:
  
  ProperNoun → Kate \{λP. P(Kate)\}

• For Kate sleeps, this gives us
  
  λP. P(Kate) (λy. ∃e. \text{Sleeping}(e) ∧ \text{Sleeper}(e, y))
  (λy. ∃e. \text{Sleeping}(e) ∧ \text{Sleeper}(e, y))(Kate)
  ∃e. \text{Sleeping}(e) ∧ \text{Sleeper}(e, 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?

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

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 now in use:

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

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.,

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

\[
\text{AyCaramba serves meat} \quad \exists e. \text{Serving}(e) \land \text{Server}(e, \text{AyCaramba}) \land \text{Served}(e, \text{Meat})
\]

• 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)

References


