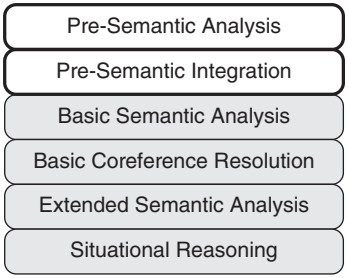
Linguistics for the Age of AI

The title-level DOI for this work is:

doi:10.7551/mitpress/13618.001.0001

3

Pre-Semantic Analysis and Integration



Before addressing semantics, the LEIA carries out two preparatory stages of analysis. The first one, *Pre-Semantic Analysis*, includes preprocessing and syntactic parsing, for which we use externally developed tools. The reasons why we have not developed knowledge-based alternatives are these: tools addressing the needed phenomena exist; they are freely available; they yield results that are acceptable for research purposes; Ontological Semantics makes no theoretical claims about pre-semantic aspects of language processing; and our approach to semantic analysis does not require pre-semantic heuristics to be complete or perfect. However, using externally developed tools comes at a price: their output must be integrated into the agent's knowledge environment. This is carried out at the stage called *Pre-Semantic Integration*.

This chapter first introduces the tool set LEIAs use for pre-semantic analysis and then describes the many functions needed to mold those results into the most useful heuristics to support semantic analysis. Although the specific examples cited in the narrative apply to a particular tool set at a particular stage of its development, there is a more important generalization at hand: building systems by combining independently developed processors will always require considerable work on integration—a reality that is insufficiently addressed in the literature describing systems that treat individual linguistic phenomena (see section 2.6).

3.1 Pre-Semantic Analysis

For pre-semantic analysis (preprocessing and syntactic parsing), LEIAs currently use the Stanford CoreNLP Natural Language Processing Toolkit (Manning et al., 2014). Although CoreNLP was trained on full-sentence inputs, its results for subsentential fragments are sufficient to support our work on incremental NLU.

For preprocessing, LEIAs use results from the following CoreNLP annotators:

- *ssplit*, which splits texts into sentences;
- *tokenize*, which breaks the input into individual tokens;

- *pos*, which carries out part-of-speech tagging;
- *lemma*, which returns the lemmas for tokens;
- *ner*, which carries out named-entity recognition; and
- *entitymentions*, which provides a list of the mentions identified by named-entity recognition.

Since CoreNLP uses a different inventory of grammatical labels than Ontological Semantics, several types of conversions are necessary, along with a battery of fix-up rules—all of which are too fine-grained for this description. We mention them only to emphasize the overhead that is involved when importing external resources and why it is infeasible to switch between different external resources each time a slight gain in the precision of one or another is reported.

For syntactic analysis, CoreNLP offers both a constituent parse and a dependency parse.[1] A constituent parse is composed of nested constituents in a tree structure, whereas a dependency parse links words according to their syntactic functions. Figures 3.1 and 3.2 show screenshots of the constituent and dependency parses for the sentence *A fox caught a rabbit*, generated by the online tool available at the website corenlp.run.[2]

Consider one example of the difference in information provided by these different parsing strategies. Whereas the constituency parse labels *a fox* and *a rabbit* as noun phrases and places them in their appropriate hierarchical positions in the tree structure, the dependency parse indicates that *fox* is the subject of *caught*, and *rabbit* is its direct object.

Both kinds of parses provide useful information for the upcoming semantic analysis. However, at the current state of the art, the results are error-prone, especially in less-formal speech genres such as dialogs. Therefore, rather than rely on either type of parse wholesale, the NLU system uses parsing output judiciously, as described below.

3.2 Pre-Semantic Integration

The Pre-Semantic Integration module adapts the outputs of preprocessing and parsing to the needs of semantic analysis. The subsections below describe the contentful (not bookkeeping-oriented) procedures developed for this purpose.

3.2.1 Syntactic Mapping: Basic Strategy

Syntactic mapping—or syn-mapping, for short—is the process by which a LEIA matches constituents of input with the syn-struc (syntactic structure) zones of word senses in the lexicon. This process answers the question, Syntactically speaking, what is the best combination of word senses to cover this input? Figure 3.3 illustrates the syn-mapping process for the input *He ate a sandwich*. It shows the relevant excerpts from two senses of *eat* (presented in section 2.2), one of which is syntactically suitable (eat-v1) and the other of which is not (eat-v2).

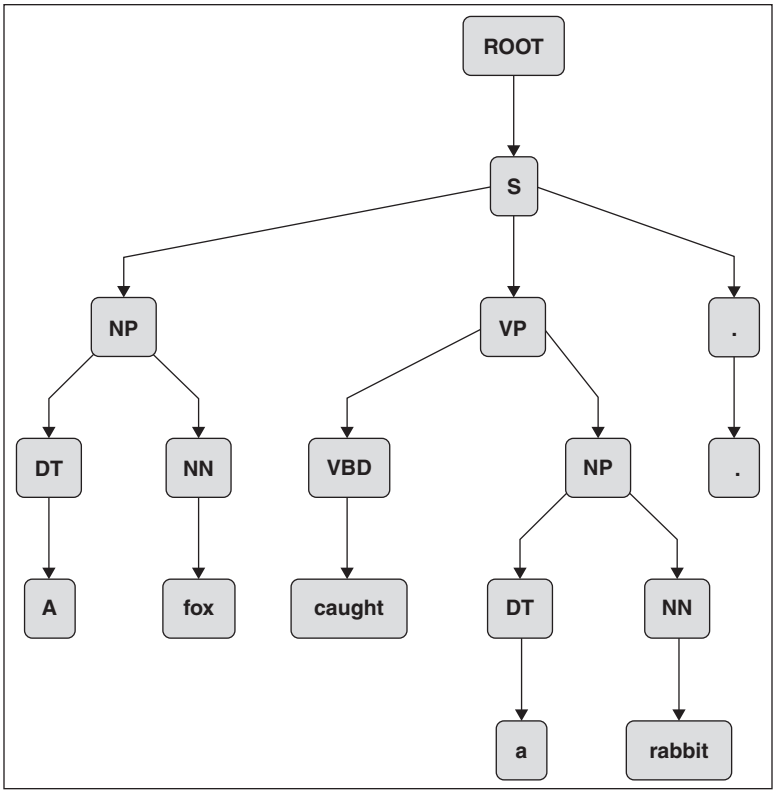


Figure 3.1
The constituency parse for *A fox caught a rabbit*.

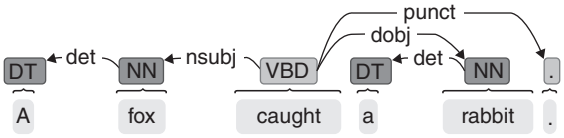


Figure 3.2
The dependency parse for *A fox caught a rabbit*.

Later on, during Basic Semantic Analysis, the LEIA will determine whether the meanings of the variables filling the subject and direct object slots of eat-v1 are appropriate fillers of the AGENT and THEME case roles of INGEST.[3]

Although the syn-mapping process looks easy for an example like *He ate a sandwich*, it gets complicated fast as inputs become more complex. In fact, it often happens that no syn-mapping works perfectly. There are many reasons for this, four of which we cite for illustration.

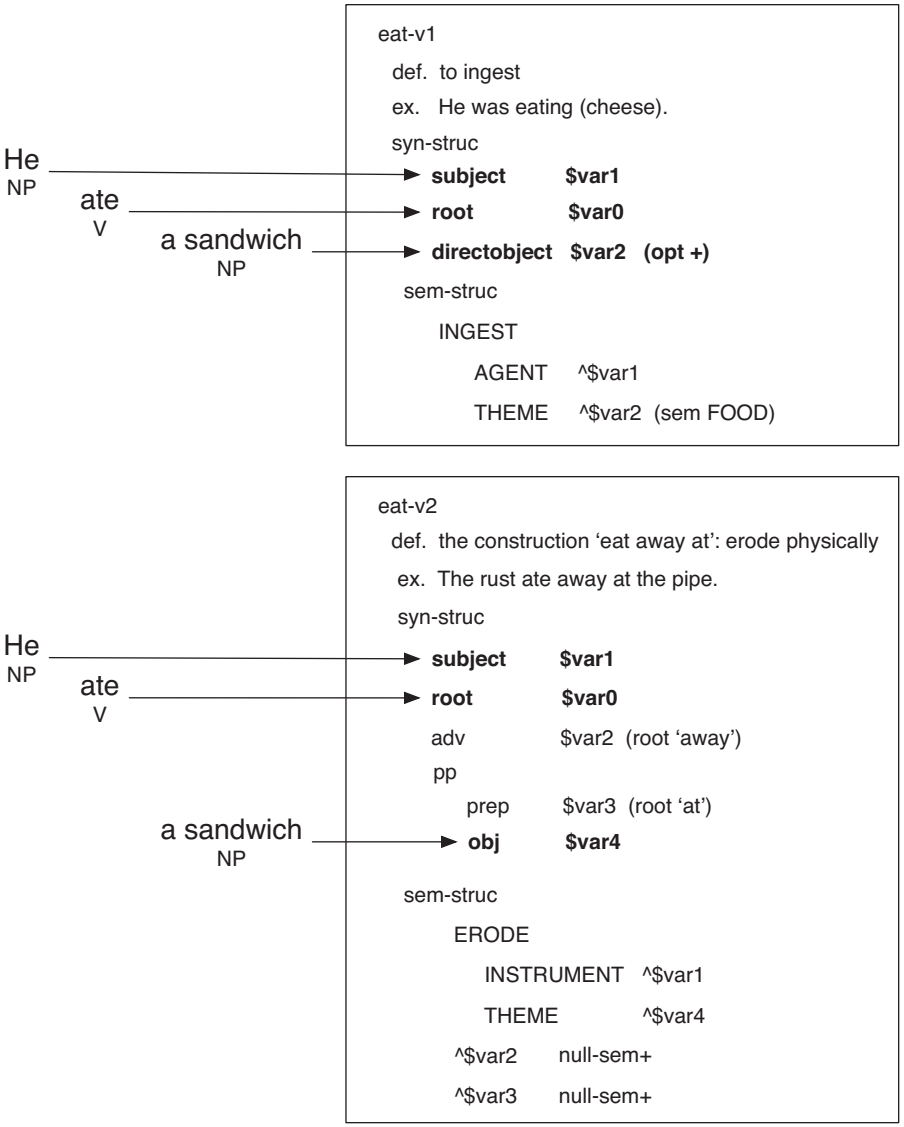


Figure 3.3
A visual representation of syn-mapping. For the input *He ate a sandwich*, eat-v1 is a good match because all syntactic expectations are satisfied by elements of input. Eat-v2 is not a good match because the required words *away* and *at* are not attested in the input.

1. Our syntactic theory does not completely align with that of CoreNLP: for example, our inventories of parts of speech and syntactic constituents are different from those used by CoreNLP. Although we have implemented default conversions, they are not correct in every case.
2. The parser is inconsistent in ways that cannot be anticipated linguistically. For example, the multiword expressions *Right at the light* and *Left at the light*—which, in the context of giving directions, mean *Turn right/left at the next traffic light*—are linguistically parallel, but the parser treats them differently, tagging *right* as an adverb but *left* as a verb. These kinds of inconsistencies are a good example of the challenges that arise when implementing models in systems. After all, the most natural and efficient way to prepare LEIAs to treat such expressions (i.e., the best modeling strategy) is to

    a. record the lexical construction "[direction] at the N", such that 'direction' can be filled by *right*, *left*, *hard right*, *hard left*, *slight right*, *slight left*, and N can indicate any physical object; and

    b. test the construction using a sample sentence to be sure that it is parsed as expected. However, when using a statistically trained parser, a correct parse for one example does not guarantee a correct parse for another structurally identical example. To generalize, any construction that includes variable elements can end up being parsed differently given different actual words of input.

3. The lexicon is incomplete. It can, for example, include one sense of a word requiring one syntactic construction but not another sense requiring a different syntactic construction. The question is, If an input uses a known word in an unexpected syntactic construction, should the system create a fuzzy match with an existing sense—and use that sense's semantic interpretation—or assume that a new sense needs to be learned? The answer: It depends. We illustrate the eventualities using simple examples that artificially impoverish our lexicon:

    a. Let us assume, for the sake of this example, that all of the verbal senses of *hit* in the lexicon are transitive—that is, they require a subject and a direct object. Let us assume further that the input is *He hit me up for 10 bucks*. Although any verb can accommodate an optional prepositional phrase (here: *for 10 bucks*), particles (here: *up*) cannot be freely added to any verb, so fuzzy matching would be the wrong solution. Instead, the agent needs to attempt to learn this new (idiomatic) word sense.

    b. By contrast, let us assume that the only available sense of the verb *try* requires its complement to be a progressive verb form, as in *Sebastian tried learning French*. Assume further that the input contains an infinitival complement: *Sebastian tried to learn French*. In this case, fuzzy matching of the syntactic structures would be correct since the same semantic analysis applies to both.

    So, when is fuzzy matching of syntactic structures appropriate and when isn't it?

Although our examples suggest a couple of rules of thumb (avoid fuzzy matching in the case of unexpected particles; do fuzzy matching given different realizations of verbal complements), the overall problem is larger and more complex.

4. Many inputs are actually noncanonical, reflecting production errors such as repetitions, disfluencies, self-corrections, and the like. These cannot, even in principle, be neatly syn-mapped.

Because of these complications, we have enabled agents to approach syn-mapping in two different ways, each one appropriate for different types of applications.

1. *Require a perfect syn-map*. Under this setting, if there is no perfect syn-map, the agent bypasses the typical syntax-informs-semantics NLU strategy and jumps directly to Situational Reasoning, where it attempts to compute the meaning of the input with minimal reliance on syntactic features (see section 7.2). This strategy is appropriate, for example, in applications involving informal, task-oriented dialogs because (a) they can contain extensive fragmentary utterances, and (b) the agent should have enough domain knowledge to make computing semantics with minimal syntax feasible.
2. *Optimize available syn-maps*. Under this setting, the agent must generate one or more syn-maps, no matter how far the parse diverges from the expectations recorded in the lexicon. These syn-maps feed the canonical syntax-informs-semantics approach to NLU. Optimizing imperfect syn-mapping is appropriate, for example, (a) in applications that operate over unrestricted corpora (since open-corpus applications cannot, in the current state of the art, expect full and perfect analysis of every sentence), (b) in applications for which confidently analyzing subsentential chunks of input can be sufficient (e.g., new word senses can be learned from cleanly parsed individual clauses, even if the full sentence containing them involves parsing irregularities), and (c) in lower-risk applications where the agent is expected to just do the best it can.

The processing flow involving syn-mapping is shown in figure 3.4.

Syn-mapping can work out perfectly even for subsentential fragments as long as they are valid beginnings of what might result in a canonical structure. Obviously, this is an important aspect of modeling incremental language understanding. For example, the inputs "The rust is eating" and "The rust is eating away" are both unfinished, but they will map perfectly to the syntactic expectations of eat-v2 presented earlier.

We already described how syn-mapping proceeds when everything works out well—that is, when the input aligns with the syntactic expectations recorded in the lexicon. Now we turn to cases in which it doesn't. Specifically (cf. figure 3.4), we will describe (a) the two recovery methods that attempt to normalize imperfect syn-maps and (b) the process of optimizing imperfect syn-maps when the input cannot be normalized.

One strategic detail is worth mentioning. When syn-mapping does not work perfectly, the agent waits until the end of the sentence to attempt recovery. That is, it does not attempt

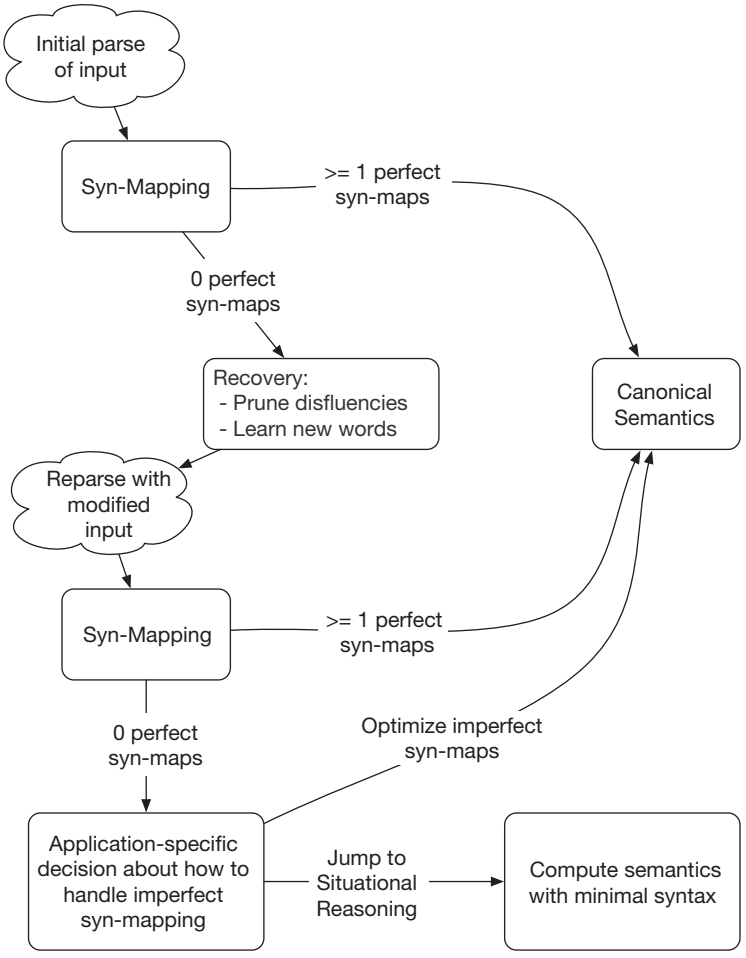


Figure 3.4
The processing flow involving syn-mapping. If the initial parse generates at least one perfect syn-map, then the agent proceeds along the normal course of analysis (stages 3–6: Basic Semantic Analysis, Basic Coreference Resolution, Extended Semantic Analysis, and Situational Reasoning). If it does not, then two recovery strategies are attempted, followed by reparsing. If the new parse is perfect, then the agent proceeds normally (stages 3–6). By contrast, if the new parse is also imperfect, the agent decides whether to optimize the available syn-maps and proceed normally (stages 3–6) or skip stages 3–5 and jump directly to stage 6, Situational Reasoning, where computing semantics with minimal syntax will be attempted.

recovery on sentence fragments during incremental analysis. This not only is a computationally expedient solution (the recovery programs need as much information as they can get) but also makes sense in terms of cognitive modeling, as it is unlikely that the high cognitive load of trying to reconstruct meaning out of nonnormative, subsentential inputs will be worthwhile.

3.2.2 Recovering from Production Errors

Noncanonical syntax—reflecting such things as disfluencies, repetitions, unfinished thoughts, and self-corrections—is remarkably common in unedited speech.[4] Even more remarkable is the fact that people, mercifully, tend not to even notice such lapses unless they look at written transcripts of informal dialogs. Consider, for example, the following excerpt from the Santa Barbara Corpus of Spoken American English, in which a student of equine science is talking about blacksmithing while engaged in some associated physical activity.

> we did a lot of stuff with the—like we had the, um, … the burners? you know, and you'd put the—you'd have—you started out with the straight … iron? … you know? and you'd stick it into the, … into the, … you know like, actual blacksmithing (DuBois et al. 2000–2005).[5]

Outside of context, and unsupported by the intonation of spoken language, this excerpt requires a lot of effort to understand.[6] Presumably, we get the gist by partially matching elements of input against the expectations in our mental grammar, lexicon, and ontology (i.e., we told you this was about blacksmithing).

The first method the LEIA uses to recover from these lapses is to strip the input of disfluencies (e.g., *um*, *uh*, *er*, *hmm*) and precisely repeated strings (e.g., *into the, ... into the*) and then attempt to parse the amended input. The stripping done at this stage only addresses the simplest, highest-confidence cases. The need for stripping is nicely illustrated by examples from the TRAINS corpus (Allen & Heeman, 1995).

(3.1) um so let's see where are there oranges (TRAINS)

(3.2) there's let me let me summarize this and make sure I did everything correctly (TRAINS)

If the new, stripped input results in a parse that can be successfully syntactically mapped, recovery was successful.

A research question is, Can more stripping methods be reliably carried out at this stage, or would this risk inadvertently removing meaningful elements before semantics had its say? For now, more sophisticated stripping methods are postponed until Situational Reasoning (chapter 7).

3.2.3 Learning New Words and Word Senses

The LEIA's lexicon currently contains about 30,000 word senses, which makes it sufficient for validating our microtheories but far from comprehensive. This means that LEIAs

must be able to process both unknown words and unknown senses of known words. For example, if the lexicon happens to lack the word *grapefruit*, then the agent will have to undertake new-word learning when analyzing the seemingly mundane *I ate a grapefruit for breakfast*. The results of its learning will be similar to what many readers would conclude if faced with the input *Paul ate some cupuacu for breakfast this morning*: it must be some sort of food but it's unclear exactly what kind. (It is a fruit that grows wild in the Amazon rain forest.) The frequent need for new-word learning is actually beneficial for our program of work in developing LEIAs. After all, the holy grail of NLU is for agents to engage in lifelong learning of lexical, ontological, and episodic knowledge, so the more practice LEIAs get, and the more we troubleshoot the challenges they face, the better.

Since at this stage of the processing the agent is focusing exclusively on syntax, the only operation it carries out to handle unknown words is to posit a new template-like lexical sense that allows for syn-mapping to proceed in the normal way. To enable this, we have created templates for each typical dependency structure for open-class parts of speech. For example, the template for unknown transitive verbs is as follows. It will be used if the system encounters an input like *Esmeralda snarfed a hamburger*, which includes the colloquial verb *snarf*, meaning 'to eat greedily.'

```
new-transitive-verb-v1
    syn-struc
        subject              $var1
        v                    $var0
        directobject         $var2
    sem-struc
        EVENT-#1
            CASE-ROLE        ^$var1
            CASE-ROLE        ^$var2
    meaning-procedures
        seek-specification EVENT-#1
```

Note that although the syntactic description is precise (exactly the same as for known transitive verb senses), the semantic description is maximally generic: an unspecified EVENT is supplied with two unspecified CASE-ROLE slots to accommodate the meanings of the attested arguments. (Note that the EVENT is indicated by EVENT-#1 in order to establish the necessary coreference between its use in the sem-struc and meaning-procedures zones.) Later on, during Basic Semantic Analysis (and, sometimes, Situational Reasoning), the agent will attempt to enhance the nascent lexicon entry by using the meanings of those case role fillers to (a) narrow down the meaning of the EVENT and (b) determine which case roles are appropriate. This future processing is put on agenda using the meaning-procedures zone, which contains a call to the procedural semantic routine called seek-specification, whose argument is the underspecified EVENT.

Whereas there is a single template for unknown transitive verbs, there are three templates for unknown nouns, since they can refer to an OBJECT (*meerkat*), EVENT (*hullabaloo*), or PROPERTY (*raunchiness*). The agent generates all three candidate analyses at this

stage and waits until later—namely, Basic Semantic Analysis—to not only discard the unnecessary two but also attempt to narrow down the meaning of the selected one.[7] The following is the new-noun template mapping to OBJECT.

```
new-noun-object-n1
    comments     This sense covers the eventuality that the noun refers to an OBJECT.
    syn-struc
        root $var0
    sem-struc
        OBJECT-#1
    meaning-procedures
        seek-specification OBJECT-#1
```

As for adjectives, such as *fidgety*, they are learned using the following template:

```
new-adj-adj1
    syn-struc
        mod     $var0
        n       $var1
    sem-struc
        PROPERTY-#1
            DOMAIN     ^$var1
    meaning-procedures
        seek-specification PROPERTY-#1
```

To reiterate, at this stage, the agent learns the syntax of new words and prepares for learning the associated semantics later on.

3.2.4 Optimizing Imperfect Syn-Maps

If, after attempting to normalize imperfect syn-maps (figure 3.4), there is still no perfect syn-map, the agent needs to make an application-oriented decision about whether to optimize imperfect syn-maps (essentially, do the best it can to push through the normal flow of processing) or skip to stage 6, where it can attempt to compute semantics with minimal syntax. Here we describe the default behavior: optimizing the imperfect syn-maps.

When the agent chooses to optimize imperfect syn-maps, it (a) generates all possible binding sets (i.e., mappings from elements of input to variables in lexical senses), (b) prioritizes them in a way that reflects their syntactic suitability, and (c) removes the ones that are highly implausible. This leaves a reasonable-sized subset of candidate mappings for the semantic analyzer to consider.[8] The need for this process is best illustrated by an example:

(3.3) Cake—no, *chocolate* cake—I'd eat every day.

CoreNLP generates an underspecified dependency parse for this input (see the online appendix at https://homepages.hass.rpi.edu/mcsham2/Linguistics-for-the-Age-of-AI.html for a screen shot of its output). Although the parse asserts that *I* is the subject of *eat* and

every day is its modifier, it does not capture that *chocolate cake* is the direct object of *eat*. This leaves the syn-mapper little to go on in determining how to fill the case role slots of the available lexical senses of *eat* using all the available syntactic constituents.

The syn-mapper's approach to solving this problem is to assume, from the outset, that any non-verbal constituent can (a) fill any argument slot of the most proximate verb or (b) not fill any argument slot at all.[9] (Note that the latter is important in our example: the first instance of *cake* and the word *no* actually do not fill slots of *eat*.) The idea is to generate all candidate binding sets and then prune out the ones that seem too implausible to pass on to the semantic analyzer. Table 3.1 shows a small subset of the available binding sets if eat-v1 (the INGEST sense) is being considered as the analysis of *eat* in our sentence. Recall that eat-v1 is optionally transitive, which means that it does not require a direct object.

As a human, you might think that it makes no sense to even consider *every day* as the subject or direct object of *eat*, and that it makes no sense to leave both *chocolate cake* and *cake* unbound when they so obviously play a role in the eating being described. But making sense involves semantic analysis, and we haven't gotten there yet! What this *syntactic* analysis process sees is "NP—ADV, NP—NP AUX V NP," along with the parser's best guesses as to syntactic constituency and dependencies, but, as we have explained, its reliability decreases as input become more complex and/or less canonical.

The syn-mapper's algorithm for preferring some binding sets to others, and for establishing the plausibility cutoff for passing candidate binding sets to the semantic analyzer, involves a large inventory of considerations, most of which are too detailed for this exposition. But a few examples will illustrate the point.

Table 3.1
This is a subset of the binding sets that use *eat-v1* to analyze the input *Cake—no, chocolate cake—I'd eat every day.* The ellipses in the last row indicate that many more binding sets are actually generated, including even a set that leaves everything unbound, since this computational approach involves generating every possibility and then discarding all but the highest-scoring ones.

Subject	(Direct Object)	Unbound	Unbound	Unbound
Cake	chocolate cake	every day	I	
chocolate cake	Cake	every day	I	
I	Cake	every day	chocolate cake	
I	chocolate cake	every day	cake	
I	every day	chocolate cake	cake	
I	-	chocolate cake	cake	every day
chocolate cake	-	I	cake	every day
every day	I	chocolate cake	cake	
…	…	…	…	…

- Lexical senses for multiword expressions require particular lexemes; if those lexemes are not in the input, then the multiword senses are excluded. For example, the idiomatic sense of *eat* that covers *eat away at* (ERODE) will not be used to analyze the input *He ate a sandwich*.[10]
- If a lexical sense is mandatorily transitive, but the input has no direct object, then the given sense is strongly penalized. For example, the transitive sense of *walk* that is intended for contexts like *Miranda is walking the dog* will not be used to analyze inputs like *Miranda walks in the evenings*.
- If the lexical sense expects its internal argument to be an NP but the input contains a verbal complement, then that sense is strongly penalized. For example, the transitive sense of *like* that is intended for inputs such as *I like ice cream* will not be used for inputs such as *I like to ski*.
- In imperative clauses, the subject argument must not be bound. For example, when analyzing *Eat the cookies!* the syn-mapper will exclude the candidate set in which *the cookies* is used to fill the subject slot of the lexical sense eat-v1.

3.2.5 Reambiguating Certain Syntactic Decisions

No matter how the syn-mapping process proceeds—whether or not it involves recovery procedures, whether or not it generates perfect syn-maps—certain additional parse-modification procedures need to be carried out. This is because syntactic parsers are usually engineered to prefer yielding one result. However, they are not suited to making certain decisions in principle because the disambiguating heuristics are semantic in nature. Three syntactic phenomena that require parsers to make semantics-oriented guesses are prepositional phrase (PP) attachments, nominal compounds, and phrasal verbs.

- ***PP attachments.*** When a PP immediately follows a post-verbal NP, it can modify either the verb or the adjacent NP. A famous example is *I saw the man with the binoculars*.
  - If the binoculars are the instrument of seeing (they are used to see better), then the PP attaches to the verb: I [$_{VP}$ saw [$_{NP}$ the man] [$_{PP}$ with the binoculars]].
  - If the binoculars are associated with the man (he is holding or using them), then the PP attaches to the NP: I [$_{VP}$ saw [$_{NP}$ the man [$_{PP}$ with the binoculars]]].
- ***Nominal compounds.*** Nominal compounds containing more than two nouns have an internal structure that cannot be predicted syntactically; it requires semantic analysis. Compare:
  - [[kitchen floor] cleanser]
  - [kitchen [floor lamp]]

- ***Phrasal verbs.*** In English, many prepositions are homographous with (i.e., have the same spelling as) verbal particles. Consider the collocation *go after* + NP, which can have two different syntactic analyses associated with two different meanings:
  - [verb + particle + direct object] has the idiomatic meaning 'pursue, chase': The cops went_{V} $\text{after}_{\text{PARTICLE}}$ the $\text{criminal}_{\text{DIRECT-OBJECT}}$.
  - [verb + preposition + object of preposition] has the compositional meaning 'do some activity after somebody else finishes their activity': The bassoonist went_{V} $\text{after}_{\text{PREP}}$ the $\text{cellist}_{\text{OBJECT-OF-PREP}}$.

While there are clearly two syntactic analyses of *go after* that are associated with different meanings, and while there is often a default reading depending on the subject and object selected, it is impossible to confidently select one or the other interpretation outside of context. After all, *The cops went after the criminal* could mean that the cops provided testimony after the criminal finished doing so, and *The bassoonist went after the cellist* could mean that the former attacked the latter for having stepped on his last reed.

For all of these, LEIAs reambiguate the parse. That is, they always, as an across-the-board rule, create multiple candidates from the single one returned by the parser. Selecting among them is the job of the semantic analyzer at the next stage of analysis.

3.2.6 Handling Known Types of Parsing Errors

This book concentrates primarily on ideas—our theory of NLU, the rationale behind it, and how systems that implement it support the operation of intelligent agents. These ideas could be implemented using a wide range of engineering decisions, which are not without interest, and we have devoted significant effort to them. However, had we decided to discuss them in detail, this would have doubled the length of this book. Still, we will mention select engineering issues and solutions to emphasize that engineering must be a central concern for computational linguistics, being at the heart of the model-to-system transition (see section 2.6).

The engineering solution we consider here involves parsing errors. As mentioned earlier, syntactic parsing is far from a solved problem, so parsing errors are inevitable, even for inputs that are linguistically canonical. For example, our lexicon includes a ditransitive sense of *teach* intended to cover inputs like *Gina taught George math*. However, the parser we use incorrectly analyzes *George math* as a nominal compound.[11] How did we detect this error? Manually, as a part of testing and debugging. (The agent cannot independently recognize this particular error because the parse actually works out syntactically, the input being structurally parallel to *Gina taught social studies*, which does include a nominal compound and aligns with the transitive sense of *teach* in the lexicon.)

Rather than go down the rabbit hole of creating fix-up rules for the parser, we do the following: If inputs with the given structure are not crucial for a current application, we

ignore the error and allow the associated inputs to be incorrectly analyzed. The agent then treats them as best it can despite the error. If, by contrast, such inputs *are* crucial for a current application—for example, if they must be featured in a robotic system demo tomorrow—then we use a recovery strategy that works as follows. We invent a sample sentence, run it through the parser, record its actual syntactic analysis, and then manually provide the necessary linking between syntactic and semantic variables. All of this information is stored in an adjunct database that does not corrupt the original lexicon.

Let us work through our *teach* example by way of illustration. Below is the canonical lexical sense of ditransitive *teach*, teach-v1.

```
teach-v1
    def.          to teach someone some subject matter
    ex.           Mary taught John physics.
    syn-struc
        subject           $var1
        v                 $var0
        indirectobject    $var3
        directobject      $var2
    sem-struc
        TEACH
            AGENT         ^$var1
            THEME         ^$var2
            BENEFICIARY   ^$var3
```

Compare this with the supplementary sense that accommodates the parsing error, teach-v101. This sense includes two traces that it is not canonical: it uses a special sense-numbering convention (100+), and it includes an associated comment in the comments field.

```
teach-v101
    def.          to teach someone some subject matter
    ex.           Mary taught John physics.
    comments      accommodates a parsing error
    syn-struc
        subject           $var1
        v                 $var0
        np
            n             $var2
            n             $var3
    sem-struc
        TEACH
            AGENT         ^$var1
            THEME         ^$var3
            BENEFICIARY   ^$var2
```

3.2.7 From Recovery Algorithm to Engineering Strategy

The recovery algorithm just described morphed into an available—although not default—engineering practice for adding new construction senses to the lexicon. This practice is used when acquirers either suspect or have evidence that a construction will not be treated by the parser in the way anticipated by our linguistic theory. The rationale for this practice is best explained by tracing what lexicon acquirers and programmers each want.

Lexicon acquirers want to record senses fast, with as few constraints on their expressive power as possible. They don't want to worry about the quirks of actual processors—or, more formally, about model-to-system misalignments. Programmers, for their part, want the available processors to output what the knowledge engineers are expecting. As it turns out, all of these desiderata can be met thanks to a program we developed for this purpose: the ExampleBindingInterpreter. The ExampleBindingInterpreter requires two types of input:

1. a lexical sense whose syn-struc is underspecified: it contains an ordered inventory of fixed and variable components, but the acquirer need not commit to the constituents' parts of speech or their internal structure; and
2. a sample sentence that indicates which words align with which syntactic components.

The ExampleBindingInterpreter does the rest. It creates a syn-map, no matter the actual parser output, allowing for subsequent semantic analysis to proceed in the normal way. We will illustrate the method using an example from an autonomous vehicle application.

The input is the command from the user to the agent "Turn right at the light," which is so frequent in this particular application that it merits being recorded explicitly in the lexicon. The semantic description of this input

a. includes the REQUEST-ACTION conveyed by the imperative verb form;
b. disambiguates the verb *turn* (which can also mean, e.g., 'rotate');
c. disambiguates the word *right* (which can also mean, e.g., 'correct');
d. disambiguates the word *at* (which can also, e.g., indicate a time);
e. concretizes the meaning of *the* as 'the next one' (we do not want the system to use general coreference procedures to try to identify an antecedent for *light*); and
f. disambiguates the word *light* (which can also mean, e.g., 'lamp').

If we were to record the syn-struc for this multiword expression in the usual way, it would look as follows.

```
turn-v12
    def.        The expression 'Turn right at the light.'
    ex.         Turn right at the light.
    syn-struc
        v           $var0 (form infinitive)
```

```
        adv         $var1 (root 'right')
        pp
            prep    $var2 (root 'at')
            obj     $var3 (root 'light')
    sem-struc
        …
```

But consider all the mismatches that might occur during parsing: The parser might consider *right* an adjective or a verb rather than adverb; it might attach the PP to *right* rather than to *turn*; and this formalism does not readily allow for the explicit inclusion of the word *the*, which we actually want to treat specially in the sem-struc by blocking generic coreference procedures. Now compare this with the underspecified syn-struc in the lexicon entry shown in turn-v101 (which includes the sem-struc as well). The variables x, y, and z allow for the parser to tag the given words with any parts of speech.

```
turn-v101
    def.            The expression 'Turn right at the light.'
    ex.             Turn right at the light.
    comments        Syntactically underspecified; uses ExampleBindingInterpreter.
    syn-struc
        use-example-binding t
        v           $var0
        x           $var1 (root 'right')
        prep        $var2 (root 'at')
        y           $var3 (root 'the')
        z           $var4 (root 'light')
    sem-struc
        REQUEST-ACTION
            AGENT           HUMAN-#1 ("speaker")
            BENEFICIARY     HUMAN-#2 ("hearer")
            THEME           refsem1
        refsem1
            TURN-VEHICLE-RIGHT
                AGENT       HUMAN-#2 ("hearer")
                LOCATION    NEXT-TRAFFIC-LIGHT
        ^$var1      null-sem+
        ^$var2      null-sem+
        ^$var3      null-sem+
        ^$var4      null-sem+
    example-bindings turn-0 right-1 at-2 the-3 light-4
```

Let us work through the above entry from top to bottom.

- The fact that automatic variable binding will occur is indicated by "use-example-binding t" in the syn-struc ('t' means 'true').
- In writing this syn-struc, the acquirer needs to commit to only one part of speech: the one for the headword ($var0). All other parts of speech can be either asserted (if the

acquirer is confident of a correct analysis) or indicated by variables. In this example, the part of speech for *at* (prep) is asserted and the rest are left as variables.

- The actual root word for each category can be listed or left open: e.g., the actual noun (*light*) could have been left out, allowing the expression to cover any input matching "Turn right at the N."
- Optional elements can be indicated in the usual way: (opt +).
- Variations on an element can also be indicated. For example, the two expressions "Turn right at the light" and "Turn left at the light" can be covered by changing the description of X to (root 'right' 'left') as long as the parser assigns the same part of speech to all listed variations. (Recall that for the elliptical *Right/left at the light*, 'right' and 'left' were assigned different parts of speech, which would make sense bunching impossible in this case.)
- As described earlier, the sem-struc
  - asserts that this is a REQUEST-ACTION;
  - includes disambiguation decisions for all component words by indicating the concepts that describe their meaning; and
  - uses those concepts to fill case roles slots.
- The sem-struc indicates the roles of the speaker and hearer, which must be grounded in the application. We will use the convention "HUMAN-#1 ("speaker")" and "HUMAN-#2 ("hearer")" throughout as a shorthand to indicate the necessary grounding.
- The concepts TURN-VEHICLE-RIGHT and NEXT-TRAFFIC-LIGHT are, like all concepts, described by properties in the ontology—they are not vacuous labels in upper-case semantics. The reason they were promoted to the status of concepts is that, within our driving script—which was acquired to support a particular application—they play a central role. It is, therefore, more efficient to encapsulate them as concepts rather than to compositionally compute the elements of the expression on the fly every time. (Cf. the discussion of *eating hot liquids with a spoon* in section 2.8.2.)
- The null-semming of the variables reflects that their meanings have already been incorporated into the sem-struc. (For example, ^$var1, which corresponds to the word *right*, is null-semmed because its interpretation is folded into the choice of the concept TURN-VEHICLE-RIGHT. The other variables are null-semmed for analogous reasons.) If these variables were not null-semmed, then the analysis system—based on its global processing algorithm—would try to compositionally incorporate all available meanings of these words into the TMR, even though their meanings are already fully taken care of by the description in the sem-struc.
- The example-bindings field contains the sample sentence to be parsed, whose words are appended with the associated variable numbers from the syn-struc.

The reason for presenting this automatic syn-mapping process in such detail is to underscore that it addresses two core needs of NLU: (1) the need to populate the lexicon with constructions, since so much of language analysis is not purely compositional, and (2) the need to proactively manage the inevitable mismatches between idealized models and the actual results of actual processors that are available to be used in systems.

You might wonder: Aren't we losing something in terms of cognitive modeling by not recording the canonical linguistic structure of constructions like these? Yes. We are sacrificing modeling desiderata in service of making a particular system, which uses a particular parser, actually work. This is a trade-off. We are certainly not recommending that underspecifying the parts of speech should be a blanket answer to recording knowledge about constructions. Instead, it should be used judiciously, like all tools in the system-building toolbox.

3.3 Managing Combinatorial Complexity

Unfortunately, syn-mapping can result in many candidates for the semantic analyzer to work through. Because agents must function in real time, we need to address this problem of combinatorial explosions head-on, which we do with the microtheory of combinatorial complexity, to which we now turn.

The first thing to say about this microtheory is that it anticipates and attempts to circumvent the consequences of combinatorial complexity at the interface between syntactic and semantic analysis. That is, we can foresee which kinds of lexical items will predictably spawn combinatorial complexity, and we can reduce that complexity using specific types of knowledge engineering. Since there is no dedicated processing module corresponding to the syntax-semantics interface, architecturally, this microtheory best resides in Pre-Semantic Integration.

Combinatorial complexity arises because most words have multiple senses. If a sentence contains 10 words, each of which has 3 senses, then the agent must consider $3^{10} = 59{,}049$ candidate analyses. Since, in our lexicon, prepositions have many senses each, and common light verbs (such as *have*, *do*, *make*) have several dozen senses each, this means that even midlength sentences that contain even one preposition or light verb can quickly run into the tens of thousands of candidate analyses.

Consider the example in figure 3.5, which, although obviously cooked, is nicely illustrative: *Pirates attack animals with chairs in swamps*. (In a cartoon, this might even make sense.) If we consider even just two senses of each word, there will be 128 candidate interpretations.[12]

Not only does this example offer 128 candidate interpretations, but the semantic constraints available during Basic Semantic Analysis will be able to weed out only some of the interpretations. Many will remain equally viable, meaning that there will be extensive residual ambiguity. For example, both senses of *pirate* (pirate at sea and intellectual

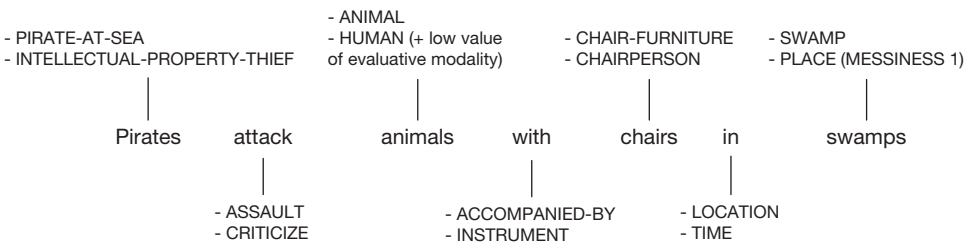


Figure 3.5
A subset of paired, syntactically identical senses.

property thief) are equally suitable as AGENTs of both senses of *attack* (physically attack and verbally assault); both senses of *swamp* (a bog and a messy place) are equally suitable as fillers of the LOCATION interpretation of *in*; both senses of *animal* (a living creature and a human viewed negatively) can be the THEME of the ASSAULT meaning of *attack*; and *with* can indicate the instrument the pirates use (chairs for sitting) or people accompanying the pirates in their actions (chairpersons).

Many instances of ambiguity are expected to remain unresolved at the stage of Basic Semantic Analysis, before more sophisticated reasoning has been leveraged. However, an interesting question arises at the interface of cognitive modeling and system engineering: Could candidate interpretations be bunched in a way that is both psychologically plausible and practically useful, in order to better manage the search space for the optimal analysis? The answer is yes, and it could be approached either through knowledge engineering or through dynamic reasoning.[13]

The knowledge-engineering solution (which we do not pursue) involves developing a hierarchical lexicon. Since both kinds of pirates are thieves, they could share the mapping THIEF, which could be used by default if the context failed to make clear which one was intended. Similarly, since both kinds of attack indicate a type of conflict, they could share the mapping CONFLICT-EVENT, which would correspondingly be used by default. There is much to like about this approach to knowledge engineering, not least of which is that it jibes with our intuitive knowledge of these word meanings. However, is developing a hierarchical lexicon—which is conceptually heavier and more time-consuming than developing a flat lexicon—really the best use of acquirer time, given that (a) many other types of knowledge are waiting to be acquired and (b) the underlying ontology is already hierarchical? We think not, which leads us to the more promising solution that involves runtime reasoning.

We have developed computational routines for dynamic sense bunching, with the following being the most useful so far:

- Bunching productive (i.e., not phrasal) prepositional senses into a generic RELATION.
- Bunching verb senses with identical syn-strucs and identical semantic constraints into their most local common ancestor. For example, *turn the steak* can mean INVERT (flip)

or ROTATE (let a different portion be over the hottest part of the grill), which have the common ancestor CHANGE-POSITION. Similarly, *distribute the seeds* can mean SPREAD-OUT or DISTRIBUTE (as to multiple people who bought them), whose common ancestor is CHANGE-LOCATION. In cases of literal and metaphorical sense pairs (e.g., *attack*), the common ancestor can be as imprecise as EVENT; however, even this is useful since it is a clue that the ambiguity could involve metaphorical usage.

- Bunching noun senses that refer to ANIMALS or HUMANS. For example, *pig* can refer to a barnyard animal, a messy person, or any animal who overeats (the latter two are described by multiconcept sem-strucs headed by HUMAN and ANIMAL, respectively).
- Bunching different senses of PHYSICAL-OBJECTS. These include, for example, the MACHINE and COMPUTER meanings of *machine* and the AUTOMOBILE and TRAIN-CAR meanings of *car*.

When senses are dynamically bunched, a procedural semantic routine is attached to the umbrella sense and recorded in the TMR. This offers the agent the option to attempt full disambiguation at a later stage of analysis.

Let us take as an example the preposition *in*. Below is one of the syntactically typical (nonphrasal) senses, followed by a list of some of the other syntactically identical senses.

```
in-prep1
    def.          refers to the physical location of an object or event
    ex.           The cat in the study is sleeping (the pp modifies 'cat').
                  The cat is sleeping in the study (the pp modifies 'sleeping').
    syn-struc
        root      $var1 (cat (n v))
        pp
            prep  $var0
            obj   $var2
    sem-struc
        ^$var1
            LOCATION    ^$var2
```

Syntactically similar senses:

- in-prep14: DURING (*In the interview he said …*)
- in-prep17: TIME (*a meeting in January*)

Compare this with the umbrella sense below that bunches them. It links the meanings of $var1 and $var2 using the generic RELATION and includes a meaning procedure (seek-specification RELATION [in-prep1/14/17]) that points to the senses that can be consulted later for full disambiguation.

```
in-prep-501[14]
    def.          the umbrella sense for several senses of 'in'
    syn-struc
        root      $var1 (cat (n v))
        pp
            prep    $var0
            obj     $var2
    sem-struc
        ^$var1
            RELATION    ^$var2
    meaning-procedures
        seek-specification RELATION [in-prep1/14/17]
```

In many cases, the semantic analyzer will be able to select a winning sense. For example, *a meeting in January* will be analyzed confidently using in-prep17 since that sense requires the object of the preposition to refer to a temporal expression, such as MONTH, YEAR, or CENTURY. In such cases, sense bunching will clearly not be the final solution. However, in many cases it will be useful—for example, when speakers use prepositions noncanonically, which is a common type of performance error (see section 6.2.2), or when a multiword expression that would ideally be recorded as a lexical sense (e.g., *in good faith*) has not yet been acquired, thus necessitating a less precise analysis.

To give just one example of how much easier it is for people to read bunched outputs, consider the TMR for the sentence *A pirate was attacked by a security guard* in which the available analyses of *pirate* and *attack* are bunched (and *security guard* is unambiguous).

```
PHYSICAL-EVENT-1
    AGENT           SECURITY-GUARD-1
    THEME           HUMAN-1
    TIME            <find-anchor-time
    BUNCHED-FROM    ATTACK (attack-v1), CRITICIZE (attack-v2)

HUMAN-1
    BUNCHED-FROM    PIRATE-AT-SEA (pirate-n1),
                    INTELLECTUAL-PROPERTY-THIEF (pirate-n2)
```

The candidates this structure covers are as follows, in plain English:

- A pirate at sea was physically attacked by a security guard.
- A pirate at sea was yelled at by a security guard.
- An intellectual property thief was physically attacked by a security guard.
- An intellectual property thief was yelled at by a security guard.

Sense bunching can be applied in many ways: all available types of sense bunching can be carried out prior to runtime and employed across the board; select types of sense

bunching (e.g., prepositions only) can be applied prior to runtime; or the agent can dynamically decide whether or not to bunch based on factors such as the number of candidate TMRs being too large or the extent to which the candidate TMRs do or do not fall within the agent's scope of interest. The actual strategy selected will depend on the requirements of the application system.

Of course, dynamic sense bunching is not the only way to deal with combinatorial complexity. In a particular application, the agent can opt to prefer domain-relevant interpretations from the outset, thereby reducing or even completely removing the problem of lexical disambiguation. (This is, in fact, what many developers of robotic systems routinely do, as this meets short-term goals.) We describe why we chose not to do this in the general case in chapter 7. Another option is to label a subset of senses as preferred, prototypical ones. But although this might seem like an easy type of knowledge acquisition at first glance, it quickly becomes complicated once we move beyond the relatively small set of simplest cases like *dog* defaulting to a canine companion. Ask people whether *cat* means a domesticated feline or a wild one, and the debate will be on! Moreover, even if we recorded knowledge to deal with most eventualities, there would still be residual ones, and one of the foci of our scientific investigation of NLU is to determine how we can best prepare a LEIA to deal with inputs that inevitably combine known and unknown information.

To conclude, although the repercussions of combinatorial complexity will not be encountered until later stages of processing, sense bunching can be incorporated into Pre-Semantic Integration to avoid at least some of those problems.

3.4 Taking Stock

This chapter has described the benefits and challenges of importing resources to carry out the pre-semantic stages of NLU; methods of preparing pre-semantic heuristics to best serve upcoming semantic analysis; the first stage of learning unknown words; and the process of dynamic sense bunching for dealing with combinatorial complexity.

In considering how much is involved in what we call Pre-Semantic Integration, one might ask, Why did we import external processors to begin with rather than developing our own? In fact, in the early days of Ontological Semantics, we *did* develop our own preprocessor along with a lexicalized parser that used a just-in-time parsing strategy (Beale et al., 2003). Although these processors were ideal for what they covered, they did not cover as many phenomena as the statistical preprocessors and parsers that were becoming available at the time. So we made the leap to import externally developed tools and invested extensive resources into integration. It was only once we had carried out the integration that we could assess how well the tools served our needs. As it turned out, there was a mixed bag of costs and benefits.

The original motivation for importing the tools was to save engineering time on pre-semantic issues. However, we did not foresee how much continued engineering effort

would be needed (a) for integration (each new and improved version of the tool set can send ripples throughout our system) and (b) for developing methods to recover from unexpected results. In hindsight, it is unclear whether importing the tool set fostered or impeded our work on semantics and pragmatics. However, a clearly positive aspect of this decision is that it shows that we practice what we preach about science and engineering in AI: that systems need to actually work, and that no single group of individuals can solve the whole problem, so some sort of integration by different teams is ultimately inevitable. Consequently, developers must not shy away from making strategic decisions under uncertainty and incorporating the outcomes, whatever they may be, back into the overall program of R&D. In the case we describe, this has meant spending more time on recovering from unexpected syntactic parses than we could have anticipated a decade ago. But this led us down the path of paying particular attention to unexpected inputs overall, which is entirely to the good. We believe that herein lies a useful lesson for all practitioners in our field.

3.5 Further Exploration

1. Get acquainted with the Stanford CoreNLP parser using the online interface available at the website corenlp.run. To show the results of more than just the default annotators, click on the "Annotations" text field and select more options from the pull-down menu: for example, lemmas, coreference. In addition to grammatical sentences, try sentences that include production errors, such as repetitions (*Put the lamp on the on the table*) and highly colloquial ellipses (*Come on—that, over here, now!*). Even though utterances like these—and many more types of noncanonical formulations—are very common in real language use, they pose challenges to current parsing technologies.

2. Practice drawing parse trees using an online tree-drawing tool, such as the one at http://ironcreek.net/syntaxtree/. This will be useful because many aspects of the upcoming discussions assume that readers at least roughly understand the syntactic structure of sentences. If you need an introduction to, or refresher about, parse trees, you can look online (e.g., Wikipedia) or consult a textbook on linguistics or NLP, such as

- *Language Files: Materials for an Introduction to Language and Linguistics* (12th ed.), edited by Vedrana Mihalicek and Christin Wilson (The Ohio State University Press, 2011).
- *Natural Language Understanding* by James Allen (Pearson, 1994).

Avoid descriptions of syntactic trees within the theory of generative grammar since their X-bar structure reflects hypotheses about the human language faculty that are not followed by natural language parsing technologies.

3. Explore how PP-attachments work using the search function of the online COCA corpus (Davies, 2008–) at https://www.english-corpora.org/coca/. Use the search string *_nn with a _nn*, which searches for [any-noun + *with a* + any-noun]. Notice the variety of eventualities that have to be handled by semantic analysis. Learn to use the various search strategies available in the interface, since we will suggest more exercises using this corpus in upcoming chapters.