Chapter 7

Semantics

When we began our study of syntax, I had to spend some time convincing you that syntax exists (that is, that there is some kind of representation of syntactic structure in your mind when you use language), but the basic shape of those representations has not been all that controversial.

By contrast, as we turn our attention to *semantics*, or meaning, I probably don't need to convince you at all that meaning exists, but in theories of semantics and computational approaches to semantics, representations of meaning come in all kinds of shapes and sizes.

Semantics is often divided into the meanings of words, known as *lexical semantics*, and the meanings of sentences, which is usually just called *semantics*. As I see it, there are three main approaches to representing meaning in NLP:

1. Vectors: whose components are features, either designed by humans or automatically learned.
2. Graphs: in which nodes are usually entities and edges are various kinds of relationships among them.
3. Logic: formulas of logics of various kinds, or SQL queries, or even computer programs.

7.1 Vectors

We've already seen lots of examples of vector representations of text. For example, we saw, in IBM Model 1, possibly the simplest way to represent the "meaning" of a text, which is to assume that each word in the text contributes a little bit of meaning, which we can represent as a one-hot vector, and that we can combine the "meanings" of words simply by adding up their vectors.

$$\text{the cat sat on the mat} \longrightarrow \{\text{cat} : 1, \text{mat} : 1, \text{on} : 1, \text{sat} : 1, \text{the} : 2\}$$

In information retrieval, this is called a *term vector*, but in NLP it's more commonly known (somewhat pejoratively) as a *bag of words*. This representation completely ignores any semantic relationships between words (like synonymy) and any kind of structure of the text (like word order or syntax). Yet, in many situations, it can be very effective.

After IBM Model 1, recall that we factored the model so that it computed a d-dimensional dense representation of the sentence. This sentence representation was computed as a by-product of a model trained to carry out a particular task, like translation. Next week, we'll revisit vector representations of texts in the context of the task of *text classification*.

7.2 Graphs

The next broad category of semantic representation I want to talk about is semantic *graphs*. I'm including under this heading a bunch of tasks like

- Named entity recognition: Which noun phrases are names of people, places, etc., and what entity do they refer to?
- Coreference resolution: Which noun phrases (including pronouns) refer to the same thing?
- Word sense disambiguation: If a word has more than one sense, which one is being used in this context?
- Semantic role labeling: For each action in the sentence, who/what is the agent (the one doing the action), the patient (the one upon whom the action is done), etc.?
- Relation extraction: Sometimes we're interested in higher-level relationships between entities, like "x is the president of y."

Ultimately, the entities and relationships that these tasks find can be assembled into a graph – either a graph that represents the meaning of a sentence, or a big graph that represents the knowledge contained in a whole collection of text.

Above we mentioned various semantics-related tasks like semantic role labeling (Gildea and Jurafsky, 2000), word sense disambiguation (Brown et al., 1991), coreference resolution (Soon, Ng, and Lim, 2001), and so on. Resources like OntoNotes (Hovy et al., 2006) provided separate resources for each of these tasks.

Some more recent work in semantic processing tries to consolidate these tasks into one. For example, the Abstract Meaning Representation (AMR) Bank (Banarescu et al., 2013) began as an effort to unify the various annotation layers of OntoNotes. Others include: the Prague Dependency Treebank (Böhmová et al., 2003), DeepBank (Oepen and Lønning, 2006), and Universal Conceptual Cognitive Annotation (Abend and Rappoport, 2013). By and large, these resources are based on, or equivalent to, *graphs*, in which vertices stand for entities and edges stand for semantic relations among them.

Abstract Meaning Representations

Here, I'll focus on AMRs, just because they're the representation I'm most familiar with. AMRs can be written in a serialized form or as directed graphs. Examples of these two representations, from the AMR Bank (LDC2014T12), are reported in

Figure 7.1 and Figure 7.2. Nodes are labeled, in order to convey lexical information. Edges are labeled to convey information about semantic roles. Labels at the edges need not be unique, meaning that edges impinging on the same node might have the same label. Furthermore, our DAGs are not ordered, meaning that there is no order relation for the edges impinging at a given node, as is usually the case in standard graph structures. A node can appear in more than one place (for example, in Figure 7.1, node `s2` appears six times).

The numbers (e.g., ask-01) require some explanation. These are from PropBank (Palmer, Gildea, and Kingsbury, 2005), which catalogues and numbers, for each verb, the different senses of the verb and ways it can be used. For example,

- ask-01 is for asking questions
- ask-02 is for asking favors
- ask-03 is for asking a price
- ask_out-04 is for asking someone on a date.

Each of these senses comes with a numbered list of arguments. For example, for ask-01,

- arg0 is the asker
- arg1 is the question
- arg2 is the hearer.

AMR parsing

Semantic parsing is the task of taking a natural language sentence and mapping it to a representation of its meaning. If the semantic representation is AMR, we call this AMR parsing.

These days, it's easy to build a barebones AMR parser – just run a neural machine translation system on parallel text consisting of English sentences and their AMRs in textual format (Figure 7.1). Research on AMR parsing has gotten plenty more sophisticated than that, but here I want to focus on one improvement to the basic NMT system (which you will implement in HW4).

Unlike language-to-language translation, it's typical for the AMR to have words in common with the source text. The NMT system can do a good job learning to copy these words if they're frequent, but for rare words, it may do so less reliably, and for unknown words, it will be unable to. So we want to add a *copy* mechanism to the model.

The basic idea is to introduce a fake target word, `COPY`, which instructs the system to copy a word from the source sentence. Which word? We use the source-to-target attention, which is a distribution over source positions, to choose one source word.

We're only interested in the last two steps of the model, which were the same in our presentation of both the RNN and Transformer models. These are for the

```
(a / and
      :op1 (a2 / ask-01
            :ARG0 (i / i)
            :ARG1 (t / thing
                  :ARG1-of (t2 / think-01
                        :ARG0 (s2 / she)
                        :ARG2 (l / location
                              :location-of (w / we))))
            :ARG2 s2)
      :op2 (s / say-01
            :ARG0 s2
            :ARG1 (a3 / and
                  :op1 (w2 / want-01 :polarity -
                        :ARG0 s2
                        :ARG1 (t3 / think-01
                              :ARG0 s2
                              :ARG1 l))
                  :op2 (r / recommend-01
                        :ARG0 s2
                        :ARG1 (c / content-01
                              :ARG1 i
                              :ARG2 (e / experience-01
                                    :ARG0 w))
                        :ARG2 i))
            :ARG2 i)
      :op3 c)
```

Figure 7.1: Example AMR in its standard format, number `DF-200-192403-625_0111.7` from the AMR Bank. The sentence is: “I asked her what she thought about where we’d be and she said she doesn’t want to think about that, and that I should be happy about the experiences we’ve had (which I am).”

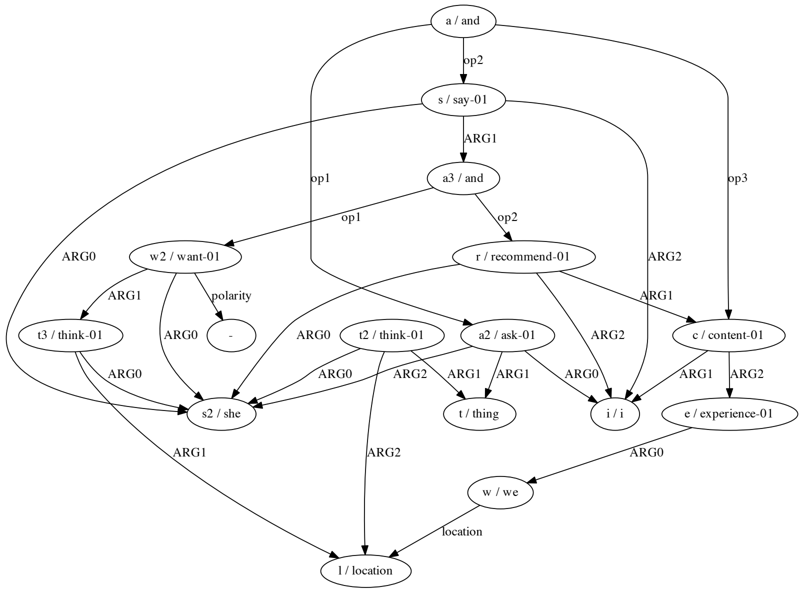


Figure 7.2: The AMR of Figure 7.1, presented as a directed graph.

context vector $\mathbf{c}^{(i)}$, and the output word distribution:

$$\begin{aligned}
\mathbf{g}^{(i)} &\in \mathbb{R}^d \\
\mathbf{H} &\in \mathbb{R}^{n\times d} \\
\mathbf{o}^{(i)} &\in \mathbb{R}^d \\
\mathbf{c}^{(i)} &\in \mathbb{R}^d \\
\mathbf{c}^{(i)} &= \text{Attention}(\mathbf{g}^{(i)}, \mathbf{H}, \mathbf{H}) && (7.1) \\
P(e_{i+1}) &= \text{SoftmaxLayer}^{\boxed{6}}(\mathbf{o}^{(i)}). && (7.2)
\end{aligned}$$

The equation for $\mathbf{c}^{(i)}$ computes both the attention and the weighted average, so we're going to "break it open" to get at the attention inside:

$$\begin{aligned}
\alpha^{(i)} &\in \mathbb{R}^n && (7.3) \\
\alpha^{(i)} &= \text{softmax}\, \mathbf{H}\mathbf{g}^{(i)} && (7.4)
\end{aligned}$$

The output word distribution now includes COPY. We modify this to:

$$\begin{aligned}
\mathbf{p}^{(i)} &\in \mathbb{R}^n && (7.5) \\
\mathbf{p}^{(i)} &= \text{SoftmaxLayer}^{\boxed{6}}(\mathbf{o}^{(i)}) && (7.6) \\
P(e) &= \mathbf{p}^{(i)}_e + \mathbf{p}^{(i)}_{\texttt{COPY}} \sum_{\substack{j=1,\ldots,n \\ f_j=e}} \alpha^{(i)}_j. && (7.7)
\end{aligned}$$

This means that there are one or more ways of choosing word e: first, we could choose it directly from the output distribution $\mathbf{o}^{(i)}$, or, for each source word f_j that is equal to e, we could copy word f_j with probability $\mathbf{p}^{(i)}_{\texttt{COPY}}\alpha_j$.

Note that

- $\alpha^{(i)}$ and $\mathbf{p}^{(i)}$ are both vectors of probabilities, not log-probabilities.
- The test $f_j = e$ compares the words *as words*, not as numbers.

7.3 Logic

The last category of semantic representations is that of logical formulas, understood broadly to include not only logics like first-order logic, but languages like SQL or even programming languages.

In these notes, I'd like to focus on a traditional approach to semantics called *Montague grammar.*

7.3.1 Logical forms

We start with a very simple example:

(7.8) a. John sees Mary.
b. $\mathit{see}(\mathit{John}, \mathit{Mary})$.

Entities are represented by constants or variables, and events are represented by predicates.

A variation (called neo-Davidsonian semantics) represents events by variables, too:

(7.9) a. John sees Mary.
b. $\exists e.\mathit{see}(e) \land \mathit{agent}(e, \mathit{John}) \land \mathit{theme}(e, \mathit{Mary})$.

This is quite similar to AMR. But let's stick with events as predicates.

A key way that logical semantics differs from graph representations like AMR is in handling of quantifiers.

(7.10) a. John sees a girl.
b. $\exists g.\mathit{girl}(g) \land \mathit{see}(\mathit{John}, g)$.

(7.11) a. A boy sees Mary.
b. $\exists b.\mathit{boy}(b) \land \mathit{see}(b, \mathit{Mary})$.

7.3.2 Compositionality

How do we compute these representations? We want to follow the principle of *compositionality*, that the meaning of any expression is computed from the meaning of its subexpressions. In other words, we want to write a recursive function that processes a syntax tree bottom-up, something like

```
function SEMANTICS(root)
```

```
    **if** root = S **and** root.children = (NP, VP) **then**
        $s_1 \leftarrow$ SEMANTICS(root.children[1])
        $s_2 \leftarrow$ SEMANTICS(root.children[2])
        build $s$ from $s_1$ and $s_2$
        **return** $s$
    **else**...
    **end if**
**end function**
```

So we want to associate with each context-free grammar rule (e.g., S $\rightarrow$ NP VP) a little function that build the semantics of S from the semantics of NP and VP. To do that, it will be convenient to have some new notation for writing little functions.

7.3.3 Lambda calculus

A λ-expression (lambda-expression) is a self-contained way of writing a function. Many programming languages now have them:

λ-calculus	$\lambda x.x \cdot x$
Scheme/Lisp	`(lambda (x) (* x x))`
Python	`lambda x: x * x`
C++	`[](float x) { return x * x; }`

In λ-calculus, the application of a function f to an expression e is simply written as $f e$. So

$$(\lambda x.x \cdot x)10 \longrightarrow 10 \cdot 10 = 100.$$

Lambda expressions can do a lot of things you might not expect them to be able to do at first; here, I want to mention just one. Traditionally, λ-expressions take exactly one argument. But you can effectively write a function of two arguments as a function that returns another function, like this:

$$f = \lambda x.\lambda y.\sqrt{x^2 + y^2} \tag{7.12}$$

$$f\ 3\ 4 = (\lambda x.\lambda y.\sqrt{x^2 + y^2})\ 3\ 4 \tag{7.13}$$

$$\longrightarrow (\lambda y.\sqrt{3^2 + y^2})\ 4 \tag{7.14}$$

$$\longrightarrow \sqrt{3^2 + 4^2} \tag{7.15}$$

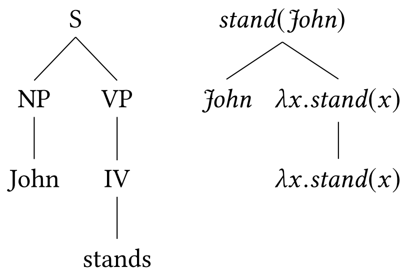
$$= 5. \tag{7.16}$$

This is called *currying* after Haskell Curry, who had nothing to do with it.

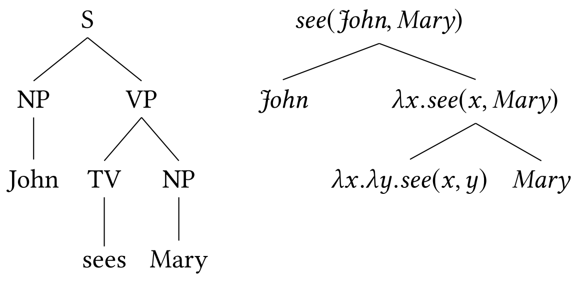
7.3.4 Examples

Here's a very simple CFG, each of whose rules is associated with a function that computes the semantics of the left-hand side (that is, the parent) in terms of the semantics of the right-hand side (that is, the children):

S → NP VP	$\lambda x.\lambda P.Px$
NP → John	*John*
NP → Mary	*Mary*
VP → IV	$\lambda P.P$
IV → stands	$\lambda x.stand(x)$

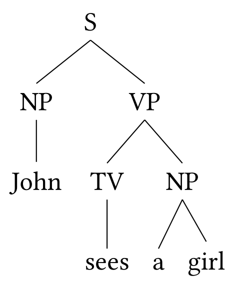


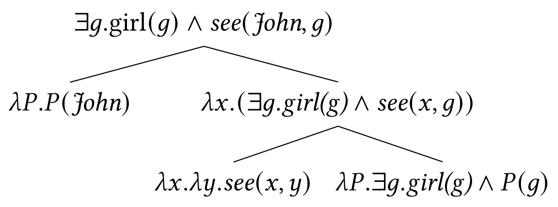
S → NP VP	$\lambda x.\lambda P.Px$
NP → John	*John*
NP → Mary	*Mary*
VP → IV	$\lambda P.P$
VP → TV NP	$\lambda P.\lambda y.\lambda x.P(x,y)$
IV → stands	$\lambda x.stand(x)$
TV → sees	$\lambda x.\lambda y.see(x,y)$



When we try to get examples like (7.10–7.11), however, problems arise. The meaning of "a" should be $\exists$, but how do we get this quantifier to appear on the very outside of the formula? The solution is to flip everything around so that the semantics for "a boy" should not just be a formula representing a boy; it should be a function that takes a predicate P about boys and returns the formula $\exists b.boy(b) \land P(b)$.

S $\longrightarrow$ NP VP	$\lambda f.\lambda P.fP$
NP $\longrightarrow$ John	$\lambda P.P(\mathit{John})$
NP $\longrightarrow$ Mary	$\lambda P.P(\mathit{Mary})$
NP $\longrightarrow$ a boy	$\lambda P.(\exists b.\mathit{boy}(b) \wedge Pb)$
NP $\longrightarrow$ a girl	$\lambda P.(\exists g.\mathit{girl}(g) \wedge Pg)$
VP $\longrightarrow$ TV NP	$\lambda P.\lambda f.\lambda x.f(\lambda y.Pxy)$
TV $\longrightarrow$ sees	$\lambda x.\lambda y.\mathit{see}(x,y)$





The computation of VP is particularly complicated, so we write it out step by step:

$$\begin{aligned}
&\underline{(\lambda P.\lambda f.\lambda x.f(\lambda y.Pxy))(\lambda x.\lambda y.\mathit{see}(x,y))}(\lambda P.(\exists g.\mathit{girl}(g) \wedge Pg))\\
&\longrightarrow (\lambda f.\lambda x.f(\lambda y.\underline{(\lambda x.\lambda y.\mathit{see}(x,y))x}y))(\lambda P.(\exists g.\mathit{girl}(g) \wedge Pg))\\
&\longrightarrow (\lambda f.\lambda x.f(\lambda y.\underline{(\lambda y.\mathit{see}(x,y))y}))(\lambda P.(\exists g.\mathit{girl}(g) \wedge Pg))\\
&\longrightarrow \underline{(\lambda f.\lambda x.f(\lambda y.\mathit{see}(x,y)))(\lambda P.(\exists g.\mathit{girl}(g) \wedge Pg))}\\
&\longrightarrow \lambda x.\underline{(\lambda P.(\exists g.\mathit{girl}(g) \wedge Pg))(\lambda y.\mathit{see}(x,y))}\\
&\longrightarrow \lambda x.(\exists g.\mathit{girl}(g) \wedge \underline{(\lambda y.\mathit{see}(x,y))g})\\
&\longrightarrow \lambda x.(\exists g.\mathit{girl}(g) \wedge \mathit{see}(x,g)).
\end{aligned}$$

Finally, we refine our grammar so that "a" has its own semantics.

S $\longrightarrow$ NP VP	$\lambda f.\lambda P.fP$
NP $\longrightarrow$ John	$\lambda P.P(\mathit{John})$
NP $\longrightarrow$ Mary	$\lambda P.P(\mathit{Mary})$
NP $\longrightarrow$ Det N	$\lambda d.\lambda f.df$
Det $\longrightarrow$ a	$\lambda N.\lambda P.(\exists x.Nx \wedge Px)$
Det $\longrightarrow$ every	$\lambda N.\lambda P.(\forall x.Nx \wedge Px)$
N $\longrightarrow$ boy	$\lambda b.\mathit{boy}(b)$
N $\longrightarrow$ girl	$\lambda g.\mathit{girl}(g)$
VP $\longrightarrow$ TV NP	$\lambda P.\lambda f.\lambda x.f(\lambda y.Pxy)$
TV $\longrightarrow$ sees	$\lambda x.\lambda y.\mathit{see}(x,y)$

References

Abend, Omri and Ari Rappoport (2013). “Universal Conceptual Cognitive Annotation (UCCA)”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria, pp. 228–238. URL: `http://www.aclweb.org/anthology/P13-1023`.

Banarescu, Laura et al. (2013). “Abstract Meaning Representation for Sembanking”. In: *Proceedings of the Linguistic Annotation Workshop*. Sofia, Bulgaria, pp. 178–186.

Böhmová, Alena et al. (2003). “The Prague Dependency Treebank: A Three-Level Annotation Scenario”. In: *Treebanks: Building and Using Parsed Corpora*. Ed. by A. Abeillé. Kluwer, pp. 103–127.

Brown, Peter F. et al. (1991). “Word-sense disambiguation using statistical methods”. In: *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL-91)*. Berkeley, CA, pp. 264–270.

Gildea, Daniel and Daniel Jurafsky (2000). “Automatic Labeling of Semantic Roles”. In: *Proceedings of the 38th Annual Conference of the Association for Computational Linguistics (ACL-00)*. Hong Kong, pp. 512–520.

Hovy, Eduard et al. (2006). “OntoNotes: The 90% Solution”. In: *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*. New York City, USA, pp. 57–60. URL: `http://www.aclweb.org/anthology/N/N06/N06-2015`.

Oepen, Stephan and Jan Tore Lønning (2006). “Discriminant-Based MRS Banking”. In: *International Conference on Language Resources and Evaluation (LREC)*. Genoa, pp. 1250–1255.

Palmer, Martha, Daniel Gildea, and Paul Kingsbury (2005). “The Proposition Bank: An Annotated Corpus of Semantic Roles”. In: *Computational Linguistics* 31.1, pp. 71–106. DOI: `10.1162/0891201053630264`. URL: `https://www.aclweb.org/anthology/J05-1004`.

Soon, Wee Meng, Hwee Tou Ng, and Daniel Chung Long Lim (2001). “A Machine Learning Approach to Coreference Resolution of Noun Phrases”. In: *Computational Linguistics* 27.4, pp. 521–544.